On the Solution of Classical and Synchronized Routing Problems

Dissertation
zur Erlangung des Grades eines Doktors der
wirtschaftlichen Staatswissenschaften
(Dr. rer. pol.)
des Fachbereichs Rechts- und Wirtschaftswissenschaften
der Johannes Gutenberg-Universität Mainz

vorgelegt von
M.Sc. Jeanette Schmidt
in Mainz

im Jahre 2024

Tag der mündlichen Prüfung: 26.09.2024

Contents

List of Papers

- Jeanette Schmidt[1], Prof. Dr. Stefan Irnich[1] (2022). New Neighborhoods and an Iterated Local Search Algorithm for the Generalized Traveling Salesman Problem. *EURO Journal on Computational Optimization* **10**, 100029. `https://doi.org/10.1016/j.ejco.2022.100029`.

- Jeanette Schmidt, Prof. Dr. Christian Tilk[2], Prof. Dr. Stefan Irnich (2024). Using Public Transport in a 2-Echelon Last-Mile Delivery Network. *European Journal of Operational Research*, 317(**3**), 827–840. `https://doi.org/10.1016/j.ejor.2022.10.041`.

- Jeanette Schmidt, Prof. Dr. Christian Tilk, Prof. Dr. Stefan Irnich (2023). Exact Solution of the Vehicle Routing Problem with Drones. Technical Report LM-2023-03, Chair of Logistics Management, Johannes Gutenberg University Mainz, Mainz, Germany. *Submitted to Transportation Science.*

- Jeanette Schmidt (2024). The Vehicle Routing Problem with Drones and Time Windows: Minimizing Route Duration. Technical Report LM-2024-02, Chair of Logistics Management, Johannes Gutenberg University Mainz, Mainz, Germany. *Submitted to OR Spectrum.*

[1]Johannes Gutenberg-Universität Mainz, Lehrstuhl für BWL insb. Logistikmanagement, Jakob-Welder-Weg 9, D-55128 Mainz, Germany

[2]Universität Wien, Institut für Business Decisions and Analytics, Kolingasse 14–16, AT-1090 Wien, Österreich

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Combinatorial optimization problems deal with finding an optimal solution within a finite set of possible solution candidates. An optimal solution is one that minimizes or maximizes an objective function subject to specific side constraints that define the set of feasible solution candidates. Typically, combinatorial optimization problems can be represented by a graph, consisting of vertices and (weighted) edges, and can be formulated as *integer linear programs* (IPs, Schrijver, 2003).

One of the most widely studied classes of combinatorial optimization problems are *routing problems*. Routing problems aim to find an optimal route for one or multiple vehicles that starts and ends at a given location (e.g. a depot, a warehouse, a school, ...), visits several locations to perform a service task (e.g. at private households, shops, companies, bus stops, streets, medical facilities, ...), and meets further requirements depending on the considered problem (Bodin, 1975). They occur in many logistics applications, including the distribution and/or collection of goods, the transportation of handicapped persons, the routing of school buses, waste collection, street cleaning, and many others (Toth and Vigo, 2002).

A rather classical and perhaps the most well-known routing problem is the *Traveling Salesman Problem* (TSP, Gutin and Punnen, 2007). Given one vehicle and a set of vertices, the TSP asks for a weight-minimal Hamiltonian tour that visits all vertices exactly once. In the thesis at hand, we consider the *Generalized Traveling Salesman Problem* (GTSP, Fischetti *et al.*, 2007) which is a generalization of the TSP. In contrast to the TSP, the GTSP divides the set of vertices into disjoint subsets, so-called *clusters*, and aims to find a weight-minimal Hamiltonian tour that visits exactly one vertex of each cluster. Thus, it comprises two related decisions: Selecting the best vertex from each cluster and finding a Hamiltonian tour with minimal weights through the chosen subset of vertices. Due to the clustered vertex set, the GTSP can also be classified as a *generalized network design problem* (Pop, 2012).

Routing problems that consider multiple vehicles are called *Vehicle Routing Problems*. In the literature, the basic version is called *Capacitated Vehicle Routing Problem* (CVRP or VRP, Toth and Vigo, 2014). Herein, the service tasks consist of

the distribution of goods from a given starting point, mostly called a *depot*, to a set of customers. To fulfill these tasks, a fleet of homogeneous trucks with a known capacity is given. The CVRP aims to find a set of feasible routes at minimum cost, where each route visits a set of customers in such a way that each customer is visited exactly once, and its entire demand is delivered. A route is feasible if it starts and ends at the depot and if the truck's capacity is always respected (Irnich *et al.*, 2014). Since Dantzig and Ramser introduced the problem in 1959 (back then called *The Truck Dispatching Problem*) numerous problem extensions have been studied. A current and quite innovative research branch is dedicated to VRPs in urban synchromodal systems. In synchromodal delivery systems, goods are carried by a combination of different layers or transportation modes from suburban distribution centers to shops and private households in the city center. A key challenge in these networks is the interoperability of all distribution modes in a dynamic and time-constrained context. From a scientific point of view, this amounts to solving VRPs that strongly rely on different *synchronization constraints*, making them particularly challenging (Drexl, 2012; Soares *et al.*, 2024). In the thesis at hand, we tackle three such VRP variants. In the following, we briefly discuss their specific challenges:

At first, we introduce a multi-level VRP called *Last-Mile Delivery Problem with Scheduled Lines* (LMDPSL) that is a special case of the *Two-Echelon Vehicle Routing Problem* (Cuda *et al.*, 2015). In two-echelon systems, first-echelon vehicles carry goods from suburban distribution centers to so-called *satellites* that are located in the city center. Second-echelon vehicles collect the goods from the satellites and deliver them to the customers. In the LMDPSL, we consider the special case of this system, where the first echelon utilizes existing public services for passenger transport (such as metro lines, trams, or buses) and where dedicated stations are used as satellites. On the second echelon, so-called *city freighters*, pick up containers from the stations and deliver the contained goods to shops or private households. The difficulty of this problem lies in ensuring that each transport adheres to exact operation and load synchronization constraints, i.e., public transport vehicles and city freighters must achieve synchronization in time, space, and load to exchange goods.

Second, we consider the *Vehicle Routing Problem with Drones* (VRP-D), an extension of the CVRP in which the fleet of trucks is equipped with a single drone each. The idea is that each truck serves as a *mobile satellite* for its drone, i.e., the truck can serve customers on its own and –whenever it is beneficial– it can release a drone to serve another customer. Both truck and drone have to meet again after the drone finished its delivery. Compared to the CVRP, such a setting requires additional decisions about which type of vehicle serves which subset of customers (task synchronization) and where to release and meet the drone (space

synchronization).

Third, we investigate the *Vehicle Routing Problem with Drones and Time Windows* (VRP-DTW). It is a generalization of the VRP-D where the service at each customer must start within a predefined time interval, denoted as *time window*. Whenever a truck or a drone arrives too early at a customer, it must wait until the time window opens for service. Arriving at a customer when the time window is already closed is not allowed. Considering such timing aspects requires additional time synchronization.

1.1 Solution Methods

From a theoretical point of view, most combinatorial optimization problems are $\mathcal{NP}$-hard (Schrijver, 2003). This means, that it is likely that no polynomial-time algorithm exists to solve these problems efficiently. In practice, there exist two main techniques to solve combinatorial optimization problems: exact and heuristic algorithms. *Exact algorithms*, such as branch-and-bound (B&B), dynamic programming, or IP techniques (e.g., branch-and-price, branch-and-cut, or branch-price-and-cut (BPC)) guarantee to provide an optimal solution (if one exists). However, the runtime increases exponentially with the size of the instance. As a consequence, exact solution methods are mostly limited to small and medium-sized instances. To tackle larger instances of practical size, there is the need to develop other effective solution methods such as *(meta-)heuristics*. They may provide 'good' solutions within an adequate amount of computation time, but they do not guarantee optimality. Prominent examples are large neighborhood search (Ropke and Pisinger, 2006; Pisinger and Ropke, 2007), variable neighborhood search (VNS, Hansen and Mladenović, 2001), iterated local search (ILS, Lourenço *et al.*, 2003), and evolutionary algorithms (Eiben and Smith, 2015).

For solving the considered problems in this thesis, we employ both exact and heuristic solution approaches. All VRP variants are solved using BPC algorithms, while the GTSP is solved using an ILS metaheuristic. The basic principles of these algorithms are briefly discussed below.

Branch-Price-and-Cut Many combinatorial optimization problems can be formulated as a compact IP with a polynomial number of variables and constraints. However, these models often have a poor linear relaxation and/or a symmetric structure that causes the B&B algorithm to perform poorly. To overcome these shortcomings, Dantzig and Wolfe (1960) proposed a method to decompose an IP into a linear *master problem* (MP) and one or more *pricing problem(s)* by redefining the original variables. Although the resulting MP contains an exponential number of variables, it is not necessary to generate all of them explicitly. The

basic idea of *column-generation* (CG) is to work with a *restricted master problem* (RMP) that considers only a (small) subset of all variables. The solution of the RMP generates dual prices that are used within the pricing problem(s) to identify missing variables (so-called *columns*). More precisely, the pricing problems have to find at least one column with negative reduced cost that is added to the RMP or to prove that no such column exists. The solution of the linear relaxation of the MP is obtained by alternating between re-optimizing the RMP and solving the pricing problem(s) until no more columns with negative reduced cost exist.

If the MP contains integrality constraints on some of its variables, the CG procedure can be incorporated into a B&B algorithm, resulting in a *branch-and-price* algorithm (Barnhart *et al.*, 1998). The B&B algorithm relies on a systematical enumeration of possible solutions to find integer solutions. Herein, clever bounding and pruning strategies are used to reduce the search space (Dakin, 1965). In some cases, it can be beneficial to add *cutting planes* at some nodes of the search tree to strengthen its linear relaxation. This solution method is then called a *branch-price-and-cut* algorithm.

Besides often providing a stronger linear relaxation than a compact formulation of an IP, CG algorithms have two further advantages: (i) complicated constraints or even non-linearities that may be present in a compact formulation can be hidden in the pricing problem(s), and (ii) there often exist highly effective solution methods to solve the corresponding pricing problem(s) (Lübbecke and Desrosiers, 2005). For example, in VRPs the pricing problem is usually formulated as a *shortest path problem with resource constraints* (SPPRC, Irnich and Desaulniers, 2005). Even though the SPPRC is known to be $\mathcal{NP}$-hard (Dror, 1994), it can be solved by means of a dynamic-programming based labeling algorithm in pseudo-polynomial time (Irnich and Desaulniers, 2005). On the downside, CG algorithms have several disadvantages resulting from instability issues, such as (i) simplex-based column-generation is known for its poor convergence (denoted as *tailing-off effect*), (ii) generating steady RMP values for several iterations (denoted as *plateau effect*), (iii) generating irrelevant columns in the first iterations due to poor dual information (denoted as *heading-in effect*), and (iv) the oscillation of the dual prices (Vanderbeck, 2005).

Iterated Local Search ILS is a simple and generally applicable local search method. Its main idea is to conduct a randomized walk that is biased towards locally optimal solutions, rather than sampling all possible solution candidates (Lourenço *et al.*, 2003). More precisely, this walk is a stochastic local search (in the sense of Hoos and Stützle, 2005): Iteratively, a current locally optimal solution is randomly perturbed. The perturbed solution then undergoes a local search, and the resulting solution is tested to be accepted as the new current locally optimal

solution. These steps are repeated until a termination condition is met. The best-found solution is the output of the ILS.

The advantages of an ILS can be stated as follows: (i) it follows a simple principle, (ii) it is easy to implement, (iii) it is robust, and (iv) it can be highly effective. However, its effectiveness strongly depends on a wise implementation of its main components. For example, the *perturbation* step helps to escape from a local minimum and guides the search towards a region of the solution space that has not been explored yet. If the perturbation is too strong, it behaves like a random restart, if the perturbation is too small, the local search will fall back into the local optimum just visited before. Analogous, the *acceptance criterion* controls the balance between intensification and diversification of the search. In case no acceptance criterion is used, the ILS performs a random walk through the space of all locally optimal solutions. On the opposite, if only better solutions are accepted, intensification becomes too strong (Lourenço *et al.*, 2003).

1.2 Contribution and Outline

This thesis by publication comprises four articles. All of them have either been published in or submitted to scientific journals. The aim of this thesis is to contribute to the development of effective solution methods, i.e., exact and heuristic algorithms, for different routing problems. Further, we conduct several computational analyses to evaluate the algorithmic components and to give managerial insights. In the following, the structure of the thesis and the contributions of each paper are presented.

In Chapter 2, we present a basic and refined ILS to solve the GTSP. Therefore, we develop three new GTSP neighborhoods that allow the simultaneous permutation of the sequence of clusters and the selection of vertices per cluster. Despite their exponential size, these neighborhoods can be explored efficiently with a polynomial worst-case complexity. In addition, we combine our neighborhoods with established neighborhoods from the literature within a *variable neighborhood descent* (VND, Hansen and Mladenović, 2001). The VND is used within the local search part of the ILS. Although the basic ILS already generates reasonable results on standard benchmark instances, we refine the ILS to compete with current state-of-the-art GTSP solvers.

Chapter 3 introduces the LMDPSL and develops full-fledged BPC algorithms. The BPC algorithms allow the exact and heuristic solution of instances with up to 150 customers. Our main contribution is an extensive computational study of the LMDPSL considering possible scenarios of the public transport systems under various hierarchical objective functions. In addition, we analyze the trade-off between several key indicators (i.e., routing costs, the number of employed

city freighters, and the number of necessary trips) when different objectives are employed and the impact of limiting the capacity of public transport vehicles.

In Chapter 4, we present a tailored BPC algorithm to solve a variant of the VRP-D for two different objective functions: the minimization of the total routing cost and the sum of completion times over all routes. Our main contribution is to develop an implicit bidirectional labeling algorithm to effectively solve the SPPRC pricing problem. A particular novelty is that we apply two different merge procedures within the bidirectional labeling depending on the current truck-and-drone positions. Finally, we conduct several computational analyses regarding its algorithmic components, compare the two objectives as well as the impact of the drone's and truck's speeds and routing costs on the structure and quality of the resulting solutions.

In Chapter 5, we study the VRP-DTW. The introduction of time windows allows the investigation of another objective function: the minimization of the sum of durations over all routes. When minimizing the completion time of a route, truck and drone leave the depot at the earliest possible time to start the delivery process. In contrast, when minimizing the route duration, the departure time at the depot is not fixed. Both types of vehicles can leave at any time within the depot's time window. Our main contribution is the development of the first algorithm that allows the exact solution of the VRP-DTW with the objective to minimize the sum of durations over all routes. Since the synchronization of truck and drone with respect to a minimal route duration is a non-trivial task, we present a method to properly synchronize both types of vehicles. We conduct an extensive computational study that evaluates the algorithmic components and demonstrates the efficacy of the overall BPC algorithm. Finally, we provide managerial insights that analyze the impact of combined truck-and-drone routing, the impact of drone types with different flying ranges, and we compare the minimization of the total duration to the minimization of the completion time.

Finally, Chapter 6 summarizes and draws final conclusions.

Chapter 2

New Neighborhoods and an Iterated Local Search Algorithm for the Generalized Traveling Salesman Problem

Jeanette Schmidt, Stefan Irnich

Abstract

For a given graph with a vertex set that is partitioned into clusters, the generalized traveling salesman problem (GTSP) is the problem of finding a cost-minimal cycle that contains exactly one vertex of every cluster. We introduce three new GTSP neighborhoods that allow the simultaneous permutation of the sequence of the clusters and the selection of vertices from each cluster. The three neighborhoods and some known neighborhoods from the literature are combined into an effective iterated local search (ILS) for the GTSP. The ILS performs a straightforward random neighborhood selection within the local search and applies an ordinary record-to-record ILS acceptance criterion. The computational experiments on four symmetric standard GTSP libraries show that, with some purposeful refinements, the ILS can compete with state-of-the-art GTSP algorithms.

2.1 Introduction

For a given graph with a vertex set that is partitioned into clusters, the generalized traveling salesman problem (GTSP) is the problem of finding a cost-minimal cycle that contains exactly one vertex of every cluster. Formally, an instance of the symmetric GTSP is defined by an edge-weighted complete undirected graph $G = (V, E)$, where V denotes the set of vertices and E the set of edges. The vertices are partitioned into N non-empty disjoint subsets, denoted as *clusters* and indexed by $i \in I = \{1, 2, \ldots, N\}$, such that $V = \bigcup_{i \in I} V_i$. Note that clusters do not necessarily result from a geometric (i.e. distance or proximity based) grouping. Depending on the application, clusters may contain vertices that are 'far away' from each other.

For a vertex $v \in V$, let $i = [v] \in I$ be the index of the cluster to which the vertex belongs, i.e., $v \in V_{[v]}$. The edge set E comprises (unordered) pairs vw of vertices $v, w \in V$ for $[v] \neq [w]$, implying $vw \equiv wv$ and (symmetric) edge weights $c_{vw} = c_{wv}$. We use the symbols n for the cardinality of the vertex set and m_i for the size of the ith cluster, i.e., $m_i = |V_i|$ for all $i \in I$. A feasible solution to the GTSP is a cycle $x = (x_1, x_2, \ldots, x_N, x_1)$ with exactly one vertex per cluster, i.e., $[x_i] \neq [x_j]$ for all $i, j \in I, i \neq j$. Such a cycle is also denoted as a *G-tour*. The cost of the G-tour x is defined as $c(x) = \sum_{i \in I} c_{x_i x_{i+1}}$ (assuming $x_{N+1} = x_1$). The objective of the GTSP is to find a minimum-cost G-tour.

In this paper, we present an effective metaheuristic for the GTSP. The metaheuristic follows the principles of a clean *iterated local search* (ILS, Lourenço *et al.*, 2003). ILS is a stochastic local search (in the sense of Hoos and Stützle, 2005): Iteratively, a current solution is randomly perturbed, the perturbed solution undergoes a local search, and the resulting solution is tested to be accepted as the new current solution by comparing current and resulting solution (possibly taking into account the search history). These simple steps are repeated until a termination condition is met. The best seen solution is the output of the ILS.

The local search part of our ILS combines known and three new GTSP neighborhoods in a *variable neighborhood descent* (VND, Hansen and Mladenović, 2001) fashion. Our ILS performs a straightforward random neighborhood prioritization within the VND and applies an ordinary record-to-record acceptance criterion (Dueck, 1993).

One contribution of our work is the introduction of three new neighborhoods for the GTSP that allow the simultaneous permutation of the sequence of the clusters and the selection of vertices from each cluster. We show that despite the exponential size of the new neighborhoods, they can be explored efficiently: We present three neighborhood exploration algorithms, all with polynomial worst-case complexity, where two algorithms allow the complete exploration of the respective neighborhood so that a best improving neighbor (if existent) is found. For the third new neighborhood, a polynomial time heuristic neighborhood exploration

is presented. With these components, our basic ILS is already competitive with GTSP metaheuristics from the literature.

The second contribution is the refinement of the basic ILS regarding a reset mechanism to the best found solution, the neighborhood prioritization in the VND, the use of the GTSP-adapted Balas-Simonetti neighborhood (Balas and Simonetti, 2001) for improving high-quality solutions, and a modified cooling schedule for the record-to-record acceptance criterion. Moreover, we present a straightforward grouping scheme for GTSP instances that allows us to control which algorithmic refinements to apply to which type of GTSP instance. In computational experiments on standard GTSP benchmark instances, we show that the resulting refined ILS produces very reasonable solutions.

The remainder of the paper is structured as follows. In Section 2.2, we review the pertinent GTSP literature. We describe GTSP neighborhoods and corresponding efficient neighborhood exploration algorithms used in our ILS in Section 2.3. The overall ILS is presented in Section 2.4 together with computational results obtained with the basic ILS implementation. Refinements of the ILS tailored to specific groups of GTSP instances are presented in Section 2.5. Here, we introduce measures that decide which ILS variant to apply for a given GTSP instance. The paper closes with final conclusions drawn in Section 2.6.

2.2 Literature Review

A variety of combinatorial optimization problems can be modeled as GTSPs, or contain the GTSP as a subproblem. Among them are location routing problems, material-flow system design problems, post-box collection problems (Laporte *et al.*, 1996) as well as the routing of clients through welfare agencies (Saksena, 1970), and computer file sequencing (Henry-Labordere, 1969).

Since its introduction in the late sixties/early seventies (Henry-Labordere, 1969; Saksena, 1970) a lot of attention has been paid to solving the GTSP. One group of solution approaches relies on the fact that every GTSP instance can be transformed into an equivalent *traveling salesman problem* (TSP) instance. Thus, different reduction algorithms were developed, e.g., by Noon and Bean (1993); Laporte and Semet (1999); Ben-Arieh *et al.* (2003). The resulting TSP instances can then be solved with a TSP solver, either heuristically or, if not too large, exactly. For example, Helsgaun (2015) combined the Noon-Bean reduction with the Lin-Kernighan-Helsgaun algorithm (LKH, Helsgaun, 2000) into a powerful GTSP solver.

Several exact approaches for the GTSP have shown very good results: The branch-and-cut algorithm of Fischetti *et al.* (1997) solves symmetric GTSP instances with up to 89 clusters and 442 vertices to optimality. A Lagrangian based

approach to solve asymmetric GTSP instances was developed by Noon and Bean (1991). Their results show success on a range of randomly generated instances with up to 100 vertices. Solving larger instances to proven optimality can still be a very hard task nowadays.

Certainly, large-scale GTSP instances require heuristic solution approaches. Many different approaches have been published, ranging from simple tour construction heuristics (e.g., Noon, 1988) to more involved and rather effective metaheuristics. Table 2.1 provides an overview of the pertinent GTSP publications. Almost

(Meta)heuristic	Reference(s)
Tour construction heuristic	Noon (1988); Fischetti *et al.* (1997)
Composite heuristic	Renaud and Boctor (1998)
Adaptive Large Neighborhood Search	Smith and Imeson (2017)
Ant Colony Optimization	Yang *et al.* (2008); Reihaneh and Karapetyan (2012); Pintea *et al.* (2017)
Genetic/memetic Algorithm	Snyder and Daskin (2006); Silberholz and Golden (2007); Gutin *et al.* (2008); Gutin and Karapetyan (2010); Bontoux *et al.* (2010)
Lin-Kerninghan adaption	Karapetyan and Gutin (2011)
Multi-start method	Cacchiani *et al.* (2011)
Particle Swarm Optimization	Tasgetiren *et al.* (2007)
Variable Neighborhood Search	Hu and Raidl (2008)

Table 2.1: Selected literature on (meta)heuristics for the GTSP.

all listed metaheuristics have at least one thing in common: They contain local optimization techniques that run one or more local/neighborhood-search heuristics to improve a given solution (an exception is Pintea *et al.*, 2017). A first approach, called RP2 and nowadays known as *cluster optimization* (CO), was invented by Fischetti *et al.* (1997). CO determines a globally best vertex selection for a given and fixed cluster sequence. This is done by constructing and solving a shortest-path problem in a layered network (a detailed description follows in Section 2.3.3). Because of its efficiency, CO can be found in many different GTSP algorithms, e.g., of Cacchiani *et al.* (2011); Reihaneh and Karapetyan (2012); Smith and Imeson (2017).

Another straightforward approach is to use a standard TSP improvement procedure, such as the 2-Opt, 3-Opt, and k-Opt (for $k \geq 4$) heuristics (Lin, 1965). Such

improvement procedures have been used in Yang *et al.* (2008) and Bontoux *et al.* (2010). CO and k-Opt are two extremes of GTSP improvement procedures, where the first only optimizes the vertex selection per cluster and the second only optimizes the sequence in which the clusters are visited. The other decision remains fully fixed in both cases.

The disadvantage of the approaches that work only on one type of improvement method is that they are too myopic. Note that often a local improvement requires a modification in both the vertex selection and cluster sequence. In order to overcome this disadvantage, several researchers have developed GTSP-tailored neighborhoods so that both types of decisions can change within one move. One of them is the RP1 procedure by Fischetti *et al.* (1997) based on 2-Opt and 3-Opt exchanges. Renaud and Boctor (1998) introduced the G2-Opt, G3-Opt, and G-Opt heuristics, which reverse tour segments and determine an optimal vertex selection via the CO algorithm. Finally, Renaud and Boctor combine G-Opt and G2-Opt to a powerful improvement part of their composite heuristic. Some years later, Hu and Raidl (2008) revisited the G2-Opt heuristic and refined the vertex selection process for a given cluster sequence. In detail, they apply an incremental bidirectional shortest-path calculation to save computation time. Another interesting method is to adapt the classical TSP 2-Opt as suggested by Gutin and Karapetyan (2009).

Special k-Opt heuristics, like swap moves (special 4-Opt) and relocation moves (special 3-Opt) are well known from the TSP. They were also adapted for the GTSP, e.g., different types of swap moves can be found in (Gutin *et al.*, 2008; Gutin and Karapetyan, 2010). Adapted relocation moves, also known as insert or node shift moves, are used in (Snyder and Daskin, 2006; Silberholz and Golden, 2007; Tasgetiren *et al.*, 2007; Gutin *et al.*, 2008; Gutin and Karapetyan, 2010; Smith and Imeson, 2017). Bontoux *et al.* (2010) search for best relocation moves with a special dynamic-programming algorithm that is inspired by the work of Feillet *et al.* (2004) on the elementary shortest-path problem with resource constraints.

As the LKH algorithm is still the state-of-the-art heuristic for the classical TSP, it can also be used in the GTSP context. For example, Bontoux *et al.* (2010) use the original version within their memetic algorithm, while Karapetyan and Gutin (2011) developed different Lin-Kernighan adaptations for the GTSP.

It is beyond the scope of this paper to provide a comprehensive classification of all known and new neighborhoods. The article by Karapetyan and Gutin (2012) provides such a synopsis. Furthermore, they explain efficient neighborhood exploration algorithms for all neighborhoods.

The GTSP remains an active research object with recent publications by El Krari *et al.* (2021, 2020), and Ren *et al.* (2020). After all, the memetic algorithm *GK* (Gutin and Karapetyan, 2010), the Lin-Kernighan-Helsgaun algorithm *GLKH*

(Helsgaun, 2015) and the adaptive large neighborhood search *GLNS* (Smith and Imeson, 2017) are still the best-performing state-of-the-art GTSP solvers. Since we will evaluate our algorithm against these GTSP solvers, we briefly outline their main algorithmic components in comparison to our ILS in Table 2.2.

Algorithm	Metaheuristic paradigm	Neighborhoods used in local search part	Acceptance criterion	Specific components
GK	Genetic Algorithm (GA)	(k-neighbor) Swap, (direct) 2-Opt, CO, Relocation+	only improving solutions	well-fitted genetic operators, efficient termination condition
GLKH	Local Search	LKH, CO	only improving solutions	Noon and Bean (1993) reduction, LKH algorithm
GLNS	Adaptive Large Neighborhood Search (ALNS)	Relocation+, CO	Metropolis	unified insertion/removal mechanism, increased randomness during insertions and removals, warm starts, additive noise on insertion costs, improved adaptive weights
our algorithm	Iterated Local Search (ILS)	2-Opt, 3-Opt, Double Bridge, Relocation+, Swap+, CO, Balas-Simonetti for GTSP, adapted Gutin neighborhood, String Relocation+	record-to-record criterion	efficient local search part with new GTSP neighborhoods, refined ILS tailored to specific groups of GTSP instances

Table 2.2: Main algorithmic components of the state-of-the-art GTSP solvers.

2.3 Neighborhoods and Efficient Neighborhood Exploration

In this section, we describe the neighborhoods used in our ILS and efficient neighborhood exploration algorithms. We distinguish between traditional TSP neighborhoods (Section 2.3.1), polynomially-sized GTSP neighborhoods (Section 2.3.2), and exponentially-sized GTSP neighborhoods (Section 2.3.3). We introduce three exponentially-sized GTSP neighborhoods that have not yet been considered in other works. Moreover, we present effective neighborhood exploration algorithms for them. For a GTSP-tailored version of the Balas-Simonetti neighborhood and an adapted version of the Gutin neighborhood, the algorithms are polynominaly bounded but guarantee that a best improving neighbor is found. The third neighborhood is String Relocation+, for which we develop an effective neighborhood exploration heuristic.

2.3.1 TSP Neighborhoods

We first consider (symmetric) TSP neighborhoods for the GTSP. Moves of this type do not change the selection of vertices from each cluster.

2-Opt and 3-Opt The k-Opt neighborhoods have been made popular by the work of Lin (1965). The current TSP solution x is divided into $k \geq 2$ segments that can be inverted and permuted. The result is a neighborhood of size $\mathcal{O}(N^k)$.

We apply a cost-based pruning to accelerate the neighborhood exploration based on the gain criterion of Lin and Kernighan (1973). This technique is known under different names as fixed radius near neighbor search (Bentley, 1992), fixed radius search (Hoos and Stützle, 2005, p. 373f), or sequential search (Irnich *et al.*, 2006). For the GTSP, a prerequisite is to have, for each vertex $v \in V$, an ordered *neighbor list* of all other vertices sorted by increasing distance. For a G-tour $x = (x_1, x_2, \ldots, x_N, x_1)$, the complete neighbor lists of the vertices $x_1, x_2, \ldots, x_N$ should be thinned out so that they comprise only vertices from $\{x_1, x_2, \ldots, x_N\}$ and no vertices from $V \setminus \{x_1, x_2, \ldots, x_N\}$. This preparatory step takes $\mathcal{O}(nN)$ time and space. The actual neighborhood exploration is then drastically accelerated on average (Hoos and Stützle, 2005, p. 373f).

Double-Bridge The double-bridge move (Johnson and McGeoch, 1997) is a special 4-Opt move that divides the given tour into four non-empty segments $x = (A, B, C, D)$ that are permuted into $x' = (A, D, C, B)$. Glover (1996) has shown that, for a given TSP tour x, the double-bridge neighborhood, which comprises $\mathcal{O}(N^4)$ tours, can be explored completely and efficiently in $\mathcal{O}(N^2)$ time and space, with the help of a dynamic program. We also use this indirect neighborhood exploration technique.

2.3.2 Polynomial GTSP Neighborhoods

Next we consider neighborhoods that are inspired by TSP neighborhoods but allow to modify the selection of a vertex for at least one cluster.

Relocation+ The classical TSP relocation move selects a vertex x_i in the given tour x, removes it from its current position, and inserts it between two other consecutive vertices x_j and x_{j+1}. For a G-tour x, it is now allowed to replace x_i by a vertex x'_i of the cluster $V_{[x_i]}$. Hence,

$$x = (x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_j, x_{j+1}, \ldots, x_N, x_1)$$

is altered into

$$x' = (x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_j, x'_i, x_{j+1}, \ldots, x_N, x_1)$$

with $[x'_i] = [x_i]$. The size of the neighborhood is $\mathcal{O}(nN)$.

Swap+ The swap neighborhood of a TSP tour selects two non-neighboring vertices and swaps them. For a G-tour x, the two swapped vertices x_i and x_j can be replaced by other vertices $x'_i \in V_{[x_i]}$ and $x'_j \in V_{[x_j]}$. Hence, the given

$$x = (x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_N, x_1)$$

is modified into

$$x = (x_1, x_2, \ldots, x_{i-1}, x'_j, x_{i+1}, \ldots, x_{j-1}, x'_i, x_{j+1}, \ldots, x_N, x_1)$$

with $[x'_i] = [x_i]$ and $[x'_j] = [x_j]$. The size of this GTSP neighborhood is $\mathcal{O}(n^2)$.

2.3.3 Exponential GTSP Neighborhoods

The following exponential GTSP neighborhoods can all be explored with an indirect search method such that the computational effort is polynomially bounded.

Cluster Optimization CO takes a given G-tour $x = (x_1, x_2, \ldots, x_N, x_1)$ and replaces it by a shortest G-tour $x' = (x'_1, x'_2, \ldots, x'_N, x'_1)$ with $[x'_i] = [x_i]$ for all $i \in I$ (Fischetti *et al.*, 1997). Hence, the sequence of the clusters is kept, however, different vertices in all clusters can be selected. This is a neighborhood of size $\mathcal{O}(\prod_{i \in I} m_i)$.

A best neighbor solution results from solving one or several shortest-path problems in the layered graph with clusters as layers, see Figure 2.1. Consecutive layers connect all vertices $x'_i \in V_{[x_i]}$ with all vertices $x'_{i+1} \in V_{[x_{i+1}]}$ for $i \in I$ with arc weights $c_{x'_i, x'_{i+1}}$. While in Figure 2.1 the first cluster is a singleton set, the general case requires the solution of one shortest-path problem for each possible $x'_1 \in V_{[x_1]}$ as a start and end vertex.

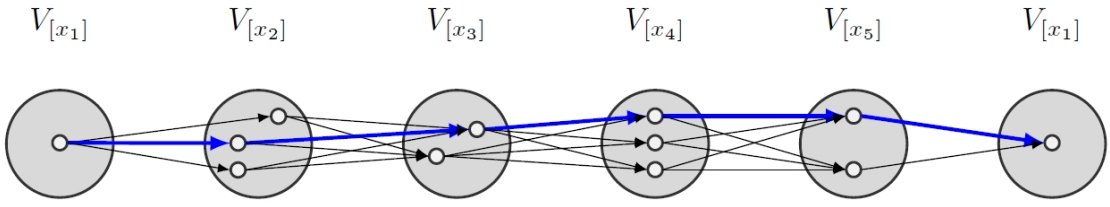


Figure 2.1: Layered network for CO.

Karapetyan and Gutin (2012) suggest several techniques that aim at lowering the worst-case time complexity of the computation. First, every G-tour can be rotated

so that w.l.o.g. $m_1 = \min_{i \in I} m_i$, i.e., the first cluster has minimum cardinality. Karapetyan and Gutin describe further implementation improvements such as the reduction of the first cluster (exploiting that the clusters $V_{[x_2]}$ and $V_{[x_N]}$ are known) and optimizing the dynamic-programming calculation order. They prove that CO can be searched efficiently in $\mathcal{O}(n \cdot \min_i m_i \cdot \max_i m_i)$ time. As the (practical) impact of the two last techniques is relatively minor in our context, we only rotate the G-tour and use a first cluster of minimum cardinality.

Balas-Simonetti for the GTSP In this section, we present a new neighborhood for the GTSP that is the synthesis of CO and the Balas-Simonetti neighborhood originally introduced for the TSP and TSP with time windows (Balas, 1999; Balas and Simonetti, 2001). We describe the TSP case first, before we provide details about the synthesis.

For a given integer $k \geq 2$, the Balas-Simonetti (BS) neighborhood $\mathcal{N}_k^{BS}$ allows the restricted permutation of the vertices relative to a given tour or path. More precisely, for a given tour $x = (x_1, x_2, \ldots, x_N, x_1)$ another tour $x' = (x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(N)}, x_{\pi(1)})$ defined by a permutation π on $\{1, 2, \ldots, N\}$ is in the neighborhood, i.e., $x' \in \mathcal{N}_k^{BS}(x)$, if

$$i + k \leq j \qquad \text{implies} \qquad \pi(i) < \pi(j) \qquad\qquad \text{for all } i, j \in \{1, 2, \ldots, N\}.$$

A larger value of k offers more flexibility so that the neighborhoods are nested, i.e., $\mathcal{N}_k^{BS} \subset \mathcal{N}_{k+1}^{BS}$ for all $k \geq 2$.

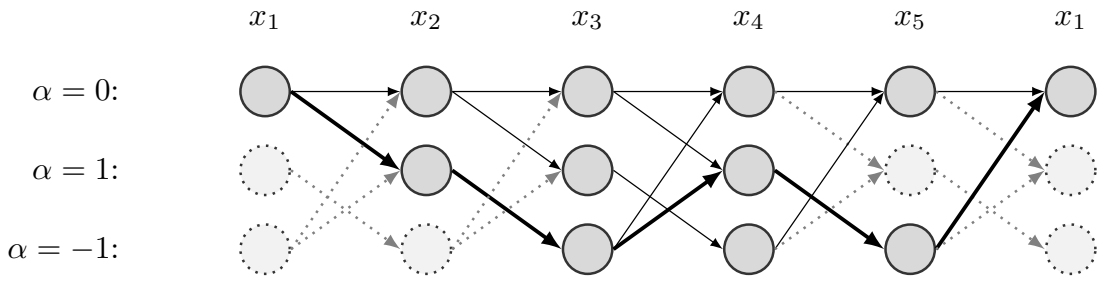


Figure 2.2: Auxiliary network G_k for $k = 2$.

A least-cost neighbor can be determined by solving a shortest-path problem in an auxiliary network G_k. Figure 2.2 shows the auxiliary network G_k for $k = 2$, where layers run horizontally and states vertically. In general, the auxiliary network G_k is a layered network with $N + 1$ layers $L_1, L_2, \ldots, L_N, L_{N+1}$ (assuming $L_{N+1} = L_1$), one for each position $(1, 2, \ldots, N, 1)$ of the given tour. All layers are identical. Each layer comprises exactly $(k + 1)2^{k-2}$ states (three states per layer for $k = 2$; $N + 1 = 5 + 1 = 6$ layers) partially describing the permutation π. To this end,

the states in each layer have an associated value α depicted left to the states of a row in Figure 2.2. Moreover, also two consecutive layers L_i and L_{i+1} induce identical subgraphs $G_k[L_i \cup L_{i+1}]$ for all $i \in \{1, 2, \ldots, N\}$ (for $k = 2$, each subgraph comprises six $[= 3 \cdot 2]$ vertices). The number of arcs of $G_k[L_i \cup L_{i+1}]$ is bounded by $k(k+1)2^{k-2}$. For $k = 2$ in Figure 2.2, there are five ($\leq 2 \cdot 3 \cdot 2^0 = 6$) arcs connecting two consecutive layers.

The point is now that every 0-0'-path in G_k describes exactly one permutation π and therefore the neighbor x', where $0 \in L_1$ and $0' \in L_{N+1}$ are specific null states in the layers (depicted on top in each layer in Figure 2.2, associated with $\alpha = 0$). A state with value α in ith layer refers to vertex $x_{i+\alpha}$. For example, the path depicted on top passing all vertices with values $\alpha = 0$ (using only the horizontally drawn arcs in Figure 2.2) represents the neighbor $x' = x = (x_1, x_2, \ldots, x_5, x_1)$. The path drawn in bold represents the neighbor $x' = (x_{1+0}, x_{2+1}, x_{3+(-1)}, x_{4+1}, x_{5+(-1)}, x_{1+0}) = (x_1, x_3, x_2, x_5, x_4, x_1)$ of x.

Accordingly, if arcs $(v, w) \in G_k[L_i \cup L_{i+1}]$ with values α_v for the state $v \in L_i$ and α_w for the state $w \in L_{i+1}$ are equipped with the cost $c_{x_{i+\alpha_v}, x_{x+1+\alpha_w}}$, the length of any 0-0'-path is identical to the length of the represented tour x'. Hence, solving a shortest-path problem between 0 and $0'$ in G_k provides a least-cost neighbor of x.

Some remarks are due:

- Since the network G_k is acyclic, a pulling or reaching type of dynamic-programming (DP) labeling approach for solving the shortest-path problem has a complexity proportional to the total number of arcs, i.e., $\mathcal{O}\left(N\,k(k+1)2^{k-2}\right)$. In particular, for a fixed k, the complexity is linear in the tour length N.
- Our sparse representation of the auxiliary network in the DP stores distance labels for all states. For the arcs, it suffices to only store the structure of $G_k[L_1 \cup L_2]$, because all consecutive layers are connected in the same way. Arc costs are computed on the fly.
- Some states in the first layers and in the last layers are irrelevant, because they are unreachable from 0 and 0' (backwardly). For the above-sketched implementation of the DP, the unreachable states pose no difficulty.

We now combine CO and BS into a new neighborhood that simultaneously permutes the order of the clusters (via BS) and allows to select alternative vertices from the permuted clusters. Figure 2.3 visualizes the idea for a given G-tour $x = (x_1, x_2, \ldots, x_N, x_1)$. Meta-states (big circles) represent the clusters that are initially sorted into the sequence $(V_{[x]_1}, V_{[x_2]}, \ldots, V_{[x]_N}, V_{[x_{N+1}]})$ (assuming $V_{[x_{N+1}]} = V_{[x_1]}$). Note the similarity between the states in Figure 2.2 and meta-states in Figure 2.3. However, meta-states are not part of the neighborhood's network G_k but are depicted only for the purpose of explanation.

In Figure 2.3, the different number of states and their graphical positioning helps to distinguish between different clusters. For example, $V_{[x_1]}$ comprises one state, $V_{[x_2]}$ three states, and $V_{[x_3]}$ two states. Both $V_{[x_2]}$ and $V_{[x_4]}$ contain three states, but are depicted differently.

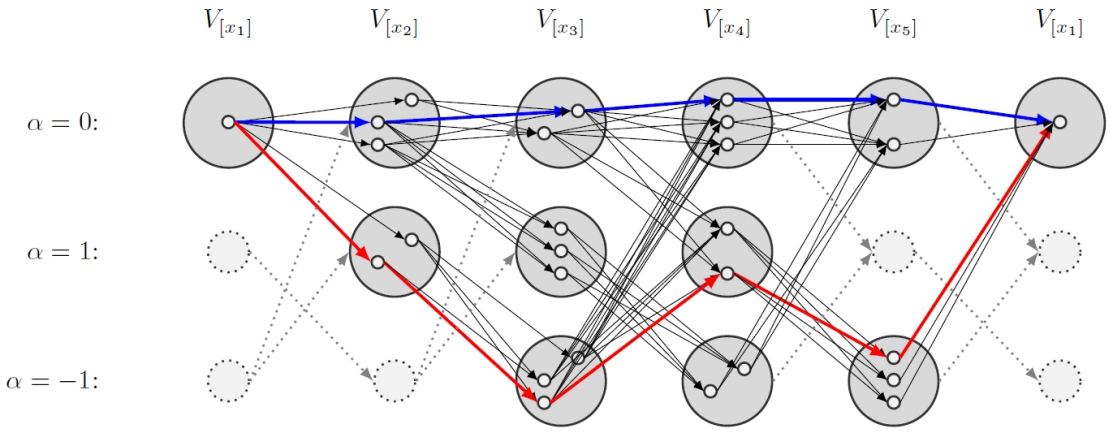


Figure 2.3: Auxiliary GTSP network G_k for $k = 2$.

All states of a meta-state at the ith layer are connected with all states of a meta-state at the $(i+1)$th layer (for $i \in \{1, 2, \ldots, N\}$) if and only if there is a corresponding arc in the original auxiliary network G_k. These complete connections between meta-states of consecutive layers are similar to the arcs in the CO network. Indeed, for $k = 1$ (note that in this case the BS neighborhood of the TSP degenerates to $\{x\} = \mathcal{N}_1^{BS}(x)$) the CO network shown in Figure 2.1 is identical to G_1. Note that the upper row of meta-states in Figure 2.3 is identical to the layered network for CO depicted in Figure 2.1. Indeed, all states in the upper row have $\alpha = 0$, i.e., the sequence of these states is not permuted at all. Any path through the meta-states with $\alpha = 0$ in Figure 2.3 stands for a selection of vertices from the currently given sequence of clusters (the path depicted in blue is one example). These are exactly the decision made by choosing an neighbor from the CO neighborhood. On the contrary, any 0-0'-path in Figure 2.3 through at least one meta-state with $\alpha \neq 0$ permutes the given sequence of clusters. The 0-0'-path depicted in red induces the new sequence $(V_{[x_1]}, V_{[x_3]}, V_{[x_2]}, V_{[x_5]}, V_{[x_4]}, V_{[x_1]})$ of the clusters.

The new GTSP neighborhood $\mathcal{N}_k^{BS}(x)$ is huge having (at least) $(k/e)^{N-1} \cdot \prod_{i \in I} m_i$ elements, where the first term bounds the number of different permutations from below (Theorem 7, Gutin *et al.*, 2007, assuming $N \geq k(k+1)$) and the second term comes from the CO analysis.

Gutin Neighborhood The assignment neighborhood of the TSP (Gutin *et al.*, 2007) first chooses a set $Z \subset \{1, 2, \ldots, N\}$ such that x_i and x_j for $i, j \in Z, i \neq j$ are non-adjacent in the given tour $x = (x_1, x_2, \ldots, x_N, x_1)$. The vertices $\{x_i : i \in Z\}$ can now be removed from x and be reinserted into the void positions one-to-one. For example, the subset $Z = \{2, 4, 7\}$ allows for the tour $x = (x_1, x_2, \ldots, x_7, x_1)$ exactly six reinsertions, where one of the resulting neighbors is $x' = (x_1, x_4, x_3, x_7, x_5, x_6, x_2, x_1)$. In general, the neighborhood $\mathcal{N}_Z^{Gutin}(x)$ comprises $|Z|!$ elements and a best neighbor can be identified in $\mathcal{O}(|Z|^3)$ by solving an assignment problem.

Our adaptation of the Gutin neighborhood for the GTSP works as follows: Let a G-tour $x = (x_1, x_2, \ldots, x_N, x_1)$ be given. First, we determine the set Z with a randomized heuristic. Initially we set $Z = \varnothing$. Iterating over $i = 1, 2, \ldots, N$, we first test whether $i - 1$ has been chosen. If $i - 1 \in Z$, we skip i (for $i = N$ we also test whether $1 \in Z$) and iterate. Otherwise, we toss a coin to decide whether $i - 1$ should be included into Z (with probability 0.5) or not, and iterate. Note that the expected length of a non-movable segment (between two consecutive elements of Z) is, therefore, $1 + 1/2 + 1/4 + 1/8 + \cdots = 2$. Hence, Z comprises $N/3$ elements on average.

Second, when computing the insertion cost for moving vertex $x_i, i \in Z$, into the void position $j \in Z$, we allow that x_i is replaced by a best $x_i' \in V_{[x_i]}$. Accordingly, we compute the insertion cost as

$$a_{i,j} := \min_{x_i' \in V_{[x_i]}} \left(c_{x_{j-1}, x_i'} + c_{x_i', x_{j+1}} \right).$$

with the convention $x_0 = x_N$ and $x_{N+1} = x_1$. In this way, the new GTSP neighborhood $\mathcal{N}_Z^{Gutin}(x)$ can simultaneously change the ordering of the clusters and the choice of cluster representatives.

The computational effort for determining a best neighbor in $\mathcal{N}_Z^{Gutin}(x)$ is bounded by $\mathcal{O}(nN + N^3)$, where the first term results from the computation of the insertion cost and the second from the exact solution of the assignment problem over $(a_{i,j})_{i,j \in Z}$.

String Relocation+ We describe now the new string relocation neighborhood $\mathcal{N}_L^{SR}$ for $L \geq 1$, which is a generalization of Relocation+ described in Section 2.3.2. Instead of moving a single vertex, a string of length up to L is removed from its current position and inserted into another position allowing different cluster representatives. Formally, a G-tour $x = (x_1, x_2, \ldots, x_N, x_1)$ is given. The string $(x_i, x_{i+1}, \ldots, x_{i+k})$ to be removed is defined by $i \in I$ and an integer k, $1 \leq k \leq L$ (assuming that $x_{i+p} = x_{i+p-N}$ for $2N > i + p > N$). It can be replaced by a string $(x_i', x_{i+1}', \ldots, x_{i+k}')$ with $[x_i'] = [x_i], [x_{i+1}'] = [x_{i+1}], \ldots, [x_{i+k}'] = [x_{i+k}]$. The new

string is then inserted between x_j and x_{j+1} for some $j \in I$ in the given G-tour. The neighbor solution is:

$$x' = (x_1, x_2, \ldots, x_{i-1}, x_{i+k+1}, \ldots, x_j, x'_i, x'_{i+1}, \ldots, x'_{i+k}, x_{j+1}, \ldots, x_N, x_1)$$

The neighborhood $\mathcal{N}_L^{SR}(x)$ comprises $\mathcal{O}(N^2 L m_{\max}^{L+1})$ elements, where $m_{\max} = \max_i m_i$.

Algorithm 1: Heuristic to explore $\mathcal{N}_L^{SR}(x)$.

```
   Input: x, L
1  for i ∈ I do
2      for x'_i ∈ V_[x_i] do
3          Let string ← (x'_i)
4          for k = 1, 2, ..., L do
5              Let x'_{i+k} ← MDE(x'_{i+k-1}, [x_{i+k}])
6              Let string ← (string, x'_{i+k})
7              for j ∈ I \ {i − 1, i, i + 1, ..., i + k, i + k + 1} do
8                  if Improving then
9                      Let i* ← i, j* ← j, string* ← string
   Output: i*, j*, string*
```

To keep the computational effort manageable, we explore $\mathcal{N}_L^{SR}(x)$ with the following heuristic. Beforehand, only once per GTSP instance, we compute and store in a lookup table the following information for each vertex w and each $i \in I$ with $i \neq [w]$: The element $u = MDE(w, i) \in V_{[i]}$ is the vertex with minimum distance to w (MDE, minimum distance element), i.e., $c_{wu} = \min_{v \in V_{[i]}} c_{wv}$ (ties are broken arbitrarily). The precomputation of the MDE values takes $\mathcal{O}(n^2)$ time.

The exploration heuristic for $\mathcal{N}_L^{SR}(x)$ is presented with the pseudo-code in Algorithm 1. In Step 1, we determine the first vertex x_i and the starting position i of the string that is relocated. With the loop in Step 2, we consider all possible replacements of x_i by an x'_i of the same cluster. The loop in Step 4 determines the length k of the string. The following replacements of x_{i+k} by x'_{i+k} are then looked up with the help of the auxiliary function MDE, i.e., the following replacements are heuristically chosen as close as possible to the preceding and last chosen vertex x'_{i+k-1} (Step 5). Thanks to the lookup table, Step 5 takes only constant time $\mathcal{O}(1)$. The result is that the string $(x_i, x_{i+1}, \ldots, x_{i+k})$ of length k is possibly replaced by the string $(x'_i, x'_{i+1}, \ldots, x'_{i+k})$. The insertion position j is determined in the loop in Step 7. Finally, the resulting move is completely determined now so that the cost of the move and possible improvement can be checked (Step 8).

The overall time complexity of the exploration heuristic is bounded by $\mathcal{O}(NLn)$.

2.4 Basic Iterated Local Search

In this section, we propose our basic metaheuristic algorithm based on ILS (Lourenço *et al.*, 2003) using a random VND (Subramanian *et al.*, 2010) as a local-search component. More precisely, given a current feasible G-tour x, the algorithm alternates between a local search starting from x using multiple neighborhoods and ending at a joint local minimum x^*, and a perturbation step. So every time the local search is trapped in a local minimum x^*, ILS perturbs it and starts a new local search based on this modified solution x'. As a consequence, ILS does a randomized walk in the space of all joint local minima.

Moreover, to better guide the search to more promising solutions, an acceptance criterion can be used. It decides whether the previous local minimum or locally optimal solution x^* is accepted as the new current solution. Hence, the acceptance criterion controls the balance between intensification and diversification.

In the following, we will discuss the implementation of the main functions considered in the basic ILS (Section 2.4.1), introduce the GTSP benchmark set (Section 2.4.2), and evaluate the basic ILS by presenting computational results (Section 2.4.3).

2.4.1 Implementation of the main ILS Functions

To achieve high performance, the four main functions *Initial Solution Construction*, *Local Search*, *Perturbation*, and *Acceptance Criterion* have to be tailored to the needs of the GTSP. In the following, we discuss how we implemented these main functions. Furthermore, Algorithm 2 shows how they are finally combined into the basic ILS metaheuristic.

Initial Solution Construction Several heuristics can be used to obtain a feasible starting solution for the GTSP. Most of them are derived by simple tour construction heuristics for the classical TSP. As an example, Fischetti *et al.* (1997) adapted the well-known TSP insertion heuristics to obtain a feasible G-tour. Based on these, Smith and Imeson (2017) provide a unified insertion procedure that contains three insertion procedures as special cases.

We create an initial solution with a random insertion procedure similar to the insertion heuristics presented in (Fischetti *et al.*, 1997). Initially, we randomly pick a vertex $x_1 \in V$ and add it as the start and end vertex of the subtour (x_1, x_1). In each iteration, the subtour $(x_1, x_2, \ldots, x_k, x_{k+1} = x_1)$ (with $k < N$) is then enlarged by randomly choosing one of the non-visited clusters $V_{[i]}$ and inserting the vertex $y \in V_{[i]}$ into the subtour that minimizes the insertion cost, i.e., $y = \arg\min_{x \in V_{[i]}, 1 \leq j \leq k} \{c_{x_j x} + c_{x x_{j+1}} - c_{x_j x_{j+1}}\}$. The procedure stops when the G-tour visits all N clusters.

Algorithm 2: Iterated Local Search (ILS) with record-to-record travel acceptance criterion.

Input: acceptance parameter $\epsilon > 0$, cooling rate $0 < h < 1$

1 $x_{init} \leftarrow$ Initial Solution Construction()
2 $x \leftarrow x_{ILS} \leftarrow$ Local Search(x_{init})
3 **repeat**
4 $\quad x' \leftarrow$ Perturbation(x)
5 $\quad x^* \leftarrow$ Local Search(x')
$\quad$ `/* Test Acceptance Criterion */`
6 $\quad$ **if** $c(x^*) < c(x) \quad or \quad c(x^*) \leq (1+\epsilon) \cdot c(x_{ILS})$ **then**
7 $\quad\quad x \leftarrow x^*$
8 $\quad$ **if** $c(x^*) < c(x_{ILS})$ **then**
9 $\quad\quad x_{ILS} \leftarrow x^*$
10 $\quad$ **if** cooling update condition fulfilled **then**
11 $\quad\quad \epsilon \leftarrow \epsilon \cdot h$
12 **until** time limit reached

Output: x_{ILS}

Local Search We apply a random VND with all neighborhoods described in Section 2.3. The idea is that the resulting VND deeply explores the solution space with the combination of traditional TSP neighborhoods (Section 2.3.1), TSP-inspired GTSP neighborhoods (Section 2.3.2), and exponentially-sized GTSP neighborhoods (Section 2.3.3). We choose a first improvement pivoting strategy for all neighborhoods except those explored implicitly with an optimization algorithm (double-bridge, CO, BS, Gutin, and SR neighborhood). Moreover, pre-tests have shown that the BS neighborhood should be used with $k \leq 3$, because the state space for larger k grows considerably, making labeling prohibitively slow (compared to the other exploration algorithms). Note that we consider BS neighborhoods with different k values as different neighborhoods. Finally, the maximum string length for the SR neighborhood is set to $L = 4$.

Every time the local search (Step 5 in Algorithm 2) is invoked, the random VND randomly selects new priorities for all available neighborhoods $\mathcal{N}^1, \mathcal{N}^2, \ldots, \mathcal{N}^\ell$. With chosen priorities (we assume that neighborhoods are reshuffled accordingly), it works like any other standard VND, i.e., it always starts the neighborhood exploration with $\mathcal{N}^1$, continues with $\mathcal{N}^2$ etc., and finally ends with $\mathcal{N}^\ell$. More precisely: If, for the current solution x', an improving neighbor $x^* \in \mathcal{N}^1(x')$ is found, this neighbor x^* is returned, and it undergoes the acceptance criterion (Step 6). Otherwise, $\mathcal{N}^2(x')$ is explored next. In general, if an improving neighbor

$x^* \in \mathcal{N}^p(x')$ is found in the pth neighborhood, it is returned, and afterwards the VND starts from the beginning with the first neighborhood $\mathcal{N}^1$.

We ensure that a neighborhood exploration is however only started if an improvement is possible. Hence, if neighborhoods are included into one another, i.e., $\mathcal{N}^i \subset \mathcal{N}^j$ for $i \neq j$, we make sure that the smaller neighborhood $\mathcal{N}^i$ is prioritized before the larger neighborhood $\mathcal{N}^j$, i.e., $i < j$. We exploit $\mathcal{N}^{CO} = \mathcal{N}_1^{BS} \subset \mathcal{N}_2^{BS} \subset \mathcal{N}_3^{BS}$. Finally, we allow that two neighborhoods receive identical random priorities, in which case we alternatingly explore them.

Perturbation When the VND terminates, a local minimum w.r.t. all neighborhoods has been found. To escape from such a local minimum and to guide the search towards a region of the solution space not yet explored, ILS applies a perturbation step. The design of this perturbation step is delicate: If the perturbation step is too strong, ILS behaves like a random multi-start algorithm with a relatively high computational burden for the VND. In this case, an average iteration (of the main loop of Algorithm 2) takes more time. On the other hand, if the perturbation is too weak, the VND tends to fall back into the known local optimum just found in the iteration before. The diversification of the search is rather limited then.

For the sake of simplicity, we use the random double-bridge move for the perturbation of the current G-tour x. It is known from the TSP (Johnson and McGeoch, 1997) that the double-bridge move gives an effective perturbation. In comparison to the double-bridge move used within the local search (Section 2.3.1), the edges to be removed are chosen randomly. Moreover, if the VND falls back into same the local optimum three times in a row (as a necessary condition we test whether the objective function values are identical), we do not perturb with a random double-bridge move. Instead the current solution is set to a fresh starting solution, computed with the above-described randomized GTSP construction heuristic.

Acceptance Criterion The acceptance criterion controls the balance between intensification and diversification of the search. In case no acceptance criterion was used, the ILS performed a randomized walk in the space of all local optima, i.e., a strong diversification was achieved. On the opposite, if only better solutions were accepted, intensification was very strong.

We balance intensification and diversification with the deterministic record-to-record travel (Dueck, 1993) acceptance criterion as follows: Every solution x^* improving the current solution x is always accepted. Moreover, non-improving solutions x^* (those with $c(x^*) > c(x)$) are accepted if the deviation from the cost of the best observed solution (= record, x_{ILS}) so far is smaller than a predefined

threshold. To this end, we test $c(x^*) \leq (1+\epsilon) \cdot c(x_{ILS})$ for a (small) value $\epsilon > 0$ (cf. Step 6 in Algorithm 2). Initially, we set ϵ to 0.03 so that a deviation of 3% from the record is allowed. With a straightforward geometric cooling schedule, commonly used in simulated annealing (Kirkpatrick *et al.*, 1987), we systematically lower ϵ in the course of the ILS. The cooling update takes place every N iterations of the main loop. The update lowers ϵ by the factor $h = 0.8$ (Step 10). That means, e.g., that ϵ is at approximately a tenth of its initial value after $10N$ iterations.

2.4.2 GTSP Instances

We evaluate the performance of the basic ILS on commonly used symmetric GTSP problem libraries that have also been used to compare

- the memetic algorithm *GK* of Gutin and Karapetyan (2010),
- the Lin-Kernighan-Helsgaun *GLKH* algorithm of Helsgaun (2015), and
- the large neighborhood search *GLNS* of Smith and Imeson (2017).

Note that *GK*, *GLKH*, and *GLNS* are the best-performing state-of-the-art GTSP solvers published in the literature (see Section 2.2). Smith and Imeson (2017) made the three GTSP algorithms well comparable by running all of them for the same amount of computational time (we provide details below). They were tested on four libraries:

- The `GTSP_LIB` was introduced by Fischetti *et al.* (1997) and extended by Silberholz and Golden (2007). The vertex clustering simulates geographical regions. The 45 largest instances of the library have been used in the `GK`, `GLKH`, and `GLNS` comparison presented by Smith and Imeson (2017). We omit the five asymmetric instances.
- The `BAF_LIB` was introduced by Bontoux (2008). The pseudo-random clustering scheme implies that there are *no* geographical regions. The standard comparison uses the 45 largest instances.
- The `MOM_LIB` was introduced by Mestria *et al.* (2013) and contains instances with six different clustering schemes. Here, the 45 largest instances are used in the *GK*, *GLKH*, and *GLNS* comparison of Smith and Imeson (2017).
- The `LARGE_LIB` was introduced by Helsgaun (2015) and contains very large instances. The clusters were also generated with the clustering scheme of Fischetti *et al.* (1997),i.e., they simulate geographical regions. The 26 smallest instances (referred to as `LARGE_LIB(I)` in the following) have been used to benchmark the *GK*, *GLKH*, and *GLNS* algorithms. Seven even larger instances (referred to as `LARGE_LIB(II)`) have been used to benchmark the *GLKH* against the *GLNS* algorithm, while no results for GK have been reported.

2.4.3 Computational Results of the Basic ILS

The basic ILS was coded in C++ and compiled with MS Visual Studio 2015 in release mode. All computations were performed on a standard PC with MS Windows 10 running on an Intel® Core™ i7-5930K CPU clocked at 3.5 GHz and with 64 GB RAM.

We designed the experiments in the same way as Smith and Imeson (2017) did: Smith and Imeson set the computation time limit to

- 300 seconds for all instances of the `GTSP_LIB`, `MOM_LIB`, and `BAF_LIB`, and
- 1200 seconds for all instances of the `LARGE_LIB`.

According to PassMark (`https://www.cpubenchmark.net/singleCompare.php`), our processor is slightly slower in the single-thread performance than the Intel® Core™ i7-6700 clocked at 3.40 GHz used by Smith and Imeson (2062 points versus 2303 points). We therefore grant 335 seconds for instances of `GTSP_LIB`, `MOM_LIB`, and `BAF_LIB` and 1341 seconds for instances of `LARGE_LIB`.

Table 2.3 shows the computational results, where columns have the following meaning:

- `instance` is the name of the GTSP instance. The name's prefix is the number N of clusters and the suffix is number n of vertices.
- `best known` shows the cost of the best-known solution x_{BKS} (BKS), i.e., $c(x_{BKS})$, known from the literature for the instance. This information is taken from Smith and Imeson (2017) (available at `https://ece.uwaterloo.ca/~sl2smith/GLNS/`) and Helsgaun (2015) (available at `http://akira.ruc.dk/~keld/research/GLKH/`).
- `#best` is the number of runs, out of ten, in which a BKS was obtained.
- $\Delta(\%)$ shows the average percentage error between the cost $c(x_{BKS})$ of a BKS and the cost $c(x_{ILS})$ obtained by our ILS over 10 runs. The percentage error e is calculated as $e = 100 \cdot (c(x_{ILS}) - c(x_{BKS}))/c(x_{BKS})$.

GTSP_LIB				MOM_LIB				BAF_LIB				LARGE_LIB			
instance	best	#best	Δ(%)	instance	best	#best	Δ(%)	instance	best	#best	Δ(%)	instance	best	#best	Δ(%)
31pr152	51,576	10	—	50i2000-603	4,325	10	—	baf20kroD100	5,266	10	—	10C1k.0	2,522,585	10	—
32u159	22,664	10	—	50i2500-707	3,961	10	—	baf20kroE100	5,449	10	—	31C3k.0	3,553,142	10	—
35si175	5,564	10	—	50i3000-802	4,070	10	—	baf20rat99	230	10	—	49usa1097	10,337	10	—
36brg180	4,420	10	—	50kroA100	15,944	10	—	baf20rd100	1,747	10	—	200C1k.0	6,375,154	10	—
39rat195	854	10	—	50kroB100	15,842	10	—	baf21eil101	105	10	—	200E1k.0	9,662,857	0	0.34
40d198	10,557	10	—	50lin105	11,294	10	—	baf21lin105	2,758	10	—	235pcb1173	23,399	1	0.78
40kroa200	13,406	10	—	50lin318	18,163	10	—	baf22pr107	6,849	10	—	259d1291	28,400	4	0.11
40krob200	13,111	10	—	50nrw1379	7,449	10	—	baf24gr120	1,377	10	—	261rl1304	150,468	0	0.22
41gr202	23,301	10	—	50pcb1173	9,385	10	—	baf25pr124	10,745	10	—	265rl1323	154,023	0	0.35
45ts225	68,340	10	—	50pcb442	14,430	10	—	baf26bier127	11,740	10	—	276nrw1379	20,050	0	0.32
45tsp225	1,612	10	—	50pr1002	54,583	10	—	baf28pr136	17,824	10	—	280fl1400	15,316	10	—
46gr229	71,972	10	—	50pr439	45,253	10	—	baf29pr144	14,070	10	—	287u1432	54,469	0	0.28
46pr226	64,007	10	—	50rat783	1,626	10	—	baf30kroA150	7,005	10	—	316fl1577	14,182	10	—
53gil262	1,013	10	—	50rat99	814	10	—	baf30kroB150	5,855	10	—	331d1655	29,443	0	0.49
53pr264	29,549	10	—	50vm1084	54,156	10	—	baf31pr152	13,002	10	—	350vm1748	185,459	0	0.38
56a280	1,079	10	—	72vm1084-8x9	64,647	10	—	baf32u159	7,301	10	—	364u1817	25,530	0	0.46
60pr299	22,615	10	—	75lin105	13,134	10	—	baf39rat195	477	10	—	378rl1889	184,034	0	0.36
64lin318	20,765	10	—	81vm1084-9x9	69,659	10	—	baf40d198	1,466	10	—	421d2103	40,049	0	1.02
80rd400	6,361	10	—	100i1000-410	5,481	10	—	baf40kroA200	7,113	10	—	431u2152	27,614	0	1.04
84u417	9,651	10	—	100i1500-506	5,088	8	0.06	baf40kroB200	7,126	10	—	464u2319	65,758	0	2.57
87gr431	101,946	10	—	100i2000-604	5,316	7	0.08	baf41gr202	3,531	10	—	479pr2392	169,874	0	2.42
88pr439	60,099	10	—	100i2500-708	5,297	10	—	baf45ts225	25,697	10	—	608pcb3038	52,416	0	5.07
89pcb442	21,657	10	—	100i3000-803	5,458	3	0.04	baf46pr226	13,555	10	—	633C3k.0	10,255,031	0	2.28
99d493	20,023	10	—	100nrw1379	10,566	10	—	baf53gil262	571	4	0.32	633E3k.0	16,197,552	0	5.37
107ali535	128,639	10	—	100pcb1173	13,901	4	0.20	baf53pr264	7,716	10	—	759fl3795	18,662	0	1.05
107att532	13,464	10	—	100pr1002	74,269	9	0.00	baf60pr299	10,047	10	—	893fnl4461	63,163	0	6.72
107si535	13,502	10	—	100prb1173-10x10	12,644	10	—	baf64lin318	7,489	10	—	**Average for LARGE_LIB(I):**		**2.50**	**1.22**
113pa561	1,038	10	—	100rat783-10x10	2,216	10	—	baf80rd400	3,254	3	1.07				
115rat575	2,388	10	—	100rat783	2,496	10	—	baf84fl417	2,226	10	—				
115u574	16,689	10	—	100vm1084	78,440	10	—	baf87gr431	10,569	10	—				
131p654	27,428	10	—	144pcb1173-12x12	16,412	1	0.22	baf88pr439	13,882	10	—	1183rl5915	309,243	0	5.71
132d657	22,498	8	0.02	144rat783-12x12	2,813	5	0.02	baf89pcb442	8,749	10	—	1187rl5934	295,767	0	5.52
134gr666	163,028	7	0.17	150i1000-411	6,296	9	0.05	baf99d493	3,081	10	—	1480pla7397	12,732,870	0	4.41
145u724	17,272	10	—	150i1500-507	6,085	10	—	baf107att532	3,880	8	0.06	100C10k.0	6,158,999	0	1.17
157rat783	3,262	2	0.10	150i2000-605	5,940	9	0.00	baf107si535	8,912	8	0.32	2000C10k.0	18,044,846	0	4.86
200dsj1000	9,187,884	4	0.10	150i2500-709	6,158	6	0.10	baf113pa561	431	0	1.39	2000E10k.0	28,769,011	0	8.41
201pr1002	114,311	5	0.03	150i3000-804	6,551	0	0.41	baf115rat575	1,330	9	0.15	2370rl11849	427,996	0	8.36
207si1032	22,306	0	0.07	150nrw1379	13,370	3	0.15	baf131p654	5,824	4	0.03	**Average for LARGE_LIB(II):**		**0.00**	**5.49**
212u1060	106,007	1	0.23	150pcb1173	17,082	3	0.35	baf132d657	8,132	10	—				
217vm1084	130,704	8	0.07	150pr1002	92,969	7	0.03	baf145u724	7,354	1	0.56				
				150rat783	3,131	2	0.31	baf157rat783	1,700	9	0.25				
				150vm1084	95,922	10	—	baf201pr1002	48,400	0	2.40				
				200i2000-606	7,272	1	0.28	baf207si1032	18,836	9	0.00				
				200i2500-710	7,191	4	0.15	baf212u1060	38,639	5	0.16				
				200i3000-805	6,902	2	0.32	baf217vm1084	44,681	10	—				
Average		**8.88**	**0.02**			**7.84**	**0.06**			**8.67**	**0.15**			**1.97**	**2.12**

Table 2.3: Results of the basic ILS on 163 symmetric GTSP instances.

We give some side notes: The basic ILS finds a BKS at least once for 134 of the 163 instances, while for 98 instances it was determined in 10 of 10 runs.

For the GTSP_LIB, a BKS is found in 8.88 of 10 runs on average. For instances with up to 131 clusters, a BKS is found or undercut in 10 of 10 runs, while for instances with more than 131 clusters, the number of runs in which a BKS is obtained varies between 0 and 10 with an average of 5.

For the MOM_LIB, the basic ILS finds a BKS in at least one run for 44 of the 45 instances. For 27 instances, all ten runs find it. For the instances for which a BKS was not obtained in all ten runs, the average number of achieved BKS is 4.6, with an average percentage error of 0.15%.

For the BAF_LIB, the basic ILS finds a BKS in all ten runs for 33 instances. For the other instances, the average number of BKSs found is 5 with an average percentage error of 0.56%. On the downside, for two instances, a BKS was never obtained in all 10 runs. For both of them, the average percentage error is rather high with 1.39% and 2.40%.

For the LARGE_LIB, we summarize the results as follows: From 26 of the subset LARGE_LIB(I) instances, only 6 instances can be solved with the BKS in 10 of 10 runs, two instances can be solved at least one time, whereas for 19 instances a BKS is never obtained. Thus, the average percentage error is relatively large with 1.22%. In particular for instances with more than 400 clusters, the average percentage error varies between 1.05% and 6.72%. On the second subset LARGE_LIB, no BKS has been obtained. Also here, the percentage error of 5.49% is relatively large.

Finally, we compare the basic ILS with the three algorithms *GK*, *GLKH*, and *GLNS*. Table 2.4 presents average values of *#best* and $\Delta(\%)$ on the five groups of instances and for the four algorithms. Note that only the symmetric instances are taken into account to be comparable with the analysis of Smith and Imeson (2017). Moreover, the *GK* algorithm could not be compared on the LARGE_LIB(II) instances, because these instances were not considered in (Gutin and Karapetyan, 2010). In spite of its simplicity, the basic ILS produces reasonable results on the first three libraries, while results for the LARGE_LIB are clearly behind.

	GTSP_LIB		MOM_LIB		BAF_LIB		LARGE_LIB(I)		LARGE_LIB(II)	
Algorithm	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)
GK	9.10	**0.01**	8.44	0.03	8.11	0.29	2.77	0.76	—	—
GLKH	**9.20**	**0.01**	5.40	0.82	5.04	6.51	3.04	0.52	0.00	**2.56**
GLNS	8.73	**0.01**	**9.18**	**0.02**	**8.91**	**0.07**	**3.31**	**0.50**	0.00	6.46
Basic ILS	8.88	0.02	7.84	0.06	8.67	0.15	2.50	1.22	0.00	5.49

Table 2.4: Comparison of the basic ILS with algorithms from the literature.

We can provide some reasons why the basic ILS is not convincing on the `LARGE_LIB`: In particular for instances with a large number N of clusters, the ILS performs only a relative small number of iterations, because some neighborhoods are very time-consuming. It is clear that, in these cases, the more time-consuming neighborhoods should either be completely omitted or should be explored less often, e.g., only when elite solutions are found.

2.5 Refined Iterated Local Search

Section 2.4 has shown that the basic ILS already achieves reasonable results. It is however not yet competitive with the currently best algorithm *GLNS* and does not consistently outperform *GK* and *GLKH*. Our overall goal for the ILS still remains to design a straightforward but powerful algorithm and well-reproducible results. In Section 2.5.1, we describe the implementation of refinements regarding a *reset component*, *prioritization of the neighborhoods in VND*, the use of the *Balas-Simonetti neighborhood for detecting high-quality solutions*, and a modified *acceptance criterion*. Section 2.5.2 introduces an instance-based selection of an ILS setup, while Section 2.5.3 discusses the final results.

2.5.1 Refinements

Reset We observed that, in particular for some more difficult instances, the best found solution x_{ILS} is identified only once in a run (if ever). It seems that intensifying the search in the solution space around x_{ILS} could help to identify other very good solutions, hopefully better ones.

Hence, if no improvement takes place for a pre-defined number of iterations, the *reset component* resets the current solution x to the best found solution x_{ILS}. Pretests have shown a reset after every 50 iterations is a good compromise balancing intensification and diversification.

VND with Neighborhood Prioritization The classical VND, as proposed by Hansen and Mladenović (2005), orders the neighborhoods according to their size and expected computational effort. The first neighborhood is then the one with the smallest computational effort. The second neighborhood is only explored when a local optimum in the first is reached. Moreover, after an improving move in the second neighborhood, one returns to the exploration of the first neighborhood. More than two neighborhoods are "prioritized" in the same fashion.

We want to find out whether a prioritization is also beneficial for the local-search component of our ILS. This includes the possibility to completely disregard

neighborhoods and to choose a pivoting strategy (first improvement or best improvement) per neighborhood. Moreover, for the BS neighborhood, a value for the parameter k has to be set. In contrast, the maximum string length in the SR neighborhood is fixed to $L = 4$, because larger values lead to unacceptably long computation times.

To this end, we consider the set of all parameterized neighborhoods

$$\mathcal{N}^{all} = \{\mathcal{N}^{2Opt}_{best}, \mathcal{N}^{2Opt}_{first}, \mathcal{N}^{3Opt}_{best}, \mathcal{N}^{3Opt}_{first}, \mathcal{N}^{dbl-brdg}, \ldots, \mathcal{N}^{BS}_2, \mathcal{N}^{BS}_3, \mathcal{N}^{BS}_4, \mathcal{N}^{BS}_5, \ldots, \mathcal{N}^{SR}_4, \ldots, \mathcal{N}^{Gutin}\}$$

resulting from the description given in Section 2.3.

Of course, we do not perform a full factorial parameter study over $\mathcal{N}^{all}$, simply because the number of possible VND variants and resulting ILS setups is too large. Furthermore, there is the danger of overfitting the possible setups to the set of 163 GTSP instances.

Our pragmatic approach to design refined ILS setups with different prioritized neighborhoods in the VND can be summarized as follows:

- Initially, we randomly select one of the four GTSP libraries and a subset $\mathcal{I}$ of $10 = |\mathcal{I}|$ instances from it.
- For this subset $\mathcal{I}$, we determine the subset $\mathcal{N}^{VND} \subset \mathcal{N}^{all}$ of useful neighborhoods by adding, one by one, a not yet selected parameterized neighborhood $\mathcal{N} \in \mathcal{N}^{all} \setminus \mathcal{N}^{VND}$ to $\mathcal{N}^{VND}$. Each addition of a neighborhood defines one iteration. Initialization and iterations are performing the following steps:
  - The neighborhood selection process is initialized with $\mathcal{N}^{VND}_0 = \varnothing$.
  - In the pth iteration ($p = 1, 2, \ldots$), i.e., when $|\mathcal{N}^{VND}_{p-1}| = p - 1$, the next parameterized neighborhood $\mathcal{N}^p \in \mathcal{N}^{all} \setminus \mathcal{N}^{VND}_{p-1}$ is the one producing the best improvement over the ILS using $\mathcal{N}^{VND}_{p-1}$, when added to $\mathcal{N}^{VND}_{p-1}$. For each candidate $\mathcal{N}^p$, we consider the tentative subset $\mathcal{N}^{VND}_p = \{\mathcal{N}^p\} \cup \mathcal{N}^{VND}_{p-1}$ and evaluate the ILS with the instances $\mathcal{I}$.
  - We measure the improvement (relative to the subset $\mathcal{I}$) as the value

    $$\Delta(\mathcal{I}, \mathcal{N}^{VND}_p) = \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \left(c(x^I_{ILS}) - c(x^I_{BKS})\right) / c(x^I_{BKS}),$$

    where $c(x^I_{ILS})$ is the cost of a best solution of instance I obtained with the refined ILS using $\mathcal{N}^{VND}_p$, and $c(x^I_{BKS})$ the cost of a BKS for instance I known from the literature.
  - We perform a single run per instance $I \in \mathcal{I}$ limited to 120 seconds of computation time.
  - If there is no improvement, i.e., $\Delta(I, \mathcal{N}^{VND}_p) \geq \Delta(I, \mathcal{N}^{VND}_{p-1})$ for all $\mathcal{N}^p \in \mathcal{N}^{all} \setminus \mathcal{N}^{VND}_{p-1}$, the process of adding neighborhoods stops.
    The final set of neighborhoods is then chosen as $\mathcal{N}^{VND} = \mathcal{N}^{VND}_{p-1}$.

The sequence in which the neighborhoods are added to the subset $\mathcal{N}^{VND}$ defines the priorities. Note that the VND is deterministic and that the selection of the instances $I \in \mathcal{I}$ is the random component of the approach to design refined ILS setups.

By repeating the random selection of instances $\mathcal{I}$, we have obtained several versions of a nested VND. Three rather well-performing ones give rise to refined versions of ILS in which the random VND is replaced by nested VND I, VND II, or VND III summarized in Table 2.5. For example, VND I uses five neighborhoods with prioritization $\mathcal{N}^1$ = Relocation+, $\mathcal{N}^2$ = BS($k = 4$), $\mathcal{N}^3$ = Double Bridge, $\mathcal{N}^4$ = 3-Opt, and $\mathcal{N}^5$ = Gutin. For all five neighborhoods, the versions with best-improvement pivoting rule are selected. None of the three VND setups uses the neighborhoods Swap+, CO, or BS($k = 3$).

	VND I		VND II		VND III	
Neighborhood	priority	pivoting	priority	pivoting	priority	pivoting
2-Opt	—	—	6	first	6	first
3-Opt	4	best	2	best	1	best
Double Bridge	3	best	4	best	4	best
Relocation+	1	best	—	—	—	—
Swap+	—	—	—	—	—	—
CO	—	—	—	—	—	—
BS $k = 3$	—	—	—	—	—	—
BS $k = 4$	2	best	—	—	—	—
BS $k = 5$	—	—	3	best	3	best
String Relocation+	—	—	5	best	5	best
Gutin Neighborhood	5	best	1	best	2	best

Table 2.5: Selection, Priorities, and Pivoting Strategies of Neighborhoods.

Note: The maximum string length for String Relocation+ is $L = 4$ for VND II and VND III. The neighborhoods Swap+, CO, and BS for $k = 3$ are not used in VND I, VND II, and VND III, opposed to the basic ILS described in the previous section.

We interpret the outcome of the parameter tuning for the three VNDs with neighborhood prioritization as follows:

- The neighborhoods Swap+ and CO are redundant. This is not surprising given that CO in contained in BS and Swap+ is contained in the Gutin neighborhood.
- All other neighborhoods are beneficial, at least in some combination.
- A high priority is either assigned to Relocation+ (in VND I) or to Gutin (in VND II and III). These neighborhoods have in common that they are able to simultaneously relocate a vertex and change the cluster representative.

- BS for $k = 3$ is not beneficial when supersets of the neighborhood (for $k \geq 4$) are available alternatives.

Balas-Simonetti for High-Quality Solutions When a high-quality solution is found, we try to further improve it with the Balas-Simonetti neighborhood with a high k-value (Section 2.3.3). We define high-quality solutions as G-tours with a cost falling into the lower 1%-fractile of all local optimal solutions x^* found so far (see Step 5 of Algorithm 2). Therefore, the ILS regularly computes and updates a bound b on the cost. More precisely, we set the first bound b after 200 iterations of the main loop and subsequently update b after every 50 iterations, but only if the newly computed bound is lower than the old bound. With this latter condition, the bound b never increases.

If a solution x^* is a high-quality solution, the BS neighborhood is explored. We have experimented with different values of k. Obviously, larger values of k offer a higher potential for an improvement but also come at a large computational effort. As explained for CO, we rotated the G-tour x so that the first cluster has minimum cardinality. Additionally, we run the corresponding shortest-path problem only for the vertex x_1 that is currently selected in $x = (x_1, x_2, \ldots, x_N, x_1)$. With these accelerations, pre-tests have shown that $k = 8$ is still possible and offers a good trade-off between solution quality and computational time.

Technically, the Balas-Simonetti neighborhood with $k = 8$ is added to the VND with the lowest possible priority, i.e., highest value *priority* (cf. Table 2.5). Every time the VND calls this new BS neighborhood, it checks if the cost of the current solution x belongs to the best 1% fractile, i.e., $c(x) \leq b$.

Modified Acceptance Criterion The rather poor performance on the `LARGE_LIB` partly results from the fact that only very few ILS iterations can be performed within the given overall time budget. As a consequence, the cooling scheme used in the record-to record acceptance criterion is not appropriate for these instances.

Therefore, we replace the naive cooling schedule (Steps 10 and 11 in Algorithm 2, i.e., every N iterations the value of ϵ is multiplied by $h = 0.8$) by a scheme that first tries to predict the overall number of ILS iterations that can be performed within the given time budget t^{max} (the equivalent of 300 or 1200 seconds, see Section 2.4.3). To this end, we always run the ILS for 50 iterations with the initial value of ϵ. Let time t_{50} denote the computation time consumed up to this point. Then,

$$iter^{remain} = 50 \cdot (t^{max} - t_{50})/t_{50}$$

is an estimate for the number of ILS iterations that can be performed in the remaining computation time. As explained in Section 2.4, we would like to perform at least 10 cooling down steps so that with the cooling rate $h = 0.8$ the initial

temperature ϵ is divided by 10 (note that $10 \approx 1/0.8^{10}$). In order to make all results reproducible, i.e., less dependent of small deviations in computation time, we round down $iter^{remain}/10$ to the next smaller power of 2. Let N^{cool} this number. The only modification to the cooling scheme is that a cooling down step (triggered in Step 10 of Algorithm 2) performs an update of ϵ after every N^{cool} ILS iterations (instead of after every N iterations). For the large-scale instances, it means that more cooling down steps are performed after fewer ILS iterations. Moreover, the initial value of ϵ is set to 0.01 (it was defined as 0.03 in the basic ILS), so that a slightly more restrictive regime for accepting deteriorating solution is established.

	GTSP_LIB		MOM_LIB		BAF_LIB		LARGE_LIB(I)		LARGE_LIB(II)	
ILS with	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)
random VND	8.88	0.02	**7.84**	**0.06**	**8.67**	**0.15**	2.50	1.22	0.00	5.49
VND I	**9.20**	**0.01**	7.42	0.11	8.22	0.43	2.81	0.96	0.00	6.44
VND II	8.90	0.02	7.44	0.10	7.44	0.95	**2.92**	**0.71**	0.00	5.11
VND III	7.73	0.09	4.71	0.42	7.24	0.38	2.23	0.77	0.00	**3.47**

Table 2.6: Performance of the four ILS setups on four GTSP Libraries.

Table 2.6 summarizes the comparison of the basic ILS and the three new refined ILS versions using VND I, VND II, and VND III. Note that the first line repeats the results presented before for the basic ILS, which uses the random VND and no reset to x_{ILS}. The refined ILS with VND I performs very well on the GTSP_LIB both regarding the average number of BKSs found (*#best*) and the average deviation to the BKS (Δ(%)).

The refined ILS with VND II outperforms the other VND versions on the LARGE_LIB(I). Likewise, the refined ILS with VND III performs best on LARGE_LIB(II). For MOM_LIB and BAF_LIB, the basic ILS is still the best algorithm.

We interpret the outcome of the experiments as follows:

- The ILS with VND I is the clear winning setup for the GTSP_LIB.
- No refined ILS setups are successful for the MOM_LIB and BAF_LIB.
- ILS with VND II and VND III lead to substantial improvements for the LARGE_LIB.
- VND II and III are very similar (see Table 2.5). The different performance on the two subsets (I) and (II) of the LARGE_LIB are surprising. It seems that the very largest instances (subset II) benefit more from the fast 3-Opt neighborhood than the more time-consuming Gutin neighborhood. Indeed, when G-tours are very long the routing aspect becomes increasingly important.
- The comparison of the four ILS setups on the entire benchmark shows that an instance-based selection of a setup has a high potential to improve the overall performance. This idea is elaborated and detailed in the next section.

2.5.2 Instance-Based Selection of an ILS Setup

With four alternative ILS setups at hand (basic ILS, refined ILS with VND I, VND II or VND III), the question is now whether one can estimate the performance beforehand. Looking into instance-by-instance results for the three refined ILS setups, the refined ILS with VND I outperforms the other version on instances that are *not* geometrically clustered. Accordingly, we define for a GTSP instance, the *average relative inner-cluster distance* as

$$\gamma(I) = \frac{\bar{c}_{in}}{\bar{c}} \qquad \text{with}$$

$$\bar{c}_{in} = \left(\sum_{i \in I} \sum_{x < x' \in V_i} c_{x,x'} \right) / \left(\sum_{i \in I} \binom{|V_i|}{2} \right) \quad \text{and} \quad \bar{c} = \left(\sum_{x < x' \in V} c_{x,x'} \right) / \binom{|V|}{2},$$

where $\bar{c}_{in}$ is the average distance between vertices of the same cluster and $\bar{c}$ the average distance between two arbitrary vertices. Moreover, the refined ILS with VND II works well for large instances, where we define large instance as one with $250 < N < 500$. Extra large instances, i.e., instances with $N \geq 500$, can be solved best with the refined ILS with VND III.

Having defined this, the following simple rules assign an instance I to one of the three refined ILS versions:

- If $\gamma(I) < 0.5$, use the refined ILS with VND I;
- If $\gamma(I) \geq 0.5$ and $250 < N < 500$, use the refined ILS with VND II;
- If $\gamma(I) \geq 0.5$ and $N \geq 500$, use the refined ILS with VND III;
- In all other cases, use the basic ILS (with random VND, without reset).

The resulting metaheuristic with the instance-based selection of the ILS setup is denoted as *the refined ILS* from now on.

2.5.3 Computational Results of the Refined ILS

The computational setup (time budget etc.) is the same as for the computational experiments presented in Section 2.4.3. Table 2.7 shows the results obtained with the refined ILS on the 163 symmetric GTSP instances. The meaning of the columns is the same as in Table 2.3.

GTSP_LIB				MOM_LIB				BAF_LIB				LARGE_LIB			
instance	best	#best	Δ(%)	instance	best	#best	Δ(%)	instance	best	#best	Δ(%)	instance	best	#best	Δ(%)
31pr152	51,576	10	—	50i2000-603	4,325	10	—	baf20kroD100	5,266	10	—	10C1k.0	2,522,585	10	—
32u159	22,664	10	—	50i2500-707	3,961	10	—	baf20kroE100	5,449	10	—	31C3k.0	3,553,142	10	—
35si175	5,564	10	—	50i3000-802	4,070	10	—	baf20rat99	230	10	—	49usa1097	10,337	10	—
36brg180	4,420	10	—	50kroA100	15,944	10	—	baf20rd100	1,747	10	—	200C1k.0	6,375,154	10	—
39rat195	854	10	—	50kroB100	15,842	10	—	baf21eil101	105	10	—	200E1k.0	9,662,857	2	0.18
40d198	10,557	10	—	50lin105	11,294	10	—	baf21lin105	2,758	10	—	235pcb1173	23,399	0	0.74
40kroa200	13,406	10	—	50lin318	18,163	10	—	baf22pr107	6,849	10	—	259d1291	28,400	8	0.04
40krob200	13,111	10	—	50nrw1379	7,449	10	—	baf24gr120	1,377	10	—	261rl1304	150,468	6	0.06
41gr202	23,301	10	—	50pcb1173	9,385	10	—	baf25pr124	10,745	10	—	265rl1323	154,023	0	0.12
45ts225	68,340	10	—	50pcb442	14,430	10	—	baf26bier127	11,740	10	—	276nrw1379	20,050	2	0.46
45tsp225	1,612	10	—	50pr1002	54,583	10	—	baf28pr136	17,824	10	—	280fl1400	15,316	10	—
46gr229	71,972	10	—	50pr439	45,253	10	—	baf29pr144	14,070	10	—	287u1432	54,469	0	0.31
46pr226	64,007	10	—	50rat783	1,626	10	—	baf30kroA150	7,005	10	—	316fl1577	14,182	10	—
53gil262	1,013	10	—	50rat99	814	10	—	baf30kroB150	5,855	10	—	331d1655	29,443	0	0.33
53pr264	29,549	10	—	50vm1084	54,156	10	—	baf31pr152	13,002	10	—	350vm1748	185,459	0	0.20
56a280	1,079	10	—	72vm1084-8x9	64,647	10	—	baf32u159	7,301	10	—	364u1817	25,530	0	0.32
60pr299	22,615	10	—	75lin105	13,134	10	—	baf39rat195	477	10	—	378rl1889	184,034	0	0.26
64lin318	20,765	10	—	81vm1084-9x9	69,659	10	—	baf40d198	1,466	10	—	421d2103	40,049	0	0.62
80rd400	6,361	10	—	100i1000-410	5,481	10	—	baf40kroA200	7,113	10	—	431u2152	27,614	0	0.74
84u417	9,651	10	—	100i1500-506	5,088	8	0.06	baf40kroB200	7,126	10	—	464u2319	65,758	0	1.72
87gr431	101,946	10	—	100i2000-604	5,316	7	0.08	baf41gr202	3,531	10	—	479pr2392	169,874	0	1.20
88pr439	60,099	10	—	100i2500-708	5,297	10	—	baf45ts225	25,697	10	—	608pcb3038	52,416	0	1.72
89pcb442	21,657	10	—	100i3000-803	5,458	3	0.04	baf46pr226	13,555	10	—	633C3k.0	10,255,031	0	0.92
99d493	20,023	10	—	100nrw1379	10,566	10	—	baf53gil262	571	4	0.32	633E3k.0	16,197,552	0	1.71
107ali535	128,639	10	—	100pcb1173	13,901	4	0.20	baf53pr264	7,716	10	—	759fl3795	18,662	0	0.35
107att532	13,464	10	—	100pr1002	74,269	9	0.00	baf60pr299	10,047	10	—	893fnl4461	63,163	0	3.53
107si535	13,502	10	—	100prb1173-10x10	12,644	10	—	baf64lin318	7,489	10	—	**Average for LARGE_LIB(I):**		**3.00**	**0.60**
113pa561	1,038	10	—	100rat783-10x10	2,216	10	—	baf80rd400	3,254	3	1.07				
115rat575	2,388	10	—	100rat783	2,496	10	—	baf84fl417	2,226	10	—				
115u574	16,689	8	0.04	100vm1084	78,440	10	—	baf87gr431	10,569	10	—				
131p654	27,428	10	—	144pcb1173-12x12	16,412	1	0.22	baf88pr439	13,882	10	—	1183rl5915	309,243	0	2.46
132d657	22,498	10	—	144rat783-12x12	2,813	5	0.02	baf89pcb442	8,749	10	—	1187rl5934	295,767	0	2.39
134gr666	163,028	9	0.05	150i1000-411	6,296	9	0.05	baf99d493	3,081	10	—	1480pla7397	12,732,870	0	1.95
145u724	17,272	9	0.02	150i1500-507	6,085	10	—	baf107att532	3,880	8	0.06	100C10k.0	6,158,999	0	1.17
157rat783	3,262	4	0.08	150i2000-605	5,940	9	0.00	baf107si535	8,912	8	0.32	2000C10k.0	18,044,846	0	3.44
200dsj1000	9,187,884	8	0.02	150i2500-709	6,158	6	0.10	baf113pa561	431	0	1.39	2000E10k.0	28,769,011	0	5.74
201pr1002	114,311	8	0.01	150i3000-804	6,551	0	0.41	baf115rat575	1,330	9	0.15	2370rl11849	427,996	0	5.67
207si1032	22,306	0	0.06	150nrw1379	13,370	3	0.15	baf131p654	5,824	4	0.03	**Average for LARGE_LIB(II):**		**0.00**	**3.26**
212u1060	106,007	3	0.20	150pcb1173	17,082	3	0.35	baf132d657	8,132	10	—				
217vm1084	130,704	9	0.03	150pr1002	92,969	7	0.03	baf145u724	7,354	1	0.56				
				150rat783	3,131	2	0.31	baf157rat783	1,700	9	0.25				
				150vm1084	95,922	10	—	baf201pr1002	48,400	0	2.40				
				200i2000-606	7,272	1	0.28	baf207si1032	18,836	9	0.00				
				200i2500-710	7,191	4	0.15	baf212u1060	38,639	5	0.16				
				200i3000-805	6,902	2	0.32	baf217vm1084	44,681	10	—				
Average		**9.20**	**0.01**			**7.84**	**0.06**			**8.67**	**0.15**			**2.36**	**1.16**

Table 2.7: Results of the refined ILS on the 163 symmetric GTSP instances.

Again, we give some side notes. The refined ILS finds a BKS at least once for 136 of the 163 instances (83.4%). While this number increases by two instances in comparison to the basic ILS, the number of instances for which a BKS was not found decreases from 29 to 27.

For the GTSP_LIB, a BKS is found in 9.20 of 10 runs on average.

For instances with up to 113 clusters, a BKS is found or undercut in 10 of 10 runs, while for instances with more than 113 clusters, the number of runs in which a BKS is obtained varies between 0 and 10 with an average of 7.33. Their corresponding average deviation could be reduced from 0.06% to 0.04% compared to the basic ILS.

For the LARGE_LIB(I), 6 of 26 instances are solved with a BKS in 10 of 10 runs, 4 instances are solved with a BKS at least one time, and for 16 instances the BKS is never obtained (2 less compared to the basic ILS). For instances with more than 400 clusters, the average percentage error is relatively large and varies between 0.62% and 3.53%, with an average of 1.39% (3.06% for basic ILS). During pre-tests, a new BKS is found for instance 287u1432 (See Appendix for more information). For the second part of LARGE_LIB, LARGE_LIB(II), still no BKS is found, but the average percentage error was decreased from 5.49% to 3.26%.

The results for MOM_LIB and BAF_LIB remain unchanged because all these instances are still solved with the basic ILS.

Effectiveness of the Balas-Simonetti Neighborhood for High-Quality Solutions We analyze now the impact that the Balas-Simonetti Neighborhood $\mathcal{N}_8^{BS}$ has on the overall effectiveness of the refined ILS. Note that the instance-based selection of an ILS setup (see Section 2.5.2) chooses the refined version only for the libraries GTSP_LIB and LARGE_LIB. For these instances, Table 2.8 shows the number of neighborhood explorations (*#calls*) of $\mathcal{N}_8^{BS}$, the percentage of neighborhood explorations that found an improving neighbor (*success*), and the percentage of the overall runtime (time) that is consumed by the explorations of $\mathcal{N}_8^{BS}$. For all three indicators, the table includes the minimum (*min*), the average (*avg*), and the maximum (*max*) over the ten runs per instance and all instances of the respective library. Note that whenever an improving neighbor has been found, the VND continues the current ILS iteration up to the point when a joint local minimum of all neighborhoods including $\mathcal{N}_8^{BS}$ is reached.

The 0 values for #calls occur for rather small instances with only a few vertices of GTSP_LIB as well as for very large instances of LARGE_LIB (II). In the latter case, the total number of ILS iterations is smaller than 200 (see Section 2.5.1), i.e., the exploration of the other neighborhoods is so time-consuming that the high-quality component is never invoked. Anyway, the average number of explorations is between 24 and 137 with an average success percentage between approximately

				LARGE_LIB								
	GTSP_LIB			(I)			(II)			Total		
	#calls	success (%)	time (%)	#calls	success (%)	time (%)	#calls	success (%)	time (%)	#calls	success (%)	time (%)
min	0	0	0	8.90	2.05	0.06	0	0	0	0	0	0
avg	24.34	17.58	0.31	137.08	13.57	1.34	64.99	3.96	1.53	124.40	11.72	1.43
max	270.90	36.36	4.12	594.80	27.20	7.58	212.50	10.10	4.79	594.80	27.20	7.58

Table 2.8: Indicators for the use of the Balas-Simonetti neighborhood $\mathcal{N}_8^{BS}$ in the refined ILS.

4 and 17 percent. The percentage of run time consumed by the $\mathcal{N}_8^{BS}$ neighborhood exploration is always below 8 percent, and only 0.31 and 1.43 percent on average for the libraries GTSP_LIB and LARGE_LIB, respectively.

The impact that the use of the $\mathcal{N}_8^{BS}$ neighborhood exploration has on solution quality is summarized in Table 2.9. The entries *#best* and $\Delta(\%)$ have the same meaning as those of Table 2.6. The comparison between the refined ILS and the one without the component that fosters high-quality solutions (via $\mathcal{N}_8^{BS}$ neighborhood exploration) is consistent and in favor of the additional component: All *#best*-values are larger and all $\Delta(\%)$-values are smaller in the fully-equipped algorithm. The strongest result is that the $\mathcal{N}_8^{BS}$ neighborhood exploration halves the average $\Delta(\%)$-value for the GTSP_LIB.

			LARGE_LIB					
	GTSP_LIB		(I)		(II)		Total	
Algorithm	#best	$\Delta(\%)$	#best	$\Delta(\%)$	#best	$\Delta(\%)$	#best	$\Delta(\%)$
Refined ILS	9.20	0.01	3.00	0.60	0.00	3.26	2.36	1.16
Refined ILS w/o $\mathcal{N}_8^{BS}$	9.10	0.02	2.96	0.62	0.00	3.29	2.33	1.19

Table 2.9: Comparison of the refined ILS with and without the Balas-Simonetti neighborhood for high-quality solutions.

Comparison against GK, GLKH, and GLNS Finally, we compare the refined ILS with the basic ILS and the best-performing algorithms *GK*, *GLKH* and *GLNS* from the literature. Table 2.10 provides an overview with all entries as defined before for Table 2.6. The refined ILS clearly outperforms the basic ILS on GTSP_LIB and both parts of the LARGE_LIB. In comparison to the literature, results are not clear-cut and there is no unique winner. The *GLNS* outperforms all other algorithms on the MOM_LIB, BAF_LIB, and LARGE_LIB(I), where clusters are

generated so that they do *not* comprise sets of vertices that are mutually close. For the GTSP_LIB, however, *GLKH* and the refined ILS are best and on par regarding #best and Δ(%). No algorithm finds a BKS on the LARGE_LIB(II), however, *GLKH* is best regarding Δ(%), followed by the refinded ILS. It seems that the refined ILS can cope well with instances where the clusters consist of relatively close vertices.

	GTSP_LIB		MOM_LIB		BAF_LIB		LARGE_LIB (I)		LARGE_LIB (II)		LARGE_LIB Total	
Algorithm	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)	#best	Δ(%)
GK	9.10	**0.01**	8.44	0.03	8.11	0.29	2.77	0.76	—	—	—	—
GLKH	**9.20**	**0.01**	5.40	0.82	5.04	6.51	3.04	0.52	0.00	**2.56**	2.39	**0.97**
GLNS	8.73	**0.01**	**9.18**	**0.02**	**8.91**	**0.07**	**3.31**	**0.50**	0.00	6.46	**2.61**	1.76
Basic ILS	8.88	0.02	7.84	0.06	8.67	0.15	2.50	1.22	0.00	5.49	1.97	2.12
Refined ILS	**9.20**	**0.01**	7.84	0.06	8.67	0.15	3.00	0.60	0.00	3.26	2.36	1.16
Rank	1	1	3	3	2	2	3	3		2	3	2

Table 2.10: Comparison of the basic and refined ILS with best-performing algorithms from the literature.

2.6 Conclusions

The paper has introduced an effective ILS for solving symmetric instances of the GTSP. The basic ILS combines several neighborhoods into a random VND that are then used as the local-search component of the ILS. For the VND, we introduced three new neighborhoods (GTSP-tailored Balas-Simonetti neighborhood, adapted Gutin neighborhood, and String Relocation+) that allow simultaneous modifications of the sequence of the clusters and the selection of vertices per cluster. Effective neighborhood expoloration algorithms for these neighborhoods have been developed and analyzed. Unexpectedly, this straightforward design of an ILS already gives reasonable results on standard benchmarks.

One finding of our experimental studies is that a single ILS setup that is not tailored to characteristics of the GTSP instance at hand can be easily improved. Indeed, while some instances have geometrically defined clusters that comprise mutually close vertices, other instances systematically spread the vertices of all clusters. The distinction of these cases is crucial, and we have defined a refined ILS that chooses priorities for the neighborhoods in the VND according to the size N and the average relative inner-cluster distance. Moreover, when a strict time budget for the ILS is imposed, the cooling schedule for the record-to-record-based acceptance criterion should be adapted. Estimating the number of ILS iterations

and choosing a properly aligned cooling schedule turned out to be simple but effective.

With these refinements, the computational results also show that the new GTSP neighborhoods contribute significantly to the success of the ILS versions. This can be seen, for example, from the fact that the Balas-Simonetti neighborhood with $k = 8$ further improves elite solutions. Moreover, the good performance of the adapted Gutin neighborhood becomes evident within the refined ILS, where this neighborhood is used with highest priority/first in the nested VND II.

The refined ILS is particularly powerful on the `GTSP_LIB` and improves the results of the basic ILS on the `LARGE_LIB` (we have found one new BKS during experimentation with preliminary versions of the ILS, see Appendix). For the latter library and the largest instances therein, the refined ILS performs slightly better than the GLNS algorithm, which, in turn, outperforms all other algorithms from the literature on the `MOM_LIB` and `BAF_LIB`. Overall, there is no clear winner algorithm that outperforms all others on all the available libraries that comprise GTSP instances with rather different characteristics. Indeed, the algorithms GK, GLKH, GLNS, and our ILS follow completely different solution paradigms. An overall superior algorithm may require a clever combination of these paradigms. What we offer with our work is three new exponentially-sized neighborhoods (Balas-Simonetti, String Relocation+, and an adaptation of the Gutin neighborhood) that may contribute to the attempt to design an overall superior algorithm.

For the future, further accelerating neighborhood explorations with established techniques like granular search (Toth and Vigo, 2003; Schröder *et al.*, 2020), don't look bits (Hoos and Stützle, 2005, p. 375), and dynamic sequential/radius search (Irnich *et al.*, 2006; Gauthier and Irnich, 2024) as well as with new ideas to be worked out is a promising research path.

Acknowledgement

This research was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant IR 122/7-2. This support is gratefully acknowledged.

Bibliography

Balas, E. (1999). New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research*, **86**, 529–558.

Balas, E. and Simonetti, N. (2001). Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS Journal on Computing*, **13**(1), 56–75.

Ben-Arieh, D., Gutin, G., Penn, M., Yeo, A., and Zverovitch, A. (2003). Transformations of generalized ATSP into ATSP. *Operations Research Letters*, **31**(5), 357–365.

Bentley, J. J. (1992). Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, **4**(4), 387–411.

Bontoux, B. (2008). *Techniques hybrides de recherche exacte et approchée : application à des problèmes de transport*. Ph.D. thesis, Université d'Avignon. Français.

Bontoux, B., Artigues, C., and Feillet, D. (2010). A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem. *Computers & Operations Research*, **37**(11), 1844–1852.

Cacchiani, V., Muritiba, A., Negreiros, M., and Toth, P. (2011). A multistart heuristic for the equality generalized traveling salesman problem. *Networks*, **57**, 231–239.

Dueck, G. (1993). New optimization heuristics. *Journal of Computational Physics*, **104**(1), 86–92.

Feillet, D., Dejax, P., Gendreau, M., and Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, **44**(3), 216–229.

Fischetti, M., Gonzáles, J., and Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, **45**(3), 378–394.

Gauthier, J. B. and Irnich, S. (2024). Inter-depot moves and dynamic-radius search for multi-depot vehicle routing problems. *Discrete Applied Mathematics*, **346**, 131–153.

Glover, F. (1996). Finding a best traveling salesman 4-opt move in the same time as a best 2-opt move. *Journal of Heuristics*, **2**, 169–179.

Gutin, G. and Karapetyan, D. (2009). Generalized traveling salesman problem reduction algorithms. arXiv 0804.0735v2.

Gutin, G. and Karapetyan, D. (2010). A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, **9**(1), 47–60.

Gutin, G., Yeo, A., and Zverovitch, A. (2007). Exponential neighborhoods and domination analysis for the TSP. In *The Traveling Salesman Problem and Its Variations*, Combinatorial Optimization, pages 223–256. Springer, Boston, MA.

Gutin, G., Karapetyan, D., and Krasnogor, N. (2008). Memetic algorithm for the generalized asymmetric traveling salesman problem. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, pages 199–210. Springer Berlin Heidelberg.

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, **130**(3), 449–467.

Hansen, P. and Mladenović, N. (2005). Variable neighborhood search. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 8, pages 211–238. Springer, Boston, MA.

Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, **126**, 106–130.

Helsgaun, K. (2015). Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun Algorithm. *Mathematical Programming Computation*, **7**(3), 269–287.

Henry-Labordere, A. (1969). The record balancing problem: A dynamic programming solution of a generalized travelling salesman problem. *RIRO B-2*, pages 43–49.

Hoos, H. H. and Stützle, T. (2005). *Stochastic Local Search*. Morgan Kaufmann, Amsterdam.

Hu, B. and Raidl, G. R. (2008). Effective neighborhood structures for the generalized traveling salesman problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 36–47. Springer Berlin Heidelberg.

Irnich, S., Funke, B., and Grünert, T. (2006). Sequential search and its application to vehicle-routing problems. *Computers & Operations Research*, **33**(8), 2405–2429.

Johnson, D. S. and McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 8, pages 215–310. Wiley, Chichester.

Karapetyan, D. and Gutin, G. (2011). Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem. *European Journal of Operational Research*, **208**(3), 221–232.

Karapetyan, D. and Gutin, G. (2012). Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem. *European Journal of Operational Research*, **219**(2), 234–251.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1987). Optimization by simulated annealing. In *Readings in Computer Vision*, pages 606–615. Elsevier.

El Krari, M., Ahiod, B., and El Benani, Y. B. (2020). A memetic algorithm based on breakout local search for the generalized traveling salesman problem. *Applied Artificial Intelligence*, **34**(7), 537–549.

El Krari, M., Ahiod, B., and El Benani, Y. B. (2021). A pre-processing reduction method for the generalized travelling salesman problem. *Operational Research*, **21**, 2543–2591.

Laporte, G. and Semet, F. (1999). Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, **3**7(2), 114–120.

Laporte, G., Asef-Vaziri, A., and Sriskandarajah, C. (1996). Some applications of the generalized travelling salesman problem. *Journal of the Operational Research Society*, **47**(12), 1461–1467.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, **44**, 2245–2269.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, **2**1(2), 498–516.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). Iterated local search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 320–353. Springer, New York, NY

Mestria, M., Ochi, L. S., and de Lima Martins, S. (2013). GRASP with path relinking for the symmetric Euclidean clustered traveling salesman problem. *Computers & Operations Research*, **40**(12), 3218–3229.

Noon, C. E. (1988). *The generalized traveling salesman problem.* Ph.D. thesis, University of Michigan.

Noon, C. E. and Bean, J. C. (1991). A Lagrangian based approach for the asymmetric generalized traveling salesman problem. *Operations Research*, **39**(4), 623–632.

Noon, C. E. and Bean, J. C. (1993). An efficient transformation of the generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, **31**(1), 39–44.

Pintea, C. M., Pop, P. C., and Chira, C. (2017). The generalized traveling salesman problem solved with ant algorithms. *Complex Adaptive Systems Modeling*, **5**(1).

Reihaneh, M. and Karapetyan, D. (2012). An efficient hybrid ant colony system for the generalized traveling salesman problem. *Algorithmic Operations Research*, **7**, 22–29.

Reinelt, G. (1991). TSPLIB - A traveling salesman problem library. *ORSA Journal on Computing*, **3**(4), 376–384.

Ren, X., Wang, X., Wang, Z., and Wu, T. (2020). Parallel DNA algorithms of generalized traveling salesman problem-based bioinspired computing model. *International Journal of Computational Intelligence Systems*, **14**(1), 228.

Renaud, J. and Boctor, F. F. (1998). An efficient composite heuristic for the symmetric generalized traveling salesman problem. *European Journal of Operational Research*, **108**(3), 571–584.

Saksena, J. P. (1970). Mathematical model of scheduling clients through welfare agencies. *Journal of the Canadian Operational Research Society*, **8**, 185–200.

Schröder, C., Gauthier, J. B., Gschwind, T., and Schneider, M. (2020). In-depth analysis of granular local search for capacitated vehicle routing. Working Paper DPO-2020-03, Deutsche Post Chair – Optimization of Distribution Networks, RWTH Aachen University, Aachen, Germany.

Silberholz, J. and Golden, B. (2007). The generalized traveling salesman problem: A new genetic algorithm approach. In *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, pages 165–181. Springer New York, NY.

Smith, S. L. and Imeson, F. (2017). GLNS: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, **87**, 1–19.

Snyder, L. and Daskin, M. (2006). A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*, **174**(1), 38–53.

Subramanian, A., Drummond, L. M. A., Bentes, C., Ochi, L. S., and Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, **37**(11), 1899–1911.

Tasgetiren, M. F., Suganthan, P. N., and Pan, Q. Q. (2007). A discrete particle swarm optimization algorithm for the generalized traveling salesman problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO 2007*, pages 158–167.

Toth, P. and Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, **15**(4), 333–346.

Yang, J., Shi, X., Marchese, M., and Liang, Y. (2008). An ant colony optimization method for generalized TSP problem. *Progress in Natural Science*, **18**(11), 1417–1422.

Appendix

During experimentation with different ILS setups, we have found a new best-known solution (BKS) for the instance `287u1432` of the `LARGE_LIB` with cost of 54,433:

G-tour: (295 296 292 291 1000 1001 286 281 262 260 239 265 267 270 268 187 233 227 224 223 244 99 103 8 2 1 1425 1423 1414 1421 115 111 114 215 206 153 154 199 198 177 196 183 184 362 358 354 357 400 402 436 437 567 574 576 426 425 418 319 416 412 322 325 329 330 331 303 302 311 312 788 783 782 585 586 779 775 771 768 805 808 802 801 799 909 915 921 923 895 893 867 868 891 888 882 945 949 952 956 957 954 876 877 879 870 854 853 849 843 832 830 829 824 823 821 859 818 816 752 758 759 747 739 740 734 728 723 717 716 710 702 700 698 687 672 671 666 658 656 652 646 650 632 628 627 623 592 596 604 610 547 540 526 521 514 512 509 508 504 499 485 486 487 488 441 455 458 459 467 391 392 381 380 373 367 371 370 168 164 161 160 148 146 141 137 124 126 130 1391 1388 1378 1377 1374 1381 1382 1364 1353 1335 1333 1316 1329 1328 1326 1323 1321 1309 1302 1299 1283 1280 1278 1274 1273 1184 1254 1255 1244 1247 1235 1234 1223 1226 1220 1217 1213 1212 1211 1106 1105 1110 1095 1092 1086 1082 967 970 1078 976 977 1072 1069 1066 1064 1130 1121 1120 1200 1198 1195 1125 1137 1177 1173 1169 1168 67 68 48 37 39 31 28 25 89 88 90 248 83 82 254 256 1010 1019 1023 1024 1034 1148 1147 1041 1038 1045 987 986 939 938 937 929 926 295)

Chapter 3

Using Public Transport in a 2-Echelon Last-Mile Delivery Network

Jeanette Schmidt, Christian Tilk, Stefan Irnich

Abstract

In this paper, we investigate the integration of public transport services into last-mile delivery networks. The free capacity of an already established public transport system that operates according to a given timetable on predetermined lines is used to carry goods from outside of the city into the city center. Dedicated bus or tram stations of the public transport network serve as satellites. From these satellites, city freighters pick up the goods and deliver them to the final customers. The main contribution is an extensive computational study, in which we consider various scenarios of the shared part of the public transport network and investigate different hierarchical objectives minimizing two of the three key indicators (number of city freighters, routing costs, and number of trips). We quantify the tradeoff between the key indicators for different objectives and the impact of limiting the capacity of public transport vehicles on two instance sets with different demand distributions. The key instrument to conduct all these experiments are effective exact and heuristic branch-price-and-cut algorithms that we develop and evaluate in detail.

3.1 Introduction

The Covid-19 pandemic has led to an increase in e-commerce freight (United Nations, 2021; OECD, 2020). For example, over 4 billion deliveries had to be handled in Germany in 2020, which is an increase of more than ten percent compared to the year 2019 (BIEK, 2021). This growing e-commerce poses a major challenge for logistic service providers, especially when it comes to last-mile delivery. While minimizing logistic costs, carriers have to deal with restrictions imposed by urban development, environmental policies, and operational issues like traffic congestion and strict parking regulation (Zhou *et al.*, 2018).

One of the most common designs for handling and reducing the large number of delivery vehicles going into cities, often delivering small quantities per customer, is to consolidate this fragmented volume at *urban distribution centers* (UDCs) located in cities' outskirts (Savelsbergh and van Woensel, 2016). In resulting two-echelon systems, first-echelon vehicles transport goods from the UDC to satellites, at which second-echelon vehicles pick up goods and deliver them to the customers. In this paper, we consider the special case of this system in which the first-echelon utilizes existing public services for passenger transportation, e.g., metro, tram, or bus lines. The resulting two-echelon system is multi-modal and belongs to the group of shared mobility systems (Mourad *et al.*, 2019).

Practitioners have already launched several projects utilizing scheduled services for last-mile deliveries, e.g., "GüterBim" in Vienna, "TramFret" in Saint-Étienne, "Cargo-Tram" in Zurich, and the "LastMileTram" in Frankfurt (Schocke *et al.*, 2020). While effective engineering solutions have been developed for the physical design of the system (dedicated compartments, trailers, transport boxes etc.), the operational planning for such systems is rarely studied. In this work, we therefore introduce the *last-mile delivery problem with scheduled lines* (LMDPSL) that models the associated operative planning problem: On the first echelon, scheduled public transport vehicles are used to carry containers with goods from outside of the city into the city center. Dedicated stations of the public transport services are used as satellites. In such a setting, the schedules of first-echelon vehicles are known and fixed, i.e., routes and arrival/departure times are known in advance. The question is here how to best utilize the provided transportation capacities of the first echelon. On the second-echelon, electric vehicles (Schneider *et al.*, 2014) and/or cargo bikes (Anderluh *et al.*, 2019), henceforth called *city freighters* (CFs), pick up containers from the stations and deliver the contained goods to the customers in a multi-trip setting. Paradiso *et al.* (2020) stress the importance of considering multiple trips for city logistics and last-mile delivery. The outstanding prices of space in many big cities (e.g., £14355/m^2/*year* in London, Furmanik, 2019) justify the assumption that temporarily storing goods at stations is not possible. Consequently, the transport of goods must respect exact operation and load

synchronization constraints (following the taxonomy of Drexl, 2012). Moreover, our definition of the LMDPSL features a joint limited capacity of certain subsets of stops in terms of the number of containers that can be transferred. This enables the modeling of several practically relevant constraints such as limited space in the public transport vehicle.

The main contribution of the paper at hand is an extensive computational study of the LMDPSL considering possible scenarios of the public transport system (as they occur in different cities) under various objectives (allowing to study also some tactical planning problems in connection with the LMDPSL) on two instance sets with different demand distributions. We assume hierarchical objectives minimizing two of the three key indicators which are the number of employed CFs, routing costs, and the number of trips. The latter key indicator, the number of trips, is rarely studied in the multi-trip literature, although, from an operational point of view, the start of a new trip can be seen as the most crucial part in the operative process: Public transport vehicle and CF must be perfectly synchronized in time and space taking also capacities into account (Drexl, 2012). Minimizing the number of trips, therefore, reduces the risk that public transport vehicles and CFs miss each other.

The key instrument to conduct experiments and meaningful analyses is that LMDPSL instances can be solved exactly or with a relatively small optimality gap. For this purpose, we model the LMDPSL as a variant of the vehicle routing problem with time windows and intermediate replenishment. Solution methods for these types of problems are already mature and allow the solution of at least medium-sized instances. We present *branch-price-and-cut* (BPC) algorithms that allow the exact and heuristic solution of the LMDPSL, because they include the most recent and cutting-edge algorithmic components (Costa *et al.*, 2019). Our focus is however *not* the development and description of the BPC algorithms, because their algorithmic components have been well described in several recent works (e.g., Pecin *et al.*, 2017b; Pessoa *et al.*, 2020). For this reason, the description of the BPC algorithms (Section 3.4) has been written as compact and concise as possible.

The computational study (Section 3.5) consists of several parts and is not tailored to the setting of a specific city. Instead, our intention was to be rather general and to analyze different standard scenarios. To this end, we create four different scenarios for the shared part of the public transport network. In these scenarios, the scheduled public transport vehicles operate in lines following a circle and/or cross with a central station in the middle. Two pairs of scenarios are included in each other (restriction and relaxation), which allow studying the possibility to select some lines and stops out of the complete public transport network. Moreover, we investigate four different hierarchical objectives and evaluate the exact

and heuristic BPC algorithm using all objectives. A scenario-wise comparison of the different objectives is possible because the same customer sets are used for all scenarios. In addition, we analyze the tradeoff between the key indicators when different objectives are employed. In a further study, the impact of limiting the capacity of public transport vehicles is evaluated.

The remainder of the paper is structured as follows: We review the pertinent literature in Section 3.2. Section 3.3 formally introduces the LMDPSL. In Section 3.4, we present the BPC algorithms including the column-generation procedure, valid inequalities, and the branching scheme. The results of an extensive computational study are presented in Section 3.5. The paper closes with final conclusions drawn in Section 3.6.

3.2 Literature Review

The LMDPSL is a special case of the *two-echelon vehicle routing problem* (Cuda *et al.*, 2015) and belongs to the class of *vehicle routing problems with intermediate stops* (see Schiffer *et al.*, 2019, for an overview). We focus on the operational planning perspective and only review the literature on integrating freight transportation and public transportation. The systematic literature review by Elbert and Rentschler (2021) covers the pertinent literature on qualitative approaches including also important quantitative works.

Masson *et al.* (2015) study the *mixed urban transportation problem* (MUTP) which is a special case of the LMDPSL. They propose a *mixed-integer programming* (MIP) formulation and an *adaptive large neighborhood search* (ALNS). The latter is tested on 15 real-world instances of the city La Rochelle (France) that contain up to 303 customers. Compared to the LMDPSL, the MUTP has the following three limitations: (1) only a single bus line is considered, (2) a restricted container capacity constraint bounds the number of containers that can be unloaded simultaneously at each station, and (3) the MUTP considers one objective only (minimizing the number of CFs first and the routing cost second).

Another problem related to the LMDPSL is the *pickup and delivery problem with time windows and scheduled lines* (PDPTW-SL, see Ghilas *et al.*, 2016b), which generalizes the pickup and delivery problem. The PDPTW-SL allows indirect shipments where a request is picked up at its origin location by a CF and brought to a public transport station. From there the request is carried to another station with a public transportation service. Ghilas *et al.* (2016b) introduced the term *scheduled line* for this service. Finally, the request is picked up at the station by another CF that delivers it to its destination location. Ghilas *et al.* propose an arc-based MIP formulation to solve the PDPTW-SL. The MIP solver CPLEX admits the solution of small-sized instances with up to 11 requests and two scheduled lines

proving optimality within a time limit of 24 hours. To achieve better results more quickly, Ghilas *et al.* (2016a) developed an ALNS in a follow-up publication. The ALNS allows the consideration of larger instances and is significantly faster than the direct MIP-based approach. For almost all of the small instances, it finds an optimal solution within an average computation time of just two seconds. Larger instances with up to 100 requests are solved in less than 40 minutes. A later study by Ghilas *et al.* (2016c) extends the ALNS and embeds it into a sample average approximation framework for the PDPTW-SL with stochastic demands. Finally, Ghilas *et al.* (2018) present an exact branch-and-price algorithm which obtains good results on instances with up to 60 requests.

Mourad *et al.* (2020) employ a similar ALNS-based approach for a stochastic version of the PDPTW-SL in which autonomous pickup-and-delivery robots take the role of the CFs. The problem is modeled as a two-stage stochastic problem with the objective to minimize the overall transportation costs. The proposed sample average approximation method together with the ALNS algorithm can cope with instances with up to 60 requests.

Behiri *et al.* (2018) consider the *freight-rail-transport-scheduling problem* that captures the real-life problem studied in the *Grand Paris project*, in which freight and passenger transport are combined using the rail network of Paris. The authors present a MIP formulation and several heuristics solving instances with up to 150 freight requests shipped on a single rail line.

In all contributions mentioned above, freight and passengers share the same public transport vehicle. In contrast, Ozturk and Patrick (2018) focus on sharing the infrastructure only, i.e., dedicated freight trains and regular passengers trains share the same railway line. Similarly, Fatnassi *et al.* (2015) propose a shared freight and passenger on-demand rapid transit system. They consider passenger and freight rapid transit vehicles that offer specific on-demand transportation services from a departure station to a destination station in the shared network.

3.3 Modeling and Problem Definition

The leading idea behind our modeling of the LMDPSL is that with given scheduled services, the decisions for the first echelon concern the best utilization of the provided transportation capacities. There are no routing and scheduling decisions to make for the first echelon. As a consequence, the problem definition can abstract from detailed timetables of the first-echelon vehicles. Even more, not all first-echelon public transport vehicles are equipped with a dedicated compartment or trailer for goods transportation. Only a few of these vehicles have the necessary equipment and are therefore relevant for the LMDPSL definition. A realistic setting is that these buses or trams follow, like every other bus or tram, regular

lines so that they can perform several *circulations* (back and forth operations of a scheduled service along a line) over the day. Moreover, in a more strategic phase of the system's design, potential stations suited for a transfer of goods have been identified. Combining both types of information, it is exactly known in advance when and where a potential transfer of goods between the first and second echelon can take place. We denote the combination of a point in time and a station a *stop*. The reader should not confuse a stop with, e.g., every halt where a tram or bus lets passengers leave and enter.

Formally, the LMDPSL can be defined on a directed graph $D = (V, A)$ with vertex set V and arc set A. This digraph D is used to model the CFs' movements in the system. The vertex set V comprises $\{0, 0'\} \cup N \cup H$, where 0 and $0'$ denote copies of the CFs' depot, N the set of customers, and H the set of stops. We explain the latter sets in more detail now.

Let the customer set be N. Each customer $i \in N$ has an integer demand $q_i > 0$. Moreover, the service at customer i has to start in the time window $[a_i, b_i]$, where a_i (b_i) denotes the earliest (latest) time at which the service can start. The handling time at customer i is given by $s_i \geq 0$.

We assume that a set of public transport vehicles can be used to deliver goods into the city center. These vehicles follow given public transport lines and operate as *scheduled services* a.k.a. *scheduled lines*. There can be one line or several lines, each of them visiting a sequence of several *stations* in circulation. Each of these visits is defined as a stop, and we denote by H the set of all stops. Hence, a stop $h \in H$ is defined by a station (the physical location), the given and fixed point in time e_h when the station is reached by the public transport vehicle, and a handling time s_h describing how long a CF must stay when it is replenished by this stop. Since temporarily storing goods at stations is not possible, the replenishment of a CF happens over the time period from a_h to $a_h + s_h$. Hence, we associate the time window $[a_h, b_h] = [e_h, e_h]$ with the stop $h \in H$. This modeling approach widely decouples first-echelon decisions on capacity utilization from second-echelon decisions concerned with the routing and scheduling of the CFs. For the sake of convenience, we define $q_h = 0$ for all $h \in H$.

Goods are transported in containers (see Schocke *et al.*, 2020), each with capacity Q (measured in the same unit as customers' demands). Subsets $\bar{H} \subset H$ of stops can have a joint limited capacity in terms of the number of containers that can be transferred. Such general joint capacities allow to model several practically relevant constraints: Limited space regarding the number of containers to transport in a public transport vehicle can be considered by the subset of all stops within one circulation. Moreover, limited space and time may impose that only a limited number of containers can be unloaded at a stop. Let $\mathcal{H}$ be the set of all capacity-constrained sets of stops, i.e., each $\bar{H} \in \mathcal{H}$ imposes that not more than

$C_{\bar{H}}$ containers (the capacity) can be transferred at all stops $h \in \bar{H}$ together.

A homogeneous fleet of K CFs is stationed at the depot, i.e., each CF starts its route from the depot 0 and performs one or several *trips*. A trip consists of picking up goods from a stop $h \in H$ and delivering them to some customer(s). The length of such a *trip* is limited by the time windows and the capacity of the CF. As there are no storage possibilities at a stop h, the CFs cannot arrive later than time b_h at the stop. The capacity of a CF is one container. This implies also that a CF must be empty at the end of each trip, because the container used during the trip is replaced by the new container arriving with the public transport vehicle. In particular, it is operationally infeasible to transfer boxes between containers at a public transport station. After completing one trip, the CF can either perform another trip starting from a different stop, or return to the destination depot $0'$. The start depot 0 and destination depot $0'$ have a time window spanning the whole planning horizon.

Arcs represent possible movements of the CFs (we do not model the movement of public transport vehicles because they have a fixed line and schedule). The arc set A contains the following five types of arcs $(i, j) \in A$:

- from the origin depot $i = 0$ to stop vertices $j \in H$;
- from stop vertices $i \in H$ to customer vertices $j \in N$;
- between customer vertices $i, j \in N$, $i \neq j$;
- from customer vertices to stop vertices, i.e., $i \in N$ and $j \in H$; and
- from customer vertices $i \in N$ to the destination depot $j = 0'$.

A routing cost d_{ij} and a non-negative travel time t_{ij} (including the handling time s_i at vertex i) are associated with each arc $(i, j) \in A$. Time-window infeasible arcs can be filtered out so that $A \subset \{(i, j) \in V \times V : a_i + t_{ij} + \leq b_j\}$ is assumed in the following.

For the reader's convenience, Table 3.1 summarizes all sets and parameters used in the definition of the LMDPSL.

The task of the LMDPSL is to determine a cost-minimal set of at most K feasible multi-trip routes such that each customer is served exactly once. We consider three *key indicators* (i.e., number of CFs, routing cost, and number of trips) that are relevant from an operational point of view. While the number of CFs reflects the number of employees needed on the second echelon, the number of trips is relevant for two reasons. On the one hand, the replenishment process of a second echelon vehicle can be seen as the most time-critical part in the operative process. To ensure that the reloading process runs smoothly, CF and public transport vehicle must be perfectly synchronized. Minimizing the number of trips, reduces the risk that public transport vehicle and CFs miss each other. On the other hand, it can be seen as an indicator of the utilization of the containers. The most interesting combinations of these key indicators leads us to four different

Sets		Parameters	
V	set of vertices	a_i	earliest service start at customer $i \in N$
A	set of arcs	b_i	latest service start at customer $i \in N$
H	set of stops	c_{ij}	costs on each arc $(i,j) \in A$
$\bar{H} \subset H$	set of stops with limited capacity	d_{ij}	routing costs on each arc $(i,j) \in A$
$\mathcal{H}$	set of capacity-constrained stops	e_h	time when station $h \in H$ is reached by a
N	set of customers		public transport vehicle
V	set of vertices	K	number of CFs
		Q	container capacity
		$Q_{\bar{H}}$	capacities of all stops $h \in \bar{H}$ together
		q_i	demand of each customer $i \in N$
		s_h	service time at a stop $h \in H$
		s_i	handling time at customer $i \in N$
		t_{ij}	travel time on each arc $(i,j) \in A$

Table 3.1: Mathematical notation.

hierarchical *objectives*:

O1: minimize the number of CFs first and the routing costs second;
O2: minimize the routing costs first and the number of CFs second;
O3: minimize the number of trips first and the number of CFs second; and
O4: minimize the number of trips first and the routing costs second.

The four objectives can be represented in a unified manner by defining *arc costs* in the following way: Given *routing costs* d_{ij} per arc $(i,j) \in A$ are multiplied with α. Arcs leaving the depot include the fixed cost β to account for the number of CFs in use. Likewise, arcs leaving a stop include the fixed cost γ to account for the number of trips performed. Hence, the cost of an arc $(i,j) \in A$ is defined as:

$$c_{ij} = \begin{cases} \alpha \cdot d_{ij} + \beta, & \text{if } i = 0 \\ \alpha \cdot d_{ij} + \gamma, & \text{if } i \in H \\ \alpha \cdot d_{ij}, & \text{otherwise} \end{cases}$$

Depending on the objective, appropriate values for α, β, and γ can be chosen to obtain a proper hierarchy. For example, Objective O3 requires $\gamma \gg \beta \gg \alpha$. Summarizing, the objectives O1 to O4 have been chosen deliberately. However, the BPC-based solution approach that we present and evaluate later also works for other weighted combinations of the key indicators.

Next, we formally define trips and routes. A *trip* τ is a pair of a path $P = (j_0, j_1, \ldots, j_k)$ in D and a time schedule $(T_0, T_1, \ldots, T_k)$ with $k \geq 1$. A trip is feasible, if the following conditions hold:

(Trip1): $j_0 \in H$;
(Trip2): $j_i \in N$ for all $i \in \{1, \ldots, k\}$;
(Trip3): $T_i \in [a_{j_i}, b_{j_i}]$ for all $i \in \{0, 1, \ldots, k\}$;

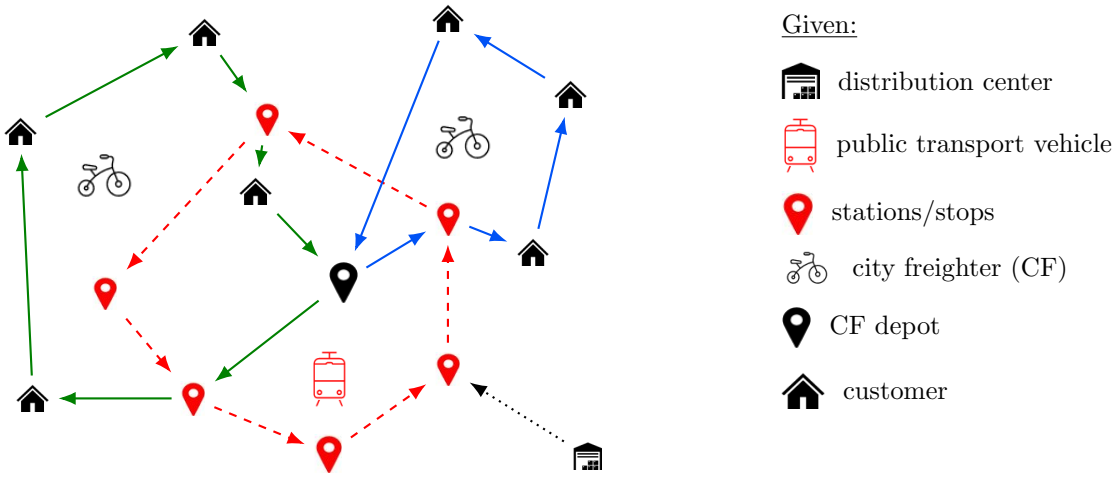


Figure 3.1: Example of an LMDPSL instance with solution.

(Trip4): $T_i + t_{j_i,j_{i+1}} \leq T_{i+1}$ for all $i \in \{0, 1, \ldots, k-1\}$; and
(Trip5): $\sum_{i=1}^{k} q_{j_i} \leq Q$.
These conditions ensure that each trip starts at a stop, visits some customer(s) afterwards, and respects travel times and customer time windows as well as the capacity of the CF. The cost of a trip is defined as the sum of the cost of all arcs traversed, i.e., $c_\tau = \sum_{i=0}^{k-1} c_{j_i,j_{i+1}}$.

A *route* r is a path $(0, P^1, \ldots, P^m, 0')$ with subpaths $P^v = (j_0, j_1, \ldots, j_{k_v})$ together with schedules $((T^1), \ldots, (T^m))$ such that each $\tau^v = (P^v, (T_i^v)_{i=0}^{k^v})$ is a feasible trip for $v \in \{1, 2, \ldots, m\}$. The route r is feasible if the entire schedule is time-consistent, i.e.,

(Route1): $a_0 + t_{0,j_0^1} \leq T_0^1$;
(Route2): $T_{k^v}^v + t_{j_{k^v}^v, j_0^{v+1}} \leq T_0^{v+1}$ for all $v \in \{1, \ldots, m-1\}$; and
(Route3): $T_{k^m}^m + t_{j_{k^m}^m, 0'} \leq b_{0'}$.
The cost c_r of a route is defined as the cost of all trips plus the cost of the connecting arcs, i.e.,

$$c_r = \sum_{v=1}^{m} c_{\tau_v} + \left(c_{0,j_0^1} + \sum_{v=1}^{m-1} c_{j_{k^v}^v, j_0^{v+1}} + c_{j_{k^m}^m, 0'} \right)$$

Figure 3.1 shows an example of an LMDPSL instance with seven customers and one public transport line. The line starts at the UDC and visits six stations in circulation (depicted with dashed red arcs). Additionally, the figure shows a solution in which two CF routes are performed to serve all customer. The first CF route (depicted in green) is composed of two trips: The first trip picks a container at a station and serves three customers and the second trip picks a container for

only one customer from another stop. The second CF route (depicted in blue) serves another three customers with only a single trip that picks up a container at a third stop.

3.4 Route-based Formulation and Branch-Price-and-Cut Algorithm

We denote by Ω the set of all feasible routes as defined in the previous section. For each vertex $i \in V$ and each route $r \in \Omega$, the integer coefficient p_{ir} indicates how often route r visits vertex i. Similarly, for each set $\bar{H} \in \mathcal{H}$ of capacity-constrained stops and each route $r \in \Omega$, we define the capacity consumption of route r as $p_{\bar{H}r} = \sum_{h \in \bar{H}} p_{hr}$. The following route-based formulation of the LMDPSL uses binary variables λ_r indicating whether a route $r \in \Omega$ is selected to be part of the optimal solution ($\lambda_r = 1$) or not ($\lambda_r = 0$). The model reads as follows:

$$\min \quad \sum_{r \in \Omega} c_r \lambda_r \tag{3.1a}$$

$$\text{subject to} \quad \sum_{r \in \Omega} p_{ir} \lambda_r = 1, \qquad \forall i \in N \tag{3.1b}$$

$$\sum_{r \in \Omega} p_{\bar{H}r} \lambda_r \leq C_{\bar{H}}, \qquad \forall \bar{H} \in \mathcal{H} \tag{3.1c}$$

$$\sum_{r \in \Omega} \lambda_r \leq K \tag{3.1d}$$

$$\lambda_r \in \{0, 1\}, \qquad \forall r \in \Omega \tag{3.1e}$$

With the definition of the cost c_r from Section 3.3, the objective (3.1a) is one of the hierarchical Objectives O1 to O4 depending on the choice of α, β, and γ. The set-partitioning constraints (3.1b) ensure that each customer is served exactly once. The capacity constraints (3.1c) limit the number of containers (one for each trip) that can be handled at a particular subset of stops. The set of stops for replenishing CFs and capacity constraints (3.1c) completely capture the impact of the first echelon on the second-echelon CF routes. The number of used CFs is bounded by the fleet size in (3.1d), and the variable domains are given by (3.1e).

For realistic instances of the LMDPSL, formulation (3.1) contains a huge number of feasible routes $r \in \Omega$ so that this model can only be solved with specialized techniques, e.g., a BPC algorithm (Desaulniers *et al.*, 2005). As BPC is a standard approach for solving vehicle routing problems (Costa *et al.*, 2019), we only sketch its main algorithmic components: the generation of route variables (Section 3.4.1), the strengthening of the linear relaxation of model (3.1) with non-robust subset-row inequalities (Section 3.4.2), and the branching scheme (Section 3.4.3).

3.4.1 Column Generation

Let $(\pi_i)_{i\in N}$ be the dual prices of constraints (3.1b), $(\kappa_{\bar{H}})$ be the dual prices of constraints (3.1c), and μ be the dual price of (3.1d). Given these values, the pricing problem must generate at least one feasible negative reduced-cost route r or prove that no such route exists. The reduced cost of an arbitrary route r is

$$\tilde{c}_r = c_r - \sum_{i\in N} p_{ir}\pi_i - \sum_{\bar{H}\in\mathcal{H}} p_{\bar{H}r}\kappa_{\bar{H}} - \mu.$$

The pricing problem can be modeled as a *shortest-path problem with resource constraints* (SPPRC, Irnich and Desaulniers, 2005) and solved by means of a labeling algorithm. To this end, we define reduced costs for all arcs $(i,j) \in A$ as:

$$\tilde{c}_{ij} = \begin{cases} c_{ij} - \pi_i, & \text{if } i \in N \\ c_{ij} - \mu, & \text{if } i = 0 \\ c_{ij} - \sum\limits_{\bar{H}\in\mathcal{H}: i\in\bar{H}} \kappa_{\bar{H}}, & \text{if } i \in H \end{cases}$$

In our labeling algorithm for the LMDPSL, a forward label $\mathcal{L}_i$ represents a forward partial path $(0,\ldots,i)$ in D starting at the origin depot 0 and ending at some vertex $i \in V$. A label $\mathcal{L}_i$ comprises the following attributes:

T_i^{rdc}: the cumulated reduced cost along the partial path;
T_i^{load}: the cumulated demand of the current trip including vertex i;
T_i^{time}: the earliest start time of service at vertex i; and
$T_i^{cust,k}$: the number of times a customer $k \in N$ is served in the path.

Starting from an initial label $\mathcal{L}_0 = (T_0^{rdc}, T_0^{load}, T_0^{time}, (T_0^{cust,k})_{k\in N}) = (0, Q, a_0, \mathbf{0})$, the labeling algorithm propagates labels towards the destination depot $0'$ with the help of *resource extension functions* (REFs, Irnich, 2008). We set the load resource $T_0^{load} = Q$ to ensure that the CF visits a stop prior to a customer. An arbitrary label $\mathcal{L}_i = (T_i^{rdc}, T_i^{load}, T_i^{time}, (T_i^{cust,k})_{k\in N})$ is extended along an arc $(i,j) \in A$ creating a new label $\mathcal{L}_j = (T_j^{rdc}, T_j^{load}, T_j^{time}, (T_j^{cust,k})_{k\in N})$ for the partial path $(0,\ldots,i,j)$ using the following REFs:

$$T_j^{rdc} = T_i^{rdc} + \tilde{c}_{ij} \tag{3.2a}$$

$$T_j^{load} = \begin{cases} T_i^{load} + q_j, & \text{if } j \in N \\ 0, & \text{otherwise} \end{cases} \tag{3.2b}$$

$$T_j^{time} = \max\{a_j, T_i + t_{ij}\} \tag{3.2c}$$

$$T_j^{cust,k} = \begin{cases} T_i^{cust,k} + 1, & \text{if } j = k \in N, \\ T_i^{cust,k}, & \text{otherwise} \end{cases}, \qquad \forall k \in N \tag{3.2d}$$

The new partial path is feasible if the label fulfills $T_j^{time} \leq b_j$, $T_j^{load} \leq Q$, and $T_j^{cust,k} \leq 1$ for all $k \in N$ (time-window, capacity, and elementarity constraints). Infeasible labels are directly discarded. To ensure that the CF performs feasible multi-trip routes, (3.2b) resets the load every time a stop is visited, i.e., when a new trip starts.

To avoid enumerating all feasible paths, provable redundant labels are eliminated through a dominance procedure. Note that we can apply the following standard dominance rules, since all REFs are non-decreasing (Irnich, 2008): A label $\mathcal{L}_1 = (T_1^{rdc}, T_1^{load}, T_1^{time}, (T_1^{cust,k})_k)$ dominates another label $\mathcal{L}_2 = (T_2^{rdc}, T_2^{load}, T_2^{time}, (T_2^{cust,k})_k)$, if $T_1^{rdc} \leq T_2^{rdc}$, $T_1^{load} \leq T_2^{load}$, $T_1^{time} \leq T_2^{time}$, and $T_1^{cust,k} \leq T_2^{cust,k}$ holds for all $k \in N$. A dominated label can be discarded as long as at least one dominating label is kept.

To accelerate pricing, we implement several standard techniques, such as bidirectional labeling (Righini and Salani, 2006), *ng*-path relaxation (Baldacci *et al.*, 2011), and partial pricing (Gamache *et al.*, 1999). These techniques are briefly summarized in the following paragraphs:

Bidirectional Labeling Bidirectional labeling consists of propagating labels in forward and backward direction up to a half-way point. Afterwards, a merge procedure is applied to combine suitable forward and backward labels to feasible routes.

A backward label $\mathcal{L}'_j = (T_j'^{,rdc}, T_j'^{,load}, T_j'^{,time}, (T_j'^{cust,k})_{k\in N})$ represents a backward partial path $(j, \ldots, 0')$ in D starting at some vertex $j \in V$ and ending at the destination depot $0'$. Like in many other time-window constrained problems, the time attribute $T_j'^{,time}$ of a backward label represents the latest start of service at vertex j. The other attributes have the same meaning as for a forward label. The initial backward label for the trivial backward path is $\mathcal{L}'_{0'} = (T_{0'}'^{,rdc}, T_{0'}'^{,load}, T_{0'}'^{,time}, (T_{0'}'^{cust,k})_{k\in N}) = (0, 0, b_{0'}, \mathbf{0})$. An arbitrary backward label $\mathcal{L}'_j = (T_j'^{,rdc}, T_j'^{,load}, T_j'^{,time}, (T_j'^{cust,k})_{k\in N})$ is extended against the orientation of an arc $(i, j) \in A$ to create a new label $\mathcal{L}'_i = (T_i'^{,rdc}, T_i'^{,load}, T_i'^{,time}, (T_i'^{cust,k})_{k\in N})$ for the partial path $(i, j, \ldots, 0')$ with the help of the following REFs:

$$
\begin{aligned}
T_i'^{,rdc} &= T_j'^{,rdc} + \tilde{c}_{ij} \\
T_i'^{,load} &= \begin{cases} T_j'^{,load} + q_i, & \text{if } i \in N \\ 0, & \text{otherwise} \end{cases} \\
T_i'^{,time} &= \min\{b_i, T_j'^{,time} - t_{ij}\} \\
T_i'^{cust,k} &= \begin{cases} T_j'^{cust,k} + 1, & \text{if } i = k \in N \\ T_j'^{cust,k}, & \text{otherwise} \end{cases}, \qquad \forall k \in N
\end{aligned}
$$

The new backward partial path is feasible if the label fulfills $T_i'^{,time} \geq a_i$, $T_i'^{,load} \leq Q$, and $T_j'^{cust,k} \leq 1$ for all $k \in N$. Moreover, the following standard dominance rule can be applied: A label $\mathcal{L}_1' = (T_1'^{,rdc}, T_1'^{,load}, T_1'^{,time}, (T_1'^{cust,k})_k)$ dominates another label $\mathcal{L}_2' = (T_2'^{,rdc}, T_2'^{,load}, T_2'^{,time}, (T_2'^{cust,k})_k)$ if $T_1'^{,rdc} \leq T_2'^{,rdc}$, $T_1'^{,load} \leq T_2'^{,load}$, $T_1'^{,time} \geq T_2'^{,time}$, and $T_1'^{cust,k} \leq T_2'^{cust,k}$ holds for all $k \in N$.

Our bidirectional labeling algorithm uses the time resource as critical resource. Forward (backward) labels are only extended if their time resource T^{time} does not exceed ($T'^{,time}$ is greater than) the chosen half-way point HWP. Bidirectional labeling is only effective if less labels are generated and processed than in the monodirectional case. Because the number of labels in forward and backward labeling can be unbalanced for an a-priori chosen half-way point, we use a dynamic half-way point HWP as proposed in (Tilk *et al.*, 2017).

The merge procedure considers all vertices $i \in V$ and pairs of a forward label $\mathcal{L}_i = (T_i^{rdc}, T_i^{load}, T_i^{time}, (T_i^{cust,k})_k)$ and a backward label $\mathcal{L}_i' = (T_i'^{,rdc}, T_i'^{,load}, T_i'^{,time}, (T_i'^{cust,k})_k)$ resident at i. To avoid generating the same route multiple times, the forward label $\mathcal{L}_i$ must fulfill $T_i^{time} > HWP$ or $i = 0'$. The labels $\mathcal{L}_i$ and $\mathcal{L}_i'$ can be merged, if $T_i^{load} + T_i'^{,load} - q_i \leq Q$, $T_i^{time} \leq T_i'^{,time}$, and $T_i^{cust,k} + T_i'^{cust,k} \leq 1$ for all $k \in N \setminus \{i\}$. The reduced cost of the resulting route r is $\tilde{c}_r = T_i^{rdc} + T_i'^{,rdc}$. Only routes with $\tilde{c}_r < 0$ are solutions to the pricing problem.

***ng*-Path Relaxation** The *ng*-path relaxation (Baldacci *et al.*, 2011) of the elementary SPPRC is a parameterized relaxation defined by neighborhoods $N_i \subset N$ for each vertex $i \in V$. A cycle $(i, i_1, \ldots, i_\ell, i)$ with $\ell \geq 1$ is feasible if $i \notin N_{i_k}$ for some $k \in \{1, 2, \ldots, \ell\}$. By varying the sizes of the neighborhoods N_i one can control the trade-off between the difficulty of the resulting pricing problem and the strength of the LP-relaxation of the master program. We use neighborhoods N_i that contain vertex i (if $i \in N$) and the next 10 closest customer vertices.

Partial Pricing with Reduced Networks Partial pricing (Gamache *et al.*, 1999) describes that (a hierarchy of) heuristics can be used to solve the pricing problem as long as they deliver some negative reduced-cost variables. Partial pricing often helps to accelerate the overall column-generation process. We solve the pricing problem heuristically over reduced networks with up to $\sigma = 5, 10$, and 15 ingoing and outgoing arcs per vertex. Only if all heuristics fail, the pricing problem is solved exactly. Arcs are chosen according to the lowest reduced cost in the current pricing iteration.

3.4.2 Valid Inequalities

To strengthen the linear relaxation of the master program, Jepsen *et al.* (2008) introduced subset-row inequalities. To reduce the computational burden of solving the resulting pricing problems while keeping the strength comparable, *limited memory subset-row inequalities* (LmSRIs, Pecin *et al.*, 2017a) can be used instead. We restrict ourselves to subsets of three customers as proposed by Jepsen *et al.* (2008). Let $S \subset N$ be a subset of customers with $|S| = 3$. For any route $r \in \Omega$ and any non-negative integer number $h_r^S \leq \sum_{i \in S} p_{ir}$ (h_r^S is a lower bound on the number of times that route r serves a customer in S) , the corresponding LmSRI is given by

$$\sum_{r \in \Omega} \left\lfloor \frac{h_r^S}{2} \right\rfloor \lambda_r \leq 1. \tag{3.3}$$

For a solution of the linear relaxation of the master program, we use the same LmSRI separation algorithm as described by Pecin *et al.*, providing subsets S and a violated associated LmSRI if the coefficients are chosen as $h_r^S = \sum_{i \in S} p_{ir}$. Pecin *et al.* suggest to define a subset S-specific vertex memory that ensures $h_r^S = \sum_{i \in S} p_{ir}$ for all routes r that are part of the current solution, i.e., with $\lambda_r^* > 0$ (we distinguish between variables λ_r and their values λ_r^*). The role of the memory is very similar to the neighborhoods in the *ng*-path relaxation (see above). With a limited memory, the difficulty of the pricing subproblem is typically drastically reduced.

LmSRIs are non-robust cuts, i.e., they affect the structure of the SPPRC algorithm. Thus, for each SRI with a strictly negative dual value, we have to add a binary attribute to each label in the labeling algorithm. The dominance rules for forward and backward labeling have to be modified to effectively cope with the additional attributes (we refer to Jepsen *et al.*, 2008; Pecin *et al.*, 2017a, for further details).

During pretest we have identified the following reasonable parameters for the cutting strategy: The total number of LmSRIs to be added is limited to a maximum of 200 inequalities. Moreover, a maximum of 10 inequalities is added per round of separation.

3.4.3 Branching

Let λ_r^* be a fractional solution of the *restricted master program* (RMP). We use the following two-stage branching scheme to finally obtain integer solutions: We first branch on the total number of CFs, whenever $K^* = \sum_{r \in \Omega} \lambda_r^*$ is fractional. We create two branches $\sum_{r \in \Omega} \lambda_r \leq \lfloor K^* \rfloor$ and $\sum_{r \in \Omega} \lambda_r \geq \lceil K^* \rceil$.

The second stage branches on the fractional flows on arcs (i, j). To this end, we compute the value $f_{ij}^* = \sum_{r \in \Omega} f_{ij}^r \lambda_r^*$ for each arc $(i, j) \in A$, where f_{ij}^r denotes the

number of times that route r traverses arc $(i, j) \in A$. In a feasible solution, the flow value f^*_{ij} cannot exceed 1 if i or j is a customer vertex due to constraints (3.1b). The only arcs without a customer vertex are $(0, j)$ with $j \in H$, and they appear only once at the start of each route. Due to flow conservation, if all flow values f^*_{ij} for $(i, j) \in A \cap ((N \times V) \cup (V \times N))$ are binary, then all arc flows are integer. As a result, the decomposition of the integer flows into routes is unique. In conclusion, the branching scheme is complete and branching on arc flows can be implemented as standard binary branching on arcs that contain at least one customer.

The binary arc flow branching decisions can be enforced directly on the underlying network by removing arcs: The zero-branch is implemented by eliminating (i, j) from the arc set A, while for the one-branch all ingoing arcs of j and all outgoing arcs of i are eliminated except for the selected arc (i, j). Routes that are incompatible with the modified arc sets are temporarily removed from the RMP.

As an arc selection rule, we apply strong branching with up to eight candidates (Achterberg, 2007). For each candidate arc (i, j), a heuristic evaluation of the lower bounds of its child nodes is performed by using only the first pricing heuristic. We then choose the arc (i, j) that maximizes the minimum of the (estimated) lower bounds of its children. Branch-and-bound nodes are processed in increasing order of the RMP solution value of the parent node (best-first search).

When considering Objectives O3 and O4, in which the number of trips are minimized first, we add an additional branching-stage at the top of the hierarchy: We branch on the number of trips first, i.e., the visits of CFs to a stop vertex. Whenever the number of trips $U^* = \sum_{r \in \Omega} \sum_{h \in H} p_{hr} \lambda^*_r$ is fractional, we branch on it by creating the two branches $\sum_{r \in \Omega} \sum_{h \in H} p_{hr} \lambda_r \leq \lfloor U^* \rfloor$ and $\sum_{r \in \Omega} \sum_{h \in H} p_{hr} \lambda_r \geq \lceil U^* \rceil$.

3.5 Computational Results

The BPC algorithm presented in Section 3.4 has been implemented in `C++` and compiled in release mode into a 64-bit single-thread code under MS Visual Studio Enterprise 2015. The callable library CPLEX 20.1 is used for (re)optimizing the RMPs and as a primal MIP-based heuristic. The time limit for the BPC algorithm is 7200 seconds per instance. CPLEX's default values are kept for all parameters except from setting the number of threads to one. All computations were performed on a standard PC with MS Windows 10 running on an Intel® Core™ i7-5930K CPU clocked at 3.5 GHz and with 64 GB RAM.

In the following, we introduce the benchmark instances (Section 3.5.1), present and discuss algorithmic results (Section 3.5.2), analyze the impact of the different conflicting Objectives O1 to O4, and the capacity constraints for subsets of stops (Section 3.5.3).

3.5.1 Instances

As no commonly used set of instances is publicly available, we generate a large set of LMDPSL instances by combining four *scenarios* for the public transport network with 120 different *sets of customers*.

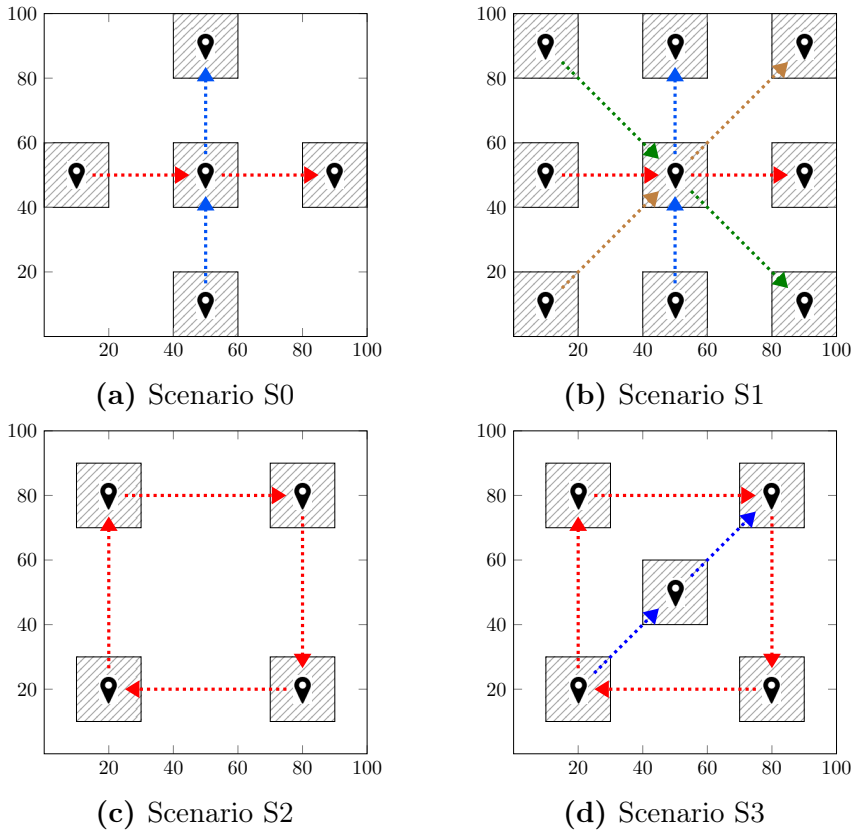
Public Transport Network For the public transport network, we consider a 100×100 grid, on which all physical locations are placed, and a planning horizon of 600 time units (minutes). The four scenarios are (see Figure 3.2):

- S0: two scheduled lines cross the same central station stopping at three stations each;
- S1: this scenario extends S0 by adding two more scheduled lines crossing the same central station and stopping at three stations each;
- S2: a single scheduled line operating in a circle around the city center stopping at four stations and
- S3: this scenario extends S2 by adding a single line crossing the central station stopping at three stations.

For each instance, the central station is placed randomly in $[40, 60] \times [40, 60]$. Similarly, the other stations are placed randomly in 20×20 areas as indicated by the shaded rectangles in Figure 3.2.

Some remarks about the setting are due:

- The detailed routing between two consecutive stations along a line is irrelevant. Therefore, the arcs in Figure 3.2 are just used to indicate the direction in which the public transport vehicle(s) travel when used within the 2-echelon distribution system.
- We further assume that deliveries to stations are always performed at the first possible instance. For example, in Scenario S0 (see Figure 3.2(a)) the public transport vehicle travels from a point in the west of the city where goods are loaded (not depicted, typically even further west than the first station) to the first station, the central station, the station in the east, and likewise (further east) to the endpoint of the red line. After a possible driver break, the same public transport vehicle travels back to the starting point finishing a single circulation. On the way back, no deliveries are made. We think that this assumption is realistic because otherwise transport boxes would have to be marked not only with the station where to unload them but with a time stamp. This bears the risk of confusion and is therefore operationally undesirable.
- In Scenarios S2 and S3, we assume that public transport vehicle(s) relevant for the 2-echelon distribution system travel in only one direction (here clockwise). Trams and buses are loaded at a station that is not necessarily one of the four depicted stations. The ring-line may or may not serve passengers in



(a) Scenario S0 **(b)** Scenario S1

(c) Scenario S2 **(d)** Scenario S3

Figure 3.2: Generation scheme of scenarios of the public transport network.

both directions.

- The modeling and solution approach also works for lines with more than three or four relevant stations per line.

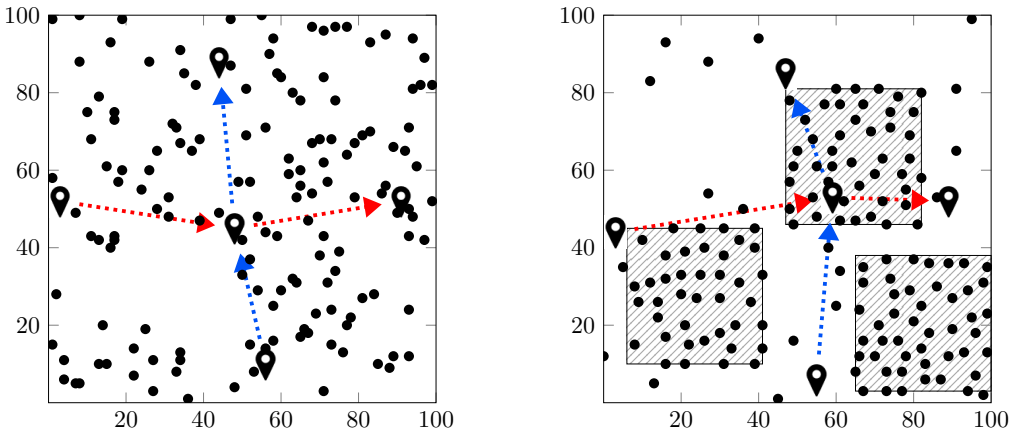
Customer Locations Customer locations are generated with two different methods leading to two subsets of instances. In the first subset, customer locations are randomly distributed over the 100×100 grid. Since it is well-known, that customer locations are not equally distributed among urban areas, we also create a second customer set. In the second subset, we generate three non-overlapping 35×35 areas for *clusters* denoted as W. Each customer location is randomly placed in one of these clusters W_1, W_2, and W_3, or on a random position. The probability p to be chosen in a cluster is $p_{W_1} = p_{W_2} = p_{W_3} = p_{\text{random}} = 0.25$. A randomly drawn customer location can also be part of a cluster. In addition, a minimum distance of 3 units must be kept between the customer locations within a cluster.

In both customer subsets, all customers $i \in N$ have a randomly drawn demand q_i that ranges between 1 and 5 packages. The handling time s_i at each customer

varies between 2 and 10 minutes and is independent from the demand. Moreover, a three- or five-hour time window is assigned to each customer. This time window is drawn from the following options $[0, 300]$, $[300, 600]$, $[60, 240]$, $[240, 420]$, and $[420, 600]$ with equal probability. Each time window reflects a preference for morning, noon, and afternoon deliveries.

The depot 0 is located in the center $(50, 50)$ of the grid. It hosts a homogeneous fleet of CFs, each with a capacity of one container which has room for $Q = 10$ packages. The number of CFs in the fleet is assumed unconstraining. Travel times and routing costs for the CFs are computed as Euclidean distances, rounded down to one decimal place. We generate 20 instances with 50, 100, and 150 customers, respectively.

Figure 3.3 visualizes two instances with 150 customers in Scenario S0. Figure 3.3a shows an instance of the first customer subset, where customer locations are randomly distributed, while Figure 3.3b visualizes an instance of the second customer subset with clustered customer locations.



(a) Random customer locations. **(b)** Clustered customer locations.

Figure 3.3: Instances with 150 customers for Scenario S0.

Table 3.2 shows the average number $|H|$ of stops and number U of circulation depending on the scenario and number of customers. These values result from the following assumptions: For the 50-customer instances, a single vehicle per scheduled line visits the corresponding stations in several circulations. For instances with 100 (150) customers, we increase the frequency by extending the fleet to 2 (3) vehicles.

Depending on the objective, we set the values for α, β, and γ as follows: O1: $\alpha = 1, \beta = 100000, \gamma = 0$; O2: $\alpha = 100, \beta = 1, \gamma = 0$; O3: $\alpha = 1, \beta = 100000, \gamma = 10000000$; O4: $\alpha = 1, \beta = 0, \gamma = 10000$. By choosing $\alpha = 1$ in the objectives O3 and O4 the routing costs are considered in all objective functions.

	$\|N\| = 50$		$\|N\| = 100$		$\|N\| = 150$	
Scenario	$\|H\|$	U	$\|H\|$	U	$\|H\|$	U
S0	22.1	7.4	39.6	13.2	57.8	19.3
S1	37.1	12.4	66.3	22.1	98.3	32.8
S2	8.2	2.1	16.0	4.0	24.0	6.0
S3	18.1	5.4	30.4	9.5	51.6	15.2

Table 3.2: Average number $|H|$ of stops and number U of circulations depending of scenario and size of the customer set.

The benchmark set comprises a total number of 240 instances and is available at `https://logistik.bwl.uni-mainz.de/research/benchmarks`.

3.5.2 Algorithmic Results

We now present algorithmic results for both instance sets on all four objectives introduced in Section 3.3. Note that the instances do not enforce specific capacities $C_{\bar{H}}$ for subsets $\bar{H} \in \mathcal{H}$ of stops (imposed by constraints (3.1c)). Since a reasonable value for the stop capacity of, e.g., a circulation, is instance-specific and hard to predict beforehand, we assume infinite stop capacities in this section. In Section 3.5.3 we investigate the impact of binding stop capacities in a separate study.

Exact Branch-Price-and-Cut Algorithm

The exact solution with the BPC algorithm uses a best-bound-first node selection strategy. It fosters the computation of tight lower bounds as fast as possible. In order to also provide an upper bound for cases when the BPC algorithm terminates without having identified a feasible solution within the given time limit (2 hours), the final RMP is solved as an integer model with the MIP solver. All generated columns until this point are included into the integer model. This MIP-based heuristic has a maximum runtime of 60 seconds so that the total runtime can never exceed 7260 seconds.

Tables 3.3 and 3.4 show aggregated results of the BPC algorithm and all four Objectives O1 to O4 on instances with randomly distributed and clustered customer locations, respectively. Results are grouped by number of customers ($|N|$) and scenarios (Scenario S0 to S3). Since no 150-customer instances are solved to optimality, we only present results for 50 and 100 customers. The column entries have the following meaning:

#Opt: the number of instances solved to proven optimality within the time limit
#UB: the number of instances for which an upper bound was found
Gap: the difference between the final upper and lower bound regarding the primary objective, i.e.,

- $\mathbf{Gap_{\#CF}}$: the number of times the gap in the number of CFs, comparing upper bound and the lower bound, is zero (± 0) or exactly one ($+1$), two ($+2$), etc.
- $\mathbf{Gap_{\#trip}}$: likewise, for the number of trips, presented in the same way as $\text{Gap}_{\#\text{CF}}$
- $\mathbf{Gap_{costs}}$: is calculated as $100 \cdot (UB_{costs} - LB_{costs})/LB_{cost}$, i.e., the average gap in percent

Note: gaps are computed only for instances for which an upper bound was found
Time: the average computing time in seconds
#BB: the average number of branch-and-bound nodes solved

$\|N\|$	Scenario	Objective O1 #Opt	#UB	$\text{Gap}_{\#\text{CF}}$ ($\pm 0/1/2/>2$)	Time	#BB	Objective O2 #Opt	#UB	Gap_{cost}	Time	#BB
50	S0	19	20	19/1/–/–	801.5	94.0	20	20	–	172.8	79.7
	S1	19	20	20/–/–/–	1,068.8	91.3	20	20	–	28.6	22.7
	S2	19	20	20/–/–/–	670.1	290.1	20	20	–	259.9	283.1
	S3	18	20	20/–/–/–	1,279.8	236.6	20	20	–	78.8	59.7
	Total/Avg.	**75**	**80**	**79/1/–/–**	**955.1**	**178.0**	**80**	**80**	**–**	**135.0**	**111.3**
100	S0	–	7	–/–/ 3/4	7,260.0	234.6	–	20	8.62	7,260.0	906.4
	S1	–	9	2/3/ 3/1	7,260.0	190.3	2	20	6.63	6,561.1	623.0
	S2	–	20	1/4/13/2	7,260.0	438.1	2	20	6.04	6,761.8	1,483.2
	S3	–	12	1/2/ 4/5	7,260.0	282.3	3	20	7.36	6,692.7	884.0
	Total/Avg.	**–**	**48**	**4/9/23/12**	**7,260.0**	**286.3**	**7**	**80**	**7.16**	**6,818.9**	**974.1**

$\|N\|$	Scenario	Objective O3 #Opt	#UB	$\text{Gap}_{\#\text{trip}}$ ($\pm 0/[1;5]/>5$)	Time	#BB	Objective O4 #Opt	#UB	$\text{Gap}_{\#\text{trip}}$ ($\pm 0/1/2/>2$)	Time	#BB
50	S0	18	20	19/1/–	2,153.6	334.9	20	20	20/–/–/–	228.9	143.9
	S1	15	20	18/2/–	2,340.5	234.4	20	20	20/–/–/–	538.5	225.3
	S2	20	20	20/–/–	575.7	276.2	20	20	20/–/–/–	61.4	59.9
	S3	15	20	20/–/–	2,709.4	1,005.9	20	20	20/–/–/–	270.2	142.7
	Total/Avg.	**68**	**80**	**77/3/–**	**1,944.8**	**462.8**	**80**	**80**	**80/–/–/–**	**274.75**	**142.9**
100	S0	–	12	–/4/ 8	7,260.0	207.7	–	20	2/3/6/9	7,260.0	773.2
	S1	–	7	–/3/ 4	7,260.0	139.5	–	20	2/6/5/7	7,260.0	549.8
	S2	–	19	–/7/12	7,260.0	370.6	2	20	4/9/4/3	6,617.8	1,087.7
	S3	–	14	–/2/12	7,260.0	192.1	–	20	–/8/9/3	7,260.0	787.4
	Total/Avg.	**–**	**52**	**–/16/36**	**7,260.0**	**227.4**	**2**	**80**	**8/26/24/22**	**7,099.5**	**799.5**

Table 3.3: Exact BPC algorithm: results for all Objectives O1 to O4 on randomly distributed customer instances.

$\|N\|$	Scenario	Objective O1 #Opt	#UB	$\text{Gap}_{\#\text{CF}}$ $(\pm 0/1/2/>2)$	Time	#BB	Objective O2 #Opt	#UB	Gap_{cost}	Time	#BB
50	S0	14	20	17/3/–/–	2,982.1	364.2	17	20	0.06	1,425.9	642.3
	S1	19	20	20/–/–/–	867.0	94.6	20	20	–	399.5	142.8
	S2	13	20	18/2/–/–	2,769.7	1,081.4	20	20	–	308.2	256.4
	S3	17	20	19/1/–/–	2,696.2	457.5	20	20	–	178.5	100.1
	Total/Avg.	**63**	**80**	**74/6/–/–**	**2,328.8**	**499.4**	**77**	**80**	**0.01**	**578.0**	**285.4**
100	S0	–	12	–/2/9/1	7,260.0	282.0	–	20	12.48	7,260.0	730.1
	S1	–	8	–/–/4/4	7,260.0	168.3	1	19	10.45	6,920.0	601.4
	S2	1	19	1/7/10/1	7,042.2	367.4	2	20	6.64	6,623.3	1,128.0
	S3	–	11	–/–/4/7	7,260.0	213.7	1	20	10.86	7,123.1	858.5
	Total/Avg.	**1**	**50**	**1/9/27/13**	**7,205.6**	**257.8**	**4**	**79**	**10.11**	**6,981.6**	**829.5**

$\|N\|$	Scenario	Objective O3 #Opt	#UB	$\text{Gap}_{\#\text{trip}}$ $(\pm 0/[1;5]/>5)$	Time	#BB	Objective O4 #Opt	#UB	$\text{Gap}_{\#\text{trip}}$ $(\pm 0/1/2/>2)$	Time	#BB
50	S0	12	20	16/4/–	3,704.8	611.8	19	20	20/–/–/–	1,484.38	760.6
	S1	11	20	17/3/–	3,789.0	552.5	20	20	20/–/–/–	501.85	171.0
	S2	19	20	20/–/–	1,356.3	717.1	19	20	20/–/–/–	197.15	159.3
	S3	12	20	17/3/–	3,672.9	1,322.9	19	20	20/–/–/–	588.89	226.3
	Total/Avg.	**54**	**80**	**70/10/–**	**3,130.8**	**801.0**	**77**	**80**	**80/–/–/–**	**693.1**	**329.3**
100	S0	–	13	–/3/10	7,260.0	258.6	1	19	2/3/4/10	6,966.0	661.2
	S1	–	7	–/2/5	7,260.0	125.6	–	20	1/2/7/10	7,260.0	536.9
	S2	–	20	–/11/9	7,260.0	374.4	–	20	–/5/8/7	7,260.0	1,171.3
	S3	–	13	–/4/9	7,260.0	188.7	–	19	–/1/6/12	7,260.0	736.9
	Total/Avg.	–	**53**	**–/20/33**	**7,260.0**	**236.8**	**1**	**78**	**3/11/25/39**	**7,186.5**	**776.5**

Table 3.4: Exact BPC algorithm: results for all Objectives O1 to O4 on clustered customer instances.

Both tables clearly show that the results strongly depend on the objective. We first discuss 50-customer results on the first instance set (see Table 3.3): For Objectives O2 and O4, all instances are solved exactly, while only 75 and 68 of 80 instances are solved for Objectives O1 and O3. Likewise, average computation times strongly differ, with approximately 3 and 5 minutes for Objectives O2 and O4, but approximately 16 and 32 minutes for Objectives O1 and O3, respectively. Comparing different scenarios, the average computation times for Objectives O1 and O2 (both minimizing the routing cost and the number of CFs) are diametrically opposed: While for Objective O1 instances of Scenarios S0 and S2 are solved faster, for Objective O2 instances of Scenarios S1 and S3 are solved faster than the others. For the Objectives O3 and O4 (both minimizing the number of trips first), Scenario S2 is solved fastest. For the 50-customer instances, all gaps regarding the primary objective are zero except for one instance with Objective O1 and three instances with Objective O3.

For instances with 100 randomly distributed customers, the BPC algorithm is able to solve a few instances to proven optimality, but only for Objectives O2 and O4. Moreover, the complete set of 80 upper bounds is also only computed for Objectives O2 and O4. The analysis of the gaps provides some insights into the practical difficulty of the four objectives: In the cases in which the primary objective is a natural number (the number of routes O1, the number of trips O3 and O4), the gaps tend to increase from Objective O4, over O1, to O3. In particular, rather large gaps can be observed for Objective O3 in combination with Scenarios S0 and S1. In many cases, the BPC algorithm is not able to provide an upper bound. Finally, there is a considerable increase in the number of branch-and-bound nodes when comparing Objectives O1 with O2 and O4 with O3.

Summarizing the results for the first subset, 50-customer instances can be solved easily in relatively short time, but instances with 100 and more customers still constitute a challenge. For these instances, the minimization of the number of CFs turns out to be much more difficult for the BPC algorithm than the minimization of the routing cost. This is rather counterintuitive, especially, that Objective O3 makes the solution so much harder than Objective O4 (recall that both minimize the number of trips first). Since every feasible solution with Objective O4 constitutes a feasible solution for Objective O3, one can take the result obtained for Objective O4 in order to minimize the number of trips.

Table 3.4 shows the results for the instance set with clustered customer locations. Instances with clustered customer locations are more difficult to solve. For 50-customer instances, the average computation time and the number of branch-and-bound nodes increases considerably on all four objectives and each scenario. The only exception is Scenario S1 on Objective O1 and O4, in which the computation times and the number of branch-and-bound nodes are slightly lower com-

pared to the first instance set. Consequently, the number of instances solved to proven optimality decreases and the corresponding gaps increase on average. For 50-customer instances, optimal solutions are only found for 63, 77, and 54 of 80 instances for Objectives O1, O2/O4, and O3, respectively. For 100-customer instances, the solution quality differs between the objectives. While the solution quality is similar on Objectives O1 and O3, it is worse on Objectives O2 and O4. Nevertheless, Objectives O2 and O4 are still easier to solve than Objectives O1 and O3 on the second instance set. While Objective O3 stays the most difficult, the BPC approach is fastest on Objectives O3 and O4 for Scenario S2.

Heuristic Branch-Price-and-Cut Algorithm

In response to the difficulty of providing good feasible solutions for instances with 100 and 150 customers and, in particular, for Objective O3, we design a heuristic version of the BPC algorithm. We omit results for 50-customer instances, since almost all average gaps were zero for the exact BPC. This heuristic version differs only slightly from the BPC algorithm presented in Section 3.4 in the following components.

On the larger instances, branching on arcs increases lower bounds only very moderately, leading to a large number of branch-and-bound nodes. Therefore, we skip branching on arc flows and, instead, use the MIP solver for the heuristic solution of each branch-and-bound node. The intention is to find promising feasible solutions as soon as possible. The time limit for each call of CPLEX is increased to a maximum of 1800 seconds leading to a total runtime limit of at most 9000 seconds. To emphasize finding high-quality solutions, we set CPLEX's parameter `CPX_PARAM_MIPEMPHASIS` to 5. Moreover, we increase the maximal number of LmSRIs from 200 to 300 and separate a maximum of 30 inequalities per round. Tables 3.5 and 3.6 summarize the results obtained with the heuristic BPC algorithm by presenting average gaps on both instance sets. As before, averages are taken over different combinations of objective and scenario.

For randomly distributed customer instances, we can directly compare the results for the 100-customer provided in Tables 3.3 and 3.5: The heuristic BPC algorithm provides a feasible solution in all cases. Moreover, the gaps decrease considerably for all four objectives. In particular, for Objective O1, the average gap reduces from two CFs to less than one. For Objective O2, the routing cost gap decreases from 7.1% to 4.6% on average, and for Objective O4, the average gap in the number of trips decreases from two to one. As expected, the heuristic BPC algorithm does not solve additional instances to optimality. Table 3.5 also shows that Objective O3 is still much harder to handle with the BPC algorithm than Objective O4. This is remarkable because both objectives minimize the number of trips first. For instances with clustered customer locations, we can compare the

$\|N\|$	Scenario	Objective O1 $\text{gap}_{\#\text{CF}}$ $(\pm 0/1/2/>2)$	Objective O2 gap_{cost}	Objective O3 $\text{gap}_{\#\text{trip}}$ $(\pm 0/[1;5]/>5)$	Objective O4 $\text{gap}_{\#\text{trip}}$ $(\pm 0/1/2/>2)$
100	S0	3/17/–/–	5.65	–/18/2	4/10/6/–
	S1	3/17/–/–	4.24	–/16/4	2/13/4/1
	S2	5/15/–/–	3.81	–/20/–	8/11/1/–
	S3	8/12/–/–	4.67	–/19/1	5/11/4/–
	Total/Avg.	**19/61/–/–**	**4.60**	**–/73/7**	**19/45/15/1**
150	S0	–/3/16/1	9.49	–/–/20	–/–/–/20
	S1	–/5/15/–	9.17	–/–/20	–/–/1/19
	S2	–/3/17/–	7.21	–/–/20	–/–/3/17
	S3	–/4/16/–	8.75	–/–/20	–/–/–/20
	Total/Avg.	**–/15/64/1**	**8.65**	**–/–/80**	**–/–/4/76**

Table 3.5: Heuristic BPC algorithm: gaps for Objectives O1 to O4 on randomly distributed customer instances.

$\|N\|$	Scenario	Objective O1 $\text{gap}_{\#\text{CF}}$ $(\pm 0/1/2/>2)$	Objective O2 gap_{cost}	Objective O3 $\text{gap}_{\#\text{trip}}$ $(\pm 0/[1;5]/>5)$	Objective O4 $\text{gap}_{\#\text{trip}}$ $(\pm 0/1/2/>2)$
100	S0	1/18/1/–	5.63	–/17/3	1/10/8/1
	S1	4/16/–/–	5.25	–/18/2	2/ 4/13/1
	S2	5/15/–/–	3.49	–/20/–	4/13/3/–
	S3	3/17/–/–	5.42	–/18/2	–/12/7/1
	Total/Avg.	**13/66/1/–**	**4.95**	**–/73/7**	**7/39/31/3**
150	S0	–/8/11/1	10.3	–/–/20	–/–/–/20
	S1	–/8/12/–	11.02	–/–/20	–/–/–/20
	S2	–/5/15/–	7.64	–/–/20	–/1/1/18
	S3	–/4/15/1	9.67	–/–/20	–/–/1/19
	Total/Avg.	**–/25/53/2**	**9.66**	**–/–/80**	**–/1/2/77**

Table 3.6: Heuristic BPC algorithm: gaps for Objectives O1 to O4 on clustered customer instances.

results for 100-customer in Tables 3.4 and 3.6. Basically, the results behave similar in comparison to the first instance set. Also here, the heuristic BPC algorithm provides a feasible solution in all cases but does not solve additional instances to optimality. Further, the gaps can be reduced for all four objectives. Especially for Objective O2 the routing cost gap decreases from 10.11% to 4.95% on average. A direct comparison between both instances sets (Tables 3.5 and 3.6) shows that the second instance set is still harder to solve, even for the heuristic branch-price-and-cut algorithm. For all 100-customer instances the gaps are slightly higher for Objectives O1, O2, and O4 on the second instance set. The solution quality for Objective O3 does not change. The gaps on 150-customer instances are remarkable. For Objective O1, 25 instances were solved with a gap of one CF on the second instance set, while just 15 instances were solved with an gap of one CF on the first instance set. Especially Scenarios S0 and S1 seem to perform much better on this kind of instances.

3.5.3 Managerial Insights

We now present two analyses on (i) the impact of the objective on the key indicators (number of CFs, routing cost, and number of trips), and (ii) the impact of additional capacity constraints for the public transport vehicle.

Impact of the Objectives

All objectives that we consider minimize two of the three key indicators, i.e., number of CFs, routing cost, or number of trips, with different prioritization. Since the previous section has shown that the results for Objective O3 are less reliable due to larger gaps compared to those of the other objectives, we omit them here. We present aggregated results grouped by number of customers ($|N|$) and the different public transport scenarios (Scenario S0 to S3) in Tables 3.7 and 3.8. The tables show the impact of the objective on the key indicators for randomly distributed and clustered customer locations, respectively. Instance size $|N|$ ranges from 50 to 150 customers, always taking the lexicographically best solution found during the exact or heuristic experiments. The columns of both tables have the following meaning:

costs: the average routing costs
#CFs: the average number of CFs
#trips: the average number of trips

These value are adjusted disregarding the multipliers α, β, and γ.

Comparison of Scenarios Table 3.7 shows that across all objectives and independently from the number of customers, Scenario S1 always has the lowest

routing cost. This can be attributed to the fact that Scenario S1 has the highest number of lines and stops. In addition, the stops are evenly distributed throughout the city resulting in short traveling distances for replenishing the CFs.

In contrast, Scenario S2 always has the highest routing cost and number of CFs, which has mainly two reasons: Scenario S2 has the lowest number of stops, and it is the only scenario without a stop at the central station, resulting in long distances for replenishing the CFs. Moreover, Scenario S2 consistently results in the lowest number of trips, while for all other scenarios, the numbers are almost identical. Table 3.8 shows that these findings are also true for the instance set with clustered customer locations.

Additionally, it shows that on instances with 50 customers, the routing costs are slightly lower compared to the first subset of instances, where customer locations are randomly distributed. On 100- and 150-customer instances, the routing costs increase. This can be due to the fact that instances with 100 and 150 customers are solved heuristically, while 50-customer instances are solved exactly. The differences in the number of used CFs and the number of trips are relatively small compared to the instance set with randomly located customers.

Figure 3.4 further aggregates the results of Tables 3.7 and 3.8 over different instance sizes, visualizing only the respective primary objective. With Objective O4 prioritizing the minimization of the number of trips, the differences are hardly visible (see Figure 3.4c). We conclude that minimizing the number of trips is close to a pure packing problem, and timing aspects are almost negligible for routing a minimum number of CFs.

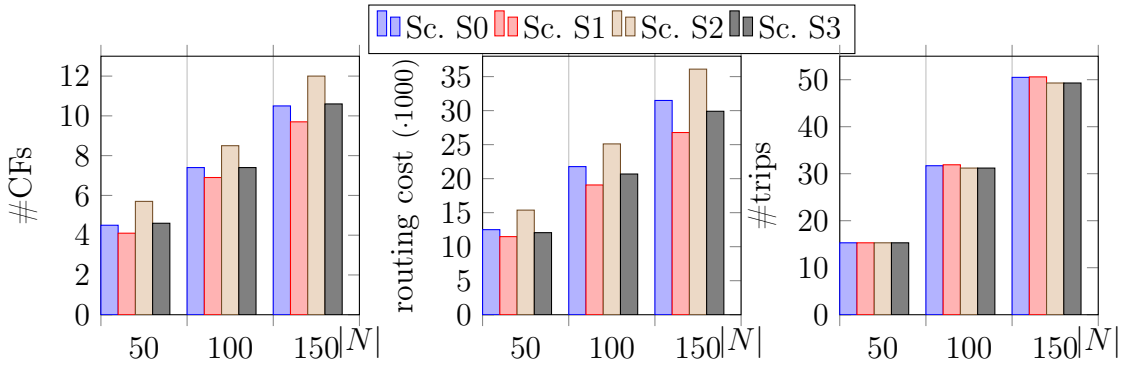
		Objective O1			Objective O2			Objective O4		
		(second.)	(prim.)		(prim.)	(second.)		(second.)		(prim.)
$\|N\|$	Scenario	costs	#CFs	#trips	costs	#CFs	#trips	costs	#CFs	#trips
50	S0	14,038	4.5	16.9	12,743	6.6	16.7	12,990	6.6	15.3
	S1	13,051	4.2	17.1	11,737	6.4	17.1	12,200	6.5	15.3
	S2	17,505	5.7	15.5	15,737	8.1	15.7	15,800	8.1	15.3
	S3	13,740	4.7	16.4	12,448	7.1	16.7	12,719	7.1	15.3
	Avg.	**14,583**	**4.7**	**16.5**	**13,166**	**7.0**	**16.5**	**13,427**	**7.0**	**15.3**
100	S0	25,485	7.3	37.4	21,562	10.3	34.2	22,533	11.5	31.4
	S1	22,147	7.0	37.4	18,682	9.7	34.2	19,978	11.0	31.5
	S2	29,990	8.5	35.7	25,051	11.8	33.4	26,028	12.6	30.9
	S3	24,869	7.3	37.6	20,414	10.6	34.3	21,334	11.7	31.3
	Avg.	**25,623**	**7.5**	**37.0**	**21,427**	**10.6**	**34.0**	**22,468**	**11.7**	**31.3**
150	S0	37,343	10.8	60.5	31,695	14.0	53.3	32,809	15.3	49.9
	S1	32,443	9.7	61.4	26,192	12.9	54.3	27,565	14.5	49.7
	S2	44,000	12.0	58.3	35,770	15.7	52.0	37,034	16.6	49.1
	S3	36,175	10.5	60.3	29,581	14.1	53.4	30,614	15.1	49.9
	Avg.	**37,490**	**10.7**	**60.1**	**30,809**	**14.1**	**53.3**	**32,006**	**15.4**	**49.6**

Table 3.7: Impact of objectives O1, O2, and O4 on key indicators for randomly distributed customer locations.

		Objective O1			Objective O2			Objective O4		
		(second.)	(prim.)		(prim.)	(second.)		(second.)		(prim.)
$\|N\|$	Scenario	costs	#CFs	#trips	costs	#CFs	#trips	costs	#CFs	#trips
50	S0	14,000	4.6	16.6	12,265	6.6	16.8	12,429	6.6	15.4
	S1	12,523	4.1	16.8	11,224	6.4	16.9	11,491	6.6	15.4
	S2	17,397	5.7	15.7	15,016	7.9	15.7	15,069	7.9	15.4
	S3	13,118	4.5	16.4	11,678	6.8	16.5	11,862	6.8	15.4
	Avg.	**14,259**	**4.7**	**16.4**	**12,546**	**6.9**	**16.5**	**12,712**	**7.0**	**15.4**
100	S0	25,613	7.5	37.7	21,981	10.0	34.2	22,535	10.7	31.9
	S1	22,870	6.9	37.9	19,464	9.6	34.8	20,271	9.9	32.2
	S2	29,806	8.6	35.2	25,168	11.4	33.6	26,023	12.3	31.5
	S3	25,242	7.4	38.1	20,951	10.0	34.5	21,707	10.8	32.1
	Avg.	**25,883**	**7.6**	**37.2**	**21,891**	**10.2**	**34.3**	**22,634**	**10.9**	**31.9**
150	S0	36,789	10.3	60.9	31,296	13.4	54.5	32,375	14.3	51.2
	S1	33,893	9.7	63.0	27,372	13.1	55.4	28,476	13.6	51.5
	S2	46,035	11.9	59.2	36,429	15.6	51.9	37,450	16.4	49.6
	S3	35,770	10.6	59.9	30,232	14.1	53.2	30,911	14.8	50.5
	Avg.	**38,122**	**10.6**	**60.7**	**31,332**	**14.0**	**53.7**	**32,303**	**14.8**	**50.7**

Table 3.8: Impact of objectives O1, O2, and O4 on key indicators for clustered customer locations.

Note that Scenario S0 is included in Scenario S1, where the latter scenario roughly doubles the number of stations and stops. In comparison, the average number of CFs (Objective O1, see Figure 3.4a) decreases by not more than one, the strongest for the 150-customer instances. At the same time, we notice a rather moderate average decrease in routing cost (Objective O2, see Figure 3.4b) of between 8% and 15% depending on the number of customers.



(a) Number of CFs (primary objective) using Objective O1

(b) Routing cost (primary objective) using Objective O2

(c) Number of trips (primary objective) using Objective O4

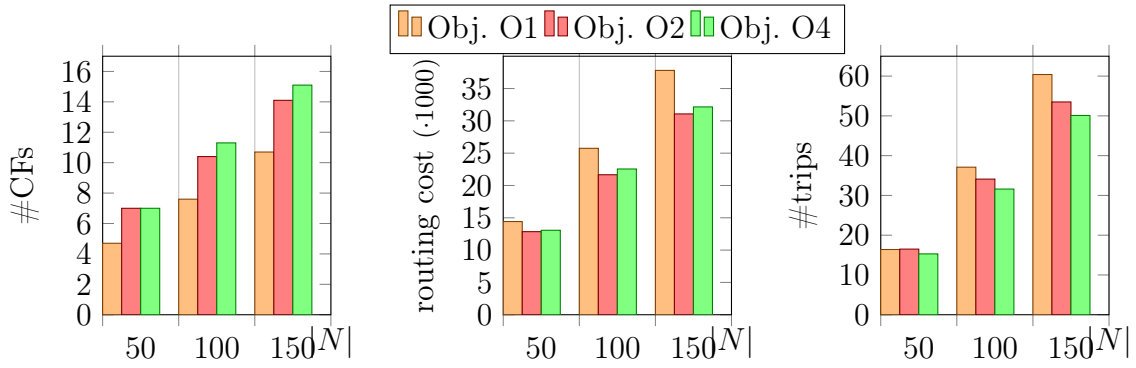
Figure 3.4: Comparison of Scenarios S0 to S3 with respect to the respective primary objective.

Likewise, Scenario S2 is included in Scenario S3, where also the latter scenario roughly doubles the number of stations and stops. On average, this leads to larger decreases in routing cost (by around 20%) and number of CFs (also around 20%), which is a more considerable decrease than in the comparison of Scenarios S0 and S1.

Finally, we compare Scenarios S0 and S3, since they feature a comparable number of stops (see also Table 3.2). While the number of CFs and trips is nearly identical, Scenario S3 gives slightly smaller routing costs (between 3.5% and 3.5%) than Scenario S0. We attribute this to the better distribution of the stops in Scenario S3; the four stops in the outskirts of the map are slightly closer to the center.

Comparison of Objectives Finally, we compare the impact of the three Objectives O1, O2, and O4 on the three key indicators (number of CFs, routing cost, and the number of trips). Recall that two of them are the primary or secondary objectives, while the third is not minimized with the respective objective.

Figure 3.5 visualizes aggregated results using the best-found solution considering results obtained with different objectives.



(a) Number of CFs (primary objective in O1)

(b) Routing costs (primary objective in O2)

(c) Number of trips (primary objective in O4)

Figure 3.5: Comparison of Objectives O1, O2, and O4 with respect to all three key indicators

Note: Objective O1 minimizes the number of CFs first and routing costs second; Objective O2 minimizes the routing costs first and number of CFs second; Objective O4 minimizes the number of trips first and the routing costs second.

Comparing Objective O1 (minimizing the number of CFs first and the routing cost second) and Objective O2 (primary and secondary objective swapped), the tradeoff between the two becomes clear: Prioritizing the minimization of the number of CFs increases the average routing costs by around 10% (20%) for instances with 50 (100 and 150) customers. Conversely, prioritizing the minimization of the routing costs increases the number of CFs by around two (three) for 50-customer (100- and 150-customer) instances.

Comparing Objectives O2 and O4, with the latter objective the average number of trips can be reduced by approximately 1.2, 2.5, and 3.4 for instances with 50, 100, and 150 customers, respectively. In turn, the routing costs increase by 1.5% to 4%. The number of CFs takes the largest value with Objective O4, and the number of trips takes the largest value with Objective O1. In conclusion, Objectives O1 and O4 are strongly opposite.

Capacity Constraints for Circulations

We now investigate the impact of limiting capacity for circulations on routing costs (Objective O2), i.e., we upper bound the number of containers that can be transported simultaneously in a scheduled line. One could perform the same type of analysis also with the other objective functions. However, with the smaller objective values for the number of CFs and trips, plateau effects make results less

interpretable.

Instances with randomly distributed as well as clustered customer locations defined by all four Scenarios S0 to S3 are compared. Finding a truly constraining capacity is a non-trivial and instance-dependent task that we approach as follows: In a preparatory step, we individually determine for each 50- and 100-customer instance the lowest capacity value C^{min} that can be added for all circulations so that this instance remains feasibly solvable. To determine C^{min}, we start with the minimal possible value that we determine by considering the sum of customer demands. Then, we successively increase the capacity value until we can find a feasible solution. Note that all 50-customer instances are solved with the exact BPC algorithm, hence, the minimum capacity values C^{min} are exact. In contrast, for some 100-customer instances, the computed minimum capacity values C^{min} might be slightly larger than the minimum, because we apply the heuristic BPC algorithm. The range of the minimum capacity values C^{min} for each instance set and scenario is shown in Table 3.9.

Scenario	$\|N\| = 50$	$\|N\| = 100$
S0	$2-3$	$3-4$
S1	$1-2$	2
S2	$6-9$	$7-10$
S3	$3-4$	$3-5$

(a) Random customer locations.

Scenario	$\|N\| = 50$	$\|N\| = 100$
S0	$2-3$	$3-4$
S1	2	2
S2	$7-9$	$8-9$
S3	$3-4$	$4-5$

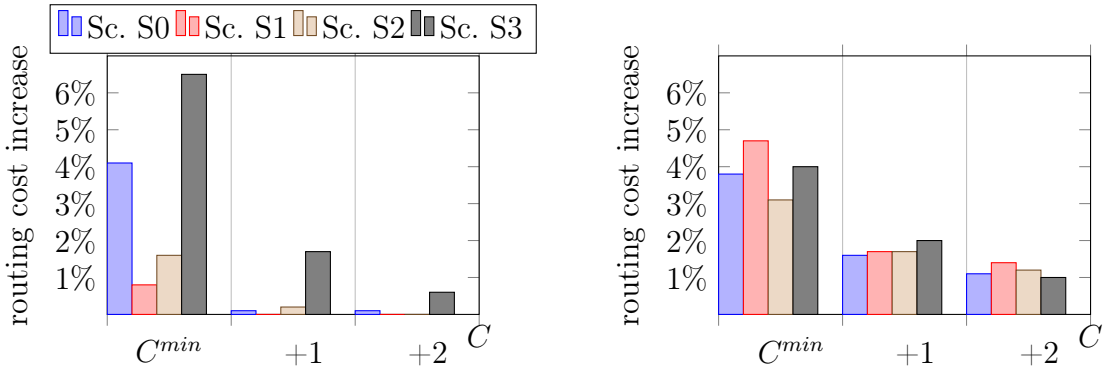
(b) Clustered customer locations.

Table 3.9: Minimum values of circulation capacities.

For analyzing the impact of different capacity values, we solve all instances with capacity values C^{min}, $C^{min}+1$, and $C^{min}+2$ (the latter two cases are abbreviated with "+1" and "+2"). Figure 3.6 visualizes the outcome showing the average increase in routing costs compared to the best-found solution with unlimited circulation capacity (see Section 3.5.2). For the 50-customer instances, Figure 3.6a clearly shows that the minimum circulation capacity is strongly restricting resulting in an average routing cost increase of up to 6.5% depending on the scenario. However, choosing a higher capacity (+1 or +2) leads to an almost negligible increase in routing costs with the exception of Scenario S3. It is noticeable that the exact solution for 50-customer instances with a capacity of C^{min} takes on average four times longer than for +1 and +2 (the latter times are comparable to the unrestricted case).

For the 100-customer instances, Figure 3.6b shows a travel cost increase of around 4% for the minimum circulation capacity C^{min}, by nearly 1.5% for +1, and more than 1% for +2. However, the 100-customer instances are solved heuris-

tically so that it remains unclear whether the exact solution values would give an identical result. Comparing different scenarios, differences are relatively minor, and there is no clear tendency. It seems that not the structure of the transport network but rather the ratio of the sum of customer demands and the overall circulation capacity is the main driver for increased routing costs. In summary, a realistic circulation capacity can be enforced for the analyzed instances leading to only a moderate increase in routing costs.



(a) 50-customer instances.

(b) 100-customer instances.

Figure 3.6: Routing cost increase in percent for capacitated circulations using Objective O2.

3.6 Conclusions

The paper has introduced the LMDPSL, a prototypical 2-echelon last-mile delivery problem that utilizes existing public transport services, and presented exact and heuristic BPC algorithms to tackle instances of LMDPSL with up to 150 customers. Scenarios reflecting the possible settings found in real-world public transport networks with different hierarchical objectives can be solved exactly or with a relatively small optimality gap with the proposed algorithms. The strong performance of the BPC algorithms did not only allow us to solve the operational planning problems (vehicle routing and scheduling) but also address various questions arising when looking at the system from a more tactical perspective (analyzing tradeoffs and the impact of different assumptions).

The studies reveal that scenarios with a high number of lines and stops (like Scenario S1) almost always have the lowest routing costs and require a smaller number of CFs regardless of the number of customers. Moreover, scenarios with scheduled lines operating in a circle around the city center (like Scenario S2) always lead to the highest routing costs and number of CFs, because a small number of stops

or a missing stop at a central station results in longer distances for replenishing the CFs. Our results also show that minimizing the number of trips is close to a pure packing problem, i.e., timing aspects are almost negligible. Under realistic assumptions, the capacity of public transport vehicles is limited. We, therefore, investigated the impact of capacity constraints on circulations and found that they result in only a moderate increase of the total routing costs for the analyzed instances.

A rather expected outcome confirmed by the different studies is that the distribution of customer locations has a significant impact on the difficulty of solving LMDPSL instances: The overall trend is that instances with randomly distributed customer locations are simpler to solve compared to instances with clustered customer locations. However, the different studies comparing scenarios, objectives, and capacity constraints for circulations did not yield substantial differences between distributions. The same managerial insights hold for both distributions of customer locations, which was a more surprising result.

The proposed shared last-mile delivery system offers several benefits, such as reduced inner-city traffic, fast and reliable deliveries, and cost savings. For a real-world implementation of the system, operators must clarify several legal and liability issues that arise when people and goods share the same public transport vehicle (Schocke *et al.*, 2020). Any negative effects on travelers must be avoided when goods and people are transported together during their journey (Mourad *et al.*, 2019), e.g., by selecting only lines and stations with a low passenger traffic for the goods transfer.

Acknowledgement

This research was supported by Deutsche Forschungsgemeinschaft (DFG) under grant IR 122/8-1. This support is gratefully acknowledged.

Bibliography

Achterberg, T. (2007). *Constraint Integer Programming*. Ph.D. thesis, Technische Universität Berlin, Fakultät II – Mathematik und Naturwissenschaften, Berlin, Germany.

Anderluh, A., Hemmelmayr, V. C., and Nolz, P. C. (2019). Sustainable logistics with cargo bikes-methods and applications. In J. Faulin, S. E. Grasman, A. A. Juan, and P. Hirsch, editors, *Sustainable Transportation and Smart Logistics*, chapter 8, pages 207–232. Elsevier.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Behiri, W., Belmokhtar-Berraf, S., and Chu, C. (2018). Urban freight transport using passenger rail network: Scientific issues and quantitative analysis. *Transportation Research Part E: Logistics and Transportation Review*, **115**, 227–245.

BIEK (2021). KEP-Studie 2021: Analyse des Marktes in Deutschland: Möglichmacher in bewegten Zeiten. Bundesverband Paket und Expresslogistik e. V. (BIEK) `https://www.biek.de/files/biek/downloads/papiere/BIEK_KEP-Studie_2021.pdf`, Berlin, Germany (in German).

Costa, L., Contardo, C., and Desaulniers, G. (2019). Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, **53**, 946–985.

Cuda, R., Guastaroba, G., and Speranza, M. G. (2015). A survey on two-echelon routing problems. *Computers & Operations Research*, **55**, 185–199.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Drexl, M. (2012). Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. *Transportation Science*, **46**(3), 297–316.

Elbert, R. and Rentschler, J. (2021). Freight on urban public transportation: A systematic literature review. *Research in Transportation Business & Management*, **45**(Part A), 100679.

Fatnassi, E., Chaouachi, J., and Klibi, W. (2015). Planning and operating a shared goods and passengers on-demand rapid transit system for sustainable city-logistics. *Transportation Research Part B: Methodological*, **81**, 440–460.

Furmanik, G. (2019). How much does retail space cost? *realla*. `https://blog.realla.co.uk/how-much-does-retail-space-cost`.

Gamache, M., Soumis, F., Marquis, G., and Desrosiers, J. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations Research*, **47**(2), 247–263.

Ghilas, V., Demir, E., and Woensel, T. V. (2016a). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. *Computers & Operations Research*, **72**, 12–30.

Ghilas, V., Demir, E., and Woensel, T. V. (2016b). The pickup and delivery problem with time windows and scheduled lines. *INFOR: Information Systems and Operational Research*, **54**(2), 147–167.

Ghilas, V., Demir, E., and Woensel, T. V. (2016c). A scenario-based planning for the pickup and delivery problem with time windows, scheduled lines and stochastic demands. *Transportation Research Part B: Methodological*, **91**, 34–51.

Ghilas, V., Cordeau, J.-F., Demir, E., and Woensel, T. V. (2018). Branch-and-price for the pickup and delivery problem with time windows and scheduled lines. *Transportation Science*, **52**(5), 1191–1210.

Irnich, S. (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**(2), 497–511.

Masson, R., Trentini, A., Lehuédé, F., Malhéné, N., Péton, O., and Tlahig, H. (2015). Optimization of a city logistics transportation system with mixed passengers and goods. *EURO Journal on Transportation and Logistics*, **6**(1), 81–109.

Mourad, A., Puchinger, J., and Chu, C. (2019). A survey of models and algorithms for optimizing shared mobility. *Transportation Research Part B: Methodological*, **123**, 323–346.

Mourad, A., Puchinger, J., and Woensel, T. V. (2020). Integrating autonomous delivery service into a passenger transportation system. *International Journal of Production Research*, **59**(7), 2116–2139.

OECD (2020). E-commerce in the time of COVID-19. `https://www.oecd.org/coronavirus/policy-responses/e-commerce-in-the-time-of-covid-19-3a2b78e8/`.

Ozturk, O. and Patrick, J. (2018). An optimization model for freight transport using urban rail transit. *European Journal of Operational Research*, **267**(3), 1110–1121.

Paradiso, R., Roberti, R., Laganá, D., and Dullaert, W. (2020). An exact solution framework for multitrip vehicle-routing problems with time windows. *Operations Research*, **68**(1), 180–198.

Pecin, D., Pessoa, A., Poggi, M., and Uchoa, E. (2017a). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, **9**, 61–100.

Pecin, D., Contardo, C., Desaulniers, G., and Uchoa, E. (2017b). New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal Computing*, **29**(3), 489–502.

Pessoa, A., Sadykov, R., Uchoa, E., and Vanderbeck, F. (2020). A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, **183**(1-2), 483–523.

Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, **3**(3), 255–273.

Savelsbergh, M. and van Woensel, T. (2016). 50th Anniversary invited article—City logistics: Challenges and opportunities. *Transportation Science*, **50**(2), 579–590.

Schiffer, M., Schneider, M., Walther, G., and Laporte, G. (2019). Vehicle routing and location routing with intermediate stops: A review. *Transportation Science*, **53**(2), 319–343.

Schneider, M., Stenger, A., and Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, **48**(4), 500–520.

Schocke, K.-O., Schäfer, P. K., Höhl, S., and Gilbert, A. (2020). *Last mile tram: Empirische Forschung zum Einsatz einer Güterstraßenbahn am Beispiel Frankfurt am Main*. Frankfurt University of Applied Sciences, Frankfurt am Main. (in German).

Tilk, C., Rothenbächer, A.-K., Gschwind, T., and Irnich, S. (2017). Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research*, **261**(2), 530–539.

United Nations (2021). COVID-19 and e-commerce: A global review. UNCTAD/DTL/STICT/2020/13 `https://unctad.org/system/files/official-document/dtlstict2020d13_en.pdf`.

Zhou, L., Baldacci, R., Vigo, D., and Wang, X. (2018). A multi-depot two-echelon vehicle routing problem with delivery options arising in the last mile distribution. *European Journal of Operational Research*, **265**(2), 765–778.

Chapter 4

Exact Solution of the Vehicle Routing Problem with Drones

Jeanette Schmidt, Christian Tilk, Stefan Irnich

Abstract

The vehicle routing problem with drones (VRP-D) that we consider is an extension of the capacitated vehicle routing problem, in which the fleet consists of trucks equipped with one drone each. A truck and its drone can either move together or separately. A truck can release its drone at the depot or at a customer location and must pick it up later at another customer or the depot location. working units. A feasible route has to satisfy the capacity constraints of both the truck and the drone. A feasible solution to the VRP-D is a set of feasible routes such that each customer is served exactly once by either a truck or a drone. We investigate two standard objectives considered in the literature, i.e., the minimization of the total routing cost and the sum of the routes' durations. To solve the VRP-D exactly, we develop a branch-price-and-cut (BPC) algorithm. In particular, we present a new forward and implicit bidirectional labeling algorithm defined over an artificial network to solve the column-generation subproblems. The new bidirectional labeling algorithm substantially accelerates the solution process compared to its monodirectional counterpart. The time needed to solve the pricing problems is reduced by 55% on average when minimizing routing costs and by 30% when minimizing the sum of the routes' durations. In further computational experiments, we analyze algorithmic components of the BPC algorithm, compare the cost and duration objectives, and highlight the impact of the drones' speed on the structure of VRP-D solutions. For the routing-cost minimization objective, our BPC algorithm is able to solve several VRP-D instances with 50 vertices to proven optimality within one hour of computation time. The same instances with duration minimization are more difficult, and the BPC algorithm provides only heuristic solutions with an average gap not exceeding three percent.

4.1 Introduction

Drones are an emerging technology. This explains why planning problems related to drones have attracted researchers from various disciplines, e.g., engineering, computer science, and control theory (Otto *et al.*, 2018). A *drone*, a.k.a. unmanned aerial vehicle, is an aircraft that does not have a pilot onboard and is controlled remotely (ITA, 2023). Compared to ground and water vehicles such as trucks and ships, modern drones are typically faster, can move in three dimensions to surmount obstacles on the ground, can take off and land autonomously, have a lower cost per kilometer to operate, and emit less CO_2 (Goodchild and Toy, 2018; D'Andrea, 2014). In this sense, the use of drones represents a greener and more versatile alternative to conventional delivery modes. However, they have the drawback of a limited payload and flight range (even if modern parcel delivery drones can fly up to several kilometers with a payload of about 5kg, more than 5% of parcels weigh between 15 and 30kg (Joerss *et al.*, 2016)). We consider combined *delivery operations* of trucks and drones, which can lead to a significant increase of efficiency and effectiveness in performing the delivery operations (Wang *et al.*, 2017; Carlsson and Song, 2018). According to Otto *et al.* (2018, Sect. 5), such combined truck-and-drone operations are of four types: (1) trucks support the delivery operations performed solely by drones, (2) drones support the delivery operations performed solely by trucks, (3) trucks and drones operate independently and perform separate delivery routes, and (4) trucks and drones work together as synchronized working units. Combined delivery operations make particularly sense when conventional vehicles cannot easily and quickly reach some intermediate or delivery points, e.g., for parcel delivery to remote households, medical supply delivery (vaccines, blood, drugs) and collection (samples of blood, urine, etc.), delivery of spare parts, etc. For the numerous existing and potential applications, we refer the reader to the surveys (Otto *et al.*, 2018; Chung *et al.*, 2020; Macrina *et al.*, 2020; Moshref-Javadi and Winkenbach, 2021; Madani and Ndiaye, 2022). We focus on the synchronized routing problem of type (4).

The variant of the *vehicle routing problem with drones* (VRP-D) that we consider has a fleet of trucks that is equipped with one drone each. A truck and its drone can either move together or separately. When truck and drone move together, the drone is either inside or on top of the truck. The drone can leave the truck at the depot or any customer location and perform a single delivery alone (this is realistic given the limited payload of the drone and for operational/safety reasons). Meanwhile, the truck continues its journey alone serving one or several other customers. In any case, the drone must return to the same truck at a customer or the depot location. The VRP-D consists of finding feasible combined truck-and-drone routes to serve a given set of customers. A feasible route must satisfy the following conditions: (a) it starts and ends at the given depot, (b) the

capacity of the truck is respected, and (c) each drone flight serves exactly one customer. Common objectives of the VRP-D are minimizing the total routing cost and the sum of the durations of all routes. The literature does not consider and discuss that these two objectives are competing. Hence, we will analyze the different structure of solutions obtained with each objective (see Section 4.5.4).

The need to synchronize the movements of trucks and drones makes the VRP-D more challenging compared to the classical *capacitated vehicle routing problem* (CVRP, Irnich *et al.*, 2014), because additional decisions about which type of vehicle serves which subset of customers must be made (called *task synchronization* in Drexl, 2012). The synchronization of a truck and its drone in space (*movement synchronization*) and, for duration minimization, synchronization in time (*operation synchronization*) must be considered.

We develop a VRP-D-tailored *branch-price-and-cut* (BPC) algorithm that allows the exact solution for both objectives, i.e., the minimization of the total routing cost and the sum of the durations. Note that in a multiple-vehicle context without time windows, the minimization of duration refers to the sum of the completion times of all routes where each route starts at time 0. It should not be confused with the minimization of the makespan, which is the minimization of the latest completion time and leads to a problem of min-max type.

The most important algorithmic component of the BPC algorithm is a bidirectional labeling algorithm to solve the column-generation subproblems. This subproblem is defined over an artificial network originally introduced by Roberti and Ruthmair (2021) for the single-vehicle version of this problem, i.e., for the *traveling salesman problem with drone* (TSP-D). While Roberti and Ruthmair use a non-trivial monodirectional labeling algorithm, we use a bidirectional labeling algorithm, which is responsible for the convincing performance of the overall BPC algorithm: We are able to tackle larger instances than solved for the TSP-D, which is opposed to the often observed relationship between the practical difficulty of single- and multiple-vehicle routing problems.

The artificial network is a multi-digraph, in which vertices represent possible combinations of locations where a truck and its associated drone perform their service. Arcs of the artificial network model possible truck-and-drone movements. A major complication in designing an effective bidirectional labeling algorithm is that the same route, once considered in forward and once in backward direction, passes through different vertices of the artificial network whenever the drone operates without the truck. In particular, the merge procedure is unclear in such a setting. Moreover, the subproblem is asymmetric even though the VRP-D is defined as a symmetric routing problem. Unexpectedly, the inherent symmetry of the VRP-D can be exploited: Instead of explicitly generating forward and backward labels, the same set of labels can be considered for both directions. This is

what we name *implicit* bidirectional labeling.

Accordingly, the main contributions of the paper at hand are as follows:

- We develop a new exact algorithm based on BPC to solve the VRP-D exactly for both objectives, minimization of routing cost and minimization of the sum of the routes' durations.
- We show that an implicit bidirectional labeling algorithm is essential for solving the pricing subproblems effectively. A particular novelty is that we apply two different merge procedures in the bidirectional labeling algorithm depending on whether the truck and its drone are travelling together (or separately) at the merge point.
- We present several computational analyses: First, we evaluate the new algorithmic components suggested to accelerate the merge procedure that constitutes the bottleneck operation of the overall BPC algorithm. Second, monodirectional labeling is compared with the implicit bidirectional labeling. Moreover, we compare the two objectives and the impact of the drone's and truck's speeds and routing costs on the structure and quality of solutions. Finally, results and analyses with TSP-D and VRP-D instances from the literature and with the VRP-D variant with limited flight range are presented. They show that the BPC algorithm is widely applicable.

The remainder of this work is structured as follows: We review the pertinent literature in Section 4.2. Section 4.3 formally introduces the VRP-D. In Section 4.4, we present the new BPC algorithm to solve the VRP-D with a particular focus on the implicit bidirectional labeling algorithm. Computational results are discussed in Section 4.5, and final conclusions are drawn in Section 4.6.

4.2 Literature Review

The integration of drones in vehicle routing is receiving increasing attention in the research literature (Otto *et al.*, 2018; Chung *et al.*, 2020; Macrina *et al.*, 2020; Moshref-Javadi and Winkenbach, 2021; Madani and Ndiaye, 2022). These surveys show the vast amount of scientific contributions that have been published in recent years, where heuristic solution approaches are predominant. For the sake of brevity, we limit the scope of the literature review to exact optimization approaches in the context of TSP-D and VRP-D variants of type (4) in the classification of Otto *et al.* (2018), i.e., when trucks and drones operate together as synchronized working units.

Roberti and Ruthmair (2021) focus on the basic variant of the TSP-D, i.e., a single truck hosting a single drone that can carry not more than one package and hence serve only one drone customer per launch from the truck. The objective is the minimization of the route's duration. The methodological contributions

of Roberti and Ruthmair (2021) is the presentation and computational analysis of a *mixed integer programming* (MIP) formulation and a *dynamic-programming* (DP) formulation for both the basic TSP-D and several extensions. Instances with up to 39 customers are solved with a branch-and-price algorithm that uses an *ng*-route relaxation of the DP formulation. These results can be considered a milestone, as prior to that only much smaller instances could be solved exactly. Moreover, their work provides an excellent overview of TSP-D variants, so that we can be brief with respect to the single-truck problems. Only the very recent approach for the TSP-D by Blufstein *et al.* (2024), that was developed in parallel with our work, must be mentioned: The authors develop an exact algorithm based on a decremental state-space relaxation. As we do, they derive a bidirectional labeling algorithm that is based on the network of Roberti and Ruthmair (2021). However, their price-fix-and-augment method is tailored to the TSP-D and applies a sophisticated combination of column generation, variable fixing using completion bounds, and a *ng*-neighborhood extension to solve the TSP-D. They introduce a restricted notion of *ng*-feasibility that leads to improved lower bounds (compared to Roberti and Ruthmair, 2021) that result from the solution of the associated dynamic-programming relaxations. Additionally, a partial dominance rule make use of the new notion and further accelerates the solution of the dynamic programs. It is remarkable that their algorithm solves every TSP-D benchmark instance of the testbed with up to 59 customers. Unfortunately, their algorithm is not directly applicable to the VRP-D. Obviously, their underlying state space has different states for different tour lengths and does not need to keep track of the accumulated load. More important is that their variable fixing fully exploits the TSP property that *the* route visits each customer exactly once. Moreover, the exploitation of tentative upper bounds of the TSP-D (for variable fixing) cannot be used when solving multiple pricing problems as needed in BPC.

The first article dealing with a VRP-D variant of type (4) was published by Wang *et al.* (2017). They define the VRP-D as a generalization of the TSP-D by using multiple trucks each equipped with multiple drones. Wang *et al.* (2017) assume that each drone is assigned to a specific truck and that a drone can only be released and picked up at either a customer location or the depot. The objective is to minimize the sum of the durations of the routes. Besides the introduction of the VRP-D, they conduct comprehensive analyses of worst-case scenarios on the benefits of using trucks and drones as synchronized work units instead of truck-only solutions. The same authors later extended their analyses in a follow-up paper (Poikonen *et al.*, 2017). Bakir and Tiniç (2020) consider the VRP-D with flexible drones, i.e., a drone does not necessarily have to return to the truck from where it was released. They formulate the problem as a *mixed integer linear program* (MILP) on a time-space network and propose a dynamic discretization

discovery approach that solves instances with up to 25 customers within five hours. Tamke and Buscher (2021) use a branch-and-cut algorithm to solve a version of the VRP-D in which each truck can be equipped with more than one drone. For larger instances, the number of drones on each truck is limited to two drones. Instances with up to 29 customers are solved within twelve hours. Zhen *et al.* (2023) developed a BPC algorithm to solve the VRP-D with the objective to minimize the total routing cost. They can solve instances with up to 14 customers to proven optimality within a 3-hour time limit. In the publication by Zhou *et al.* (2023), each truck is equipped with more than one drone that is uniquely assigned to this truck. The truck has to wait at a customer location until all launched drones return before serving the next customer. Such a direct return policy also simplifies the synchronization between trucks and drones in the branch-and-price algorithms of Zhou *et al.* (2023). For this setting with a simple synchronization, the authors develop a branch-and-price algorithm that uses a bidirectional labeling algorithm that solves some instances with up to 35 customers within three hours. Li and Wang (2022) extend the VRP-D by considering additional customer time windows. To the best of our knowledge, they propose the first exact solution approach which is a BPC algorithm for this VRP-D variant. Instances with up to 50 customers are solved to optimality within 20 hours. To solve larger instances, they combine the BPC algorithm with an adaptive large neighborhood search. In the BPC approaches of Wang and Sheu (2019) and Xia *et al.* (2023), the practically and methodologically difficult rigid synchronization between trucks and drones is relaxed with the introduction of *drone hubs* (or *docking hub*), i.e., the drones do not return to the trucks.

The heuristic community has introduced many VRP-D variants. The existing exact approaches cover only some of these variants, but they all consume considerable computation time (several hours) to solve small-sized instances to optimality. From an exact solution perspective, the planning of combined delivery operations with trucks and drones collaborating in a synchronized way is very challenging (Drexl, 2012). We present a faster exact BPC solution algorithm for the basic version of the VRP-D in which trucks and drones work together as synchronized working units. We consider duration and routing cost objectives and show that many variants discussed in the literature can be solved similarly.

4.3 The Vehicle Routing Problem with Drones

We define the VRP-D on an undirected complete graph (V, E) with vertex set V and edge set E. The vertex set V comprises $\{0, 0'\} \cup N$, where 0 and $0'$ represent the *depot* (two copies for start point and end point) and $N = \{1, 2, \ldots, m\}$ denotes the set of *customers*. Each customer $v \in N$ has a positive integer *demand* q_v which

has to be served by either a truck or a drone. If the demand q_v exceeds a given threshold $\bar{q}$, v is called a *truck-only customer* which cannot be served by drone.

A fleet of K homogeneous trucks is stationed at the depot. Each truck has a *capacity* Q and is equipped with a single drone (the assignment of the drone to its truck is assumed fixed). The truck and its drone can either move together or separately. When truck and drone move together, only the truck can *serve* customers. The drone can leave the truck at the depot 0 or at any customer location to operate alone. In the meantime, the truck continues its journey and can serve other customers on its own. The drone must return to the same truck at the truck's current location, which is either the location of a customer served by the truck or the depot $0'$. Note that the drone is not allowed to leave and return to the truck at the same customer. Moreover, each drone has a limited capacity in the sense that it can serve at most one customer before returning back to its truck. Accordingly, a *route* in the VRP-D is a pair $r = (P, D)$ consisting of the *truck path* P and a possibly empty sequence D of *drone subpaths*. The truck path $P = (v_0, v_1, \ldots, v_\ell, v_{\ell+1})$ starts at the depot $v_0 = 0$, serves the customers $v_p \in N$ for all $p \in \{1, \ldots, \ell\}$ in this sequence, and ends at the depot $v_{\ell+1} = 0'$. The drone subpaths $D = (D^1, \ldots, D^d)$ specify the drone flights along the truck path P. For each index $s \in \{1, \ldots, d\}$, each subpath $D^s = \langle v^s, k^s, w^s \rangle$ contains exactly three vertices: at $v^s \in \{0\} \cup N$ the drone leaves the truck, $k^s \in N$ specifies the customer that is served by the drone, and $w^s \in N \cup \{0'\}$ indicates where the drone lands on the truck again. In a feasible route $r = (P, D)$ there exist, for each index $s \in \{1, 2, \ldots, d\}$, two indices $g, h \in \{0, 1, \ldots, \ell, \ell + 1\}$ with $g < h$ such that $v^s = v_g$ and $w^s = v_h$ holds. For convenience, we define $I(s) = \{g, g + 1, \ldots, h - 1\}$ as the positions (indices) of P where either the drone leaves the truck or the truck serves a customer alone (intentionally, the position where the drone lands on the truck is not included). If a route contains more than one drone subpath, i.e., $d > 1$, the sets $I(s)$ and $I(s')$ for $s \neq s'$, $1 \leq s, s' \leq d$ must have empty intersection (the segments where the truck is alone do not overlap). Finally, the route $r = (P, D)$ is *feasible* if $\sum_{p=1}^{\ell} q_{v_p} + \sum_{s=1}^{d} q_{k^s} \leq Q$ and $q_{k^s} \leq \bar{q}$ for all $s \in \{1, \ldots, d\}$. The route $r = (P, D)$ is *elementary* if $v_1, \ldots, v_\ell$ and $k^1, \ldots, k^d$ are all different.

To distinguish between routing-cost and duration minimization, each edge $\{v, w\} \in E$ is associated with a non-negative routing cost c_{vw} and a non-negative travel time t_{vw} for a truck. Likewise, the routing cost of a drone is denoted by c^{dr}_{vw} and the travel time by t^{dr}_{vw}. The routing cost c_r of a route $r = (P, D)$ is defined as the sum of the routing costs of the edges traversed by both types of vehicles, i.e., $c_r = \sum_{p=0}^{\ell} c_{v_p, v_{p+1}} + \sum_{s=1}^{d} (c^{\mathrm{dr}}_{v^s, k^s} + c^{\mathrm{dr}}_{k^s, w^s})$. The duration t_r of a route $r = (P, D)$ is more intricate to describe, because the truck (drone) may have to wait for the drone (truck) depending on which vehicle is faster on a particular segment. Let

$I = \bigcup_{s=1}^{d} I(s)$. Then, the duration of route r is

$$t_r = \sum_{p \in \{0,1,\dots,\ell\} \setminus I} t_{v_p, v_{p+1}} + \sum_{s=1}^{d} \max \left\{ \sum_{p \in I(s)} t_{v_p, v_{p+1}}, t^{\mathrm{dr}}_{v^s, k^s} + t^{\mathrm{dr}}_{k^s, w^s} \right\}. \tag{4.1}$$

The first term describes travel times when truck and drone travel together, while the second term sums over the d drone subpaths taking the maximum of the travel time of either the truck or the drone.

The task of the VRP-D is to determine a set $\mathscr{R}$ of at most K feasible routes such that each customer is served exactly once and either $\sum_{r \in \mathscr{R}} c_r$ or $\sum_{r \in \mathscr{R}} t_r$ is minimized.

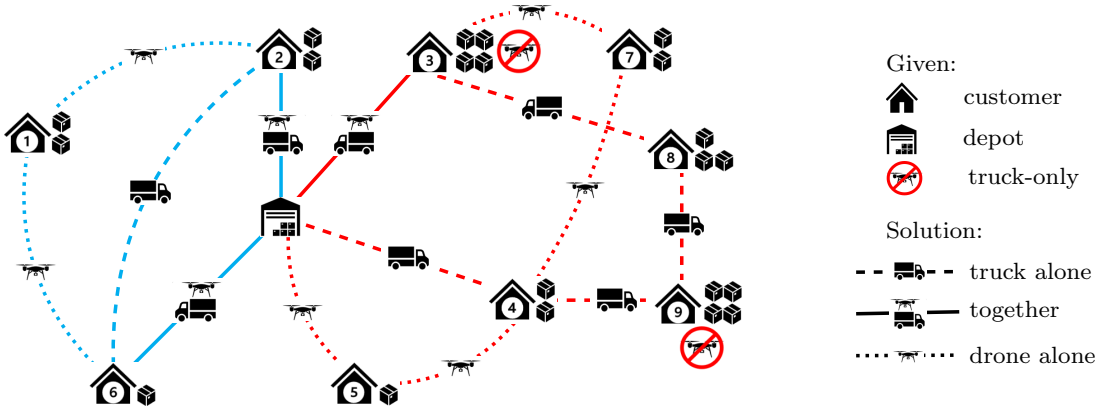


Figure 4.1: VRP-D instance with a feasible solution.

Example 1 *Figure 4.1 shows a VRP-D instance with customers* $N = \{1, 2, \dots, 9\}$ *(customers are shown as houses with numbers) as well as a feasible solution with two routes that both have a capacity of* $Q = 20$*. On the first route (depicted in red; we assume an anti-clockwise traversal), the truck releases its drone directly at the depot. While the truck travels to customer 4 and serves it, the drone moves to customer 5, serves this customer, and returns back to the truck at customer 4. Immediately at customer 4, truck and drone separate again: while the truck serves the customers 9, 8, and 3 alone, the drone serves customer 7 and lands on the truck again at the location of customer 3. Afterwards, truck and drone together travel back to the depot. Hence, the route serves customers* $\{3, 4, 5, 7, 8, 9\}$*. The corresponding truck path is* $P = (v_0, v_1, \dots, v_6) = (0, 4, 9, 8, 3, 0')$ *and the sequence of drone subpaths is* $D = (\langle 0, 5, 4 \rangle, \langle 4, 7, 3 \rangle)$*. The boxes depicted beside the customers indicate the respective demand. The first route delivers 16 boxes and is therefore feasible.*

The second route (depicted in blue; we assume a clockwise traversal) serves another three customers $\{1, 2, 6\}$ *with* $P = (0, 6, 2, 0')$ *and* $D = (\langle 6, 1, 2\rangle)$*. Assuming that all travel costs and travel times of all edges* $\{v, w\} \in E$ *are* $c_{vw} = t_{vw} = c^{dr}_{vw} = t^{dr}_{vw} = 1$*, the truck has to wait for the drone at location 4, and vice versa, the drone has to wait for the truck at location 3 in the first (red) route. Hence, the duration of the first route is* $t_r = 6$ *while its cost is* $c_r = 9$*. Under these assumptions, the truck has to wait for the drone at location 2 in the second (blue) route, which has a duration of* $t_r = 4$ *while its cost is* $c_r = 5$*.* □

4.4 Branch-Price-and-Cut Algorithm

We develop a BPC algorithm to solve the VRP-D. BPC algorithms are 'the leading exact algorithms for solving many classes of vehicle routing problems' (see Costa *et al.*, 2019). They rely on an extensive, route-based formulation (a.k.a. master problem (MP)), i.e., a model with variables for all feasible routes. A BPC algorithm is a branch-and-bound algorithm in which, at each node, the linear relaxation of the MP is solved. At some nodes, valid inequalities are added to strengthen the linear relaxation. Due to the huge number of variables in the MP, relaxations are not solved directly but with a column-generation procedure: Iteratively, *restricted master programs* (RMPs) defined over a subset of routes are solved, missing routes are identified by solving subproblems (a.k.a. pricing problems), and these are added to the RMPs (Desaulniers *et al.*, 2005).

4.4.1 Route-based Formulation

Let Ω denote the set of all feasible VRP-D routes. The route-based formulation presented below uses binary variables λ_r to indicate whether a route $r \in \Omega$ is part of the optimal solution. In addition, the non-negative variable f describes the number of trucks employed. For each customer $v \in N$ and each route $r \in \Omega$, let the integer coefficient b_{vr} indicate the number of times the route r serves customer v.

$$\min \quad \sum_{r\in\Omega} c_r\lambda_r \qquad \text{or} \qquad \min \quad \sum_{r\in\Omega} t_r\lambda_r \tag{4.2a}$$

$$\text{subject to} \quad \sum_{r\in\Omega} b_{vr}\lambda_r = 1 \qquad \forall v \in N \tag{4.2b}$$

$$\sum_{r\in\Omega} \lambda_r - f = 0 \tag{4.2c}$$

$$\underline{K} \le f \le K \tag{4.2d}$$

$$\lambda_r \in \{0, 1\} \qquad \forall r \in \Omega \tag{4.2e}$$

The objective (4.2a) is to minimize the total routing cost or the sum of the durations of all routes. Constraints (4.2b) ensure that each customer is served exactly once. The number of trucks f in use is determined via (4.2c) and restricted to the interval between a lower bound $\underline{K}$ and the given fleet size K via (4.2d). The domain of the route variables is specified in (4.2e).

A valid lower bound $\underline{K}$ for the number of necessary trucks can be calculated by solving a bin-packing problem as follows: The bin size is set equal to the truck's capacity Q. For each customer $i \in N$, an item with a weight equal to demand q_i must be introduced. We solve the bin-packing problem with the arc-flow formulation of Valério de Carvalho (1999) using a MIP solver. Note that the optimal number $\underline{K}$ of bins is an exact lower bound that helps to not solve some infeasible branch-and-bound nodes, which may otherwise result from branching on the number of vehicles (see Section 4.4.4). Solving these infeasible nodes is often very time-consuming.

4.4.2 Column Generation

Let $(\pi_v)_{v \in N} \in \mathbb{R}^{|N|}$ be the dual prices of the set-partitioning constraints (4.2b) and $\mu \in \mathbb{R}$ be the dual price of constraint (4.2c). The reduced cost of a route $r \in \Omega$ is

$$\tilde{c}_r^{cost} = c_r - \sum_{v \in N} b_{vr}\pi_v - \mu \quad \text{or} \quad \tilde{c}_r^{dur} = t_r - \sum_{v \in N} b_{vr}\pi_v - \mu$$

depending on whether the objective is cost or duration minimization.

The emerging pricing problem can be modeled as a *shortest path problem with resource constraints* (SPPRC, Irnich and Desaulniers, 2005) and solved by means of a dynamic-programming labeling algorithm on an artificial network.

Artificial Network

Compared to the classical CVRP, the major additional complication in the VRP-D is that truck and drone can move independently, at least on parts of their itinerary. Already the decision which type of vehicle will serve which subset of the customers adds a substantial degree of freedom. All these decisions impact a route's cost and duration, which can no longer be computed as the length of a single path. We have shown in Section 4.3 that route feasibility constraints are much more involved compared to the CVRP, because a route in the given network (V, E) consists of a truck path and a sequence of drone subpaths that have to fit together as defined in Section 4.3.

For solving the single-vehicle version of the problem, i.e., the TSP-D, Roberti and Ruthmair (2021) model a route/tour with the help of an *artificial network* $\mathcal{N} =$

(W, A). The vertex set

$$W = \{(0,0)\} \cup (N \cup \{0'\}) \times (\{0\} \cup N) \cup \{(0',0')\} \subset V \times V$$

represents possible truck-and-drone positions, i.e., the vertex $(i^{\mathrm{tr}}, i^{\mathrm{dr}}) \in W$ describes that the truck is at position i^{tr} and the drone is at i^{dr}.

To lighten the notation, we assume in the following that writing $i \in W$ describes the vertex $(i^{\mathrm{tr}}, i^{\mathrm{dr}})$. When another vertex is needed, we write $j \in W$ to describe the vertex $(j^{\mathrm{tr}}, j^{\mathrm{dr}})$. Note that the symbols i and j will be exclusively used in this context.

In particular and with slight abuse of notation, the *origin* vertex $0 = (0,0) \in W$ represents the initial situation at the depot when truck and drone are together ready to start a route. Likewise, the *destination* vertex $0' = (0',0') \in W$ represents the final situation when truck and drone are back together at the depot after completing a route.

Arcs of the artificial network represent movements and a possible additional service of a customer by the drone. For the latter service aspect, let $N^{dr} = \{v \in N : q_v \leq \bar{q}\}$ denote such a service by drone and let $\perp$ denote that no feasible service can be performed. The above convention to write i for $(i^{\mathrm{tr}}, i^{\mathrm{dr}})$ and j for $(j^{\mathrm{tr}}, j^{\mathrm{dr}})$ allows us to compactly describe an arc of $\mathscr{N}$ by triplets $[i,j,k]$ written out as $[(i^{\mathrm{tr}}, i^{\mathrm{dr}}), (j^{\mathrm{tr}}, j^{\mathrm{dr}}), k]$ (we use square brackets to distinguish arcs of the artificial network $\mathscr{N}$ from the edges $\{v,w\}$ of the original network (V,E) and from subpaths $\langle v,k,w\rangle$). The arcs of the artificial network $\mathscr{N}$ fall into the following three categories: First, the *alone arcs*

$$A^{alone} = \big\{[i,j,\perp] \in W \times W \times \{\perp\} : i^{\mathrm{dr}} = j^{\mathrm{dr}} \text{ and } i^{\mathrm{tr}} \neq j^{\mathrm{tr}} \text{ and } i^{\mathrm{dr}} \neq j^{\mathrm{tr}}\big\} \quad (4.3a)$$

represent that the truck is moving from position i^{tr} to position j^{tr} while the drone is airborne with launch position $i^{\mathrm{dr}} = j^{\mathrm{dr}}$. Second, the *together arcs*

$$A^{tgthr} = \big\{[i,j,\perp] \in W \times W \times \{\perp\} : i^{\mathrm{tr}} = i^{\mathrm{dr}} \neq j^{\mathrm{tr}} = j^{\mathrm{dr}}\big\} \quad (4.3b)$$

represent that truck and drone are moving together from position $i^{\mathrm{tr}} = i^{\mathrm{dr}}$ to a new position $j^{\mathrm{tr}} = j^{\mathrm{dr}}$. In both cases, alone and together arcs, the customer j^{tr} is served. Third, the *drone arcs*

$$A^{drone} = \big\{[i,j,k] \in W \times W \times N^{dr} : i^{\mathrm{tr}} = j^{\mathrm{tr}} = j^{\mathrm{dr}} \neq i^{\mathrm{dr}} \text{ and } k \notin \{i^{\mathrm{dr}}, i^{\mathrm{tr}}\}\big\} \quad (4.3c)$$

represent that the drone flies from the position i^{dr} where it has separated from the truck to customer k, serves customer k, and subsequently flies to the position $i^{\mathrm{tr}} = j^{\mathrm{tr}} = j^{\mathrm{dr}}$ where the truck is waiting or will later arrive to pick up the drone. Note that the arc sets defined via (4.3a)–(4.3c) are disjoint. Hence, each arc in

$A = A^{alone} \cup A^{tgthr} \cup A^{drone}$ belongs to exactly one category (alone, together, or drone arc). Moreover, the arcs in $A^{alone} \cup A^{tgthr}$ are referred to as *truck arcs*, since they model serving customer j^{tr} by truck.

We stress that the artificial network $\mathscr{N}$ is directed and has parallel arcs, i.e., it is a *multi-digraph*. Parallel arcs are always drone arcs $[i, j, k] \in A^{drone}$ and they differ in the additional service that is performed at customer k.

The fundamental idea of Roberti and Ruthmair (2021) is to artificially separate truck movement decisions from drone movement decisions. When truck and drone separate, it is not necessary to know in advance which customer the drone will serve and where and when it will return to the truck. Hence, these decisions can be postponed. Instead, the bookkeeping reduces to record the position where the drone has departed from the truck. The truck is then routed independently serving an arbitrary number of customers. The postponed decision about the drone's subpath is made when truck and drone meet again. This enables a forward labeling algorithm with an efficient dominance procedure as shown in Section 4.4.2.

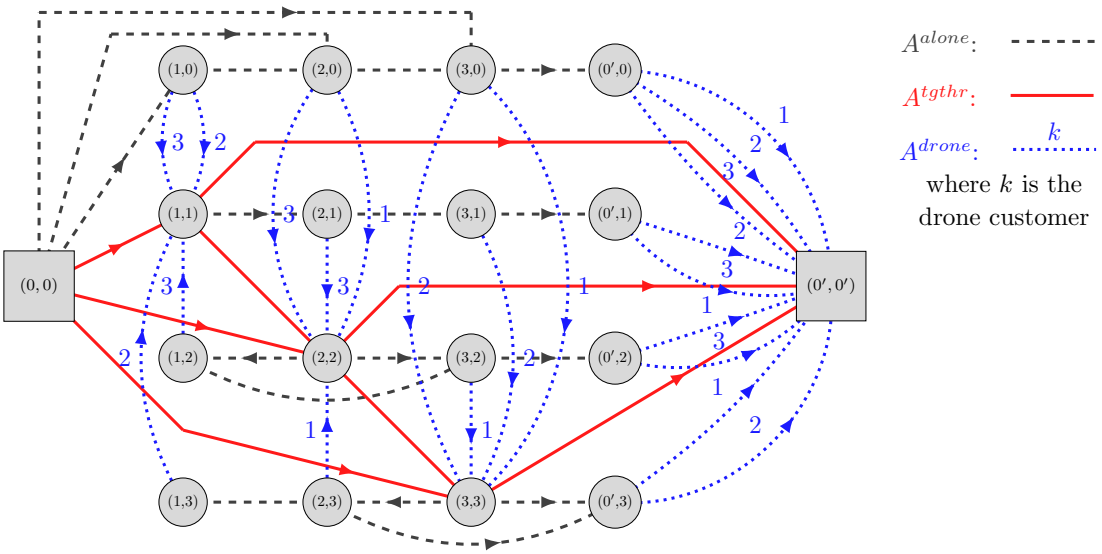


Figure 4.2: Artificial network $\mathscr{N} = (W, A)$ with customer set $N = \{1, 2, 3\}$. For the sake of simplicity, not all truck arcs are made explicit. Antiparallel arcs between vertices are depicted as undirected links. For example, in the first row, antiparallel arcs exist between the vertices $(1, 0)$, $(2, 0)$, and $(3, 0)$.

Example 2 *Figure 4.2 visualizes the artificial network for a VRP-D instance with three customers* $N = \{1, 2, 3\}$. *The path* $((0, 0), (1, 1), (2, 2), (3, 3), (0', 0'))$ *in* $\mathscr{N}$ *represents a route where truck and drone travel together all the way from the depot* 0

to the three customers 1, 2, and 3 (in this sequence), and back to the depot $0'$. *The path* $((0,0),(2,0),(2,2),(3,3),(0',0'))$ *in* $\mathscr{N}$ *has an associated unique truck path* $(0,2,3,0')$. *However, there are two parallel arcs connecting* $(2,0)$ *with* $(2,2)$ *leaving the choice of using one of them. Either customer 1 could be served leading to the associated drone subpath* $\langle 0,1,2\rangle$, *or customer 3 could be served leading to the associated drone subpath* $\langle 0,3,2\rangle$. *In the first case, the route is elementary, serving customers 1, 2, and 3 exactly once, while in the second case, the route is non-elementary, serving customer 3 twice and customer 2 once.* □

More generally, every 0-$0'$-path in $\mathscr{N}$, i.e., a path from $(0,0)$ to $(0',0')$, represents the routes of a truck and its drone. For simplicity, we refer to them as *a route* in the following. Since the artificial network is a multi-digraph, only the sequence of its arcs describes it uniquely. Note that the corresponding route is not necessarily elementary or feasible. The labeling algorithm presented in Section 4.4.2 checks elementary and feasibility.

Each 0-$0'$-path in $\mathscr{N}$ can be resolved into a corresponding truck path and drone subpaths using the information on each arc $[i,j,k]$. If the arc is a truck arc, i.e., $k=\perp$, the trucks destination j^{tr} has to be appended to the truck path. If the arc is a drone arc, i.e., $k \in N^{dr}$, the drone subpath $\langle v,k,w\rangle$ contains the following three pieces of information: the release location $v = i^{\mathrm{dr}}$ of the drone, the drone customer k, and the drone's destination location $w = j^{\mathrm{dr}}$.

The following example introduces a second, probably more intuitive (graphical) notation for routes in the artificial network.

Example 3 *We consider a VRP-D instance with* $N=\{1,2,3,4,5\}$. *The* 0-$0'$*-path* $([(0,0),(1,1),\perp],[(1,1),(2,1),\perp],[(2,1),(2,2),3],[(2,2),(4,2),\perp],[(4,2),(0',2),\perp],$ $[(0',2),(0',0'),5])$ *has* $P=(0,1,2,4,0')$ *and* $D=(\langle 1,3,2\rangle,\langle 2,5,0'\rangle)$. *In the following, we also use the more intuitive representation*

$$(0,0) = (1,1) - (2,1) \overset{3}{-} (2,2) - (4,2) - (0',2) \overset{5}{-} (0',0')$$

as a sequence of vertices where the connections are either $=$, $-$, *or* $\overset{k}{-}$. *The symbols* $-$ $(=)$ *indicate a connection with an alone (together) arc, while the symbol* $\overset{k}{-}$ *indicates a drone arc of type* $[(\cdot,\cdot),(\cdot,\cdot),k]$. □

Forward Labeling

In this section, we present a dynamic-programming labeling algorithm to solve the SPPRC pricing problem. We begin with a basic forward labeling approach and then discuss refinements including an implicit bidirectional labeling and a non-trivial merge procedure.

In our labeling algorithm, each label refers to a partial path $P_i = (0, \dots, i)$ that starts at the origin depot and ends at a vertex $i = (i^{\text{tr}}, i^{\text{dr}}) \in W$. The label stores the corresponding resource consumption up to the vertex i. The set of attributes (i.e., resources) depends on the objective function used in the VRP-D.

Cost Objective For the cost objective and a partial path P_i, the associated label $\mathcal{L}_i$ is defined by the resource vector $R = (R_i^{rdc,cost}, R_i^{load}, (R_i^{cust,n})_{n \in N})$. In the following, we write in short $\mathcal{L} = (R)$ when referring to a label and its resource vector. The resources are defined as follows:

$R_i^{rdc,cost}$: the reduced cost of path P_i;
R_i^{load}: the total demand served by path P_i excluding the demand of i; and
$R_i^{cust,n}$: the number of times that customer $n \in N$ is served along the path P_i.

The initial label is given by $\mathcal{L}_0 = (0, 0, \mathbf{0})$. Propagating a label $\mathcal{L}_i$ over an arbitrary arc $a = [i, j, k] \in A$ results in a new label $\mathcal{L}_j$ at vertex $j = (j^{\text{tr}}, j^{\text{dr}})$. We assume that dual prices $\pi = (\pi_v)_{v \in N}$ of the constraints (4.2b) and the dual price μ of the constraint (4.2c) are given. For convenience, we define $\pi_{0'} = \mu$ for the destination $0'$. Moreover, we define $q_0 = q_{0'} = 0$ for the depot. The associated resource values for the partial path $P_j = (P_i, j)$ are computed with the help of the following *resource extension functions* (REFs, Irnich, 2008):

$$R_j^{rdc,cost} = \begin{cases} R_i^{rdc,cost} + c_{i^{\text{tr}},j^{\text{tr}}} - \pi_{j^{\text{tr}}}, & \text{if } [i,j,\perp] \in A^{alone} \cup A^{tgthr} \\ R_i^{rdc,cost} + c^{\text{dr}}_{i^{\text{dr}},k} + c^{\text{dr}}_{k,j^{\text{dr}}} - \pi_k, & \text{if } [i,j,k] \in A^{drone} \end{cases} \tag{4.4a}$$

$$R_j^{load} = \begin{cases} R_i^{load} + q_{i^{\text{tr}}}, & \text{if } [i,j,\perp] \in A^{alone} \cup A^{tgthr} \\ R_i^{load} + q_k, & \text{if } [i,j,k] \in A^{drone} \end{cases} \tag{4.4b}$$

$$R_j^{cust,n} = \begin{cases} R_i^{cust,n} + 1, & \text{if } n = j^{\text{tr}} \text{ and } [i,j,\perp] \in A^{alone} \cup A^{tgthr} \\ R_i^{cust,n} + 1, & \text{if } n = k \text{ and } [i,j,k] \in A^{drone} \\ R_i^{cust,n}, & \text{otherwise} \end{cases} \tag{4.4c}$$

Note that in (4.4b) the update of the load resource is postponed regarding the customer j^{tr} to the point after the truck leaves customer j^{tr}. We will see later that this is convenient for the symmetric bidirectional labeling algorithm.

The new label $\mathcal{L}_j$ is feasible, if the capacity and the elementary constraints are fulfilled, i.e.,

$$R_j^{load} \le \begin{cases} Q - q_{j^{\text{tr}}}, & \text{if } [i,j,\perp] \in A^{alone} \cup A^{tgthr} \\ Q, & \text{if } [i,j,k] \in A^{drone} \end{cases} \tag{4.5a}$$

$$R_j^{cust,n} \le 1, \qquad \forall n \in N. \tag{4.5b}$$

Infeasible labels are discarded immediately. To avoid the enumeration of all feasible paths, redundant labels are eliminated with the help of dominance tests. Since all

forward REFs are non-decreasing, we can use the standard dominance rules (Irnich and Desaulniers, 2005):

Dominance Rule 1 *We consider two labels at the same vertex of the artificial network. Label* $\mathcal{L}_1 = (R_1^{rdc,cost}, R_1^{load}, (R_1^{cust,n})_{n\in N})$ *dominates another label* $\mathcal{L}_2 = (R_2^{rdc,cost}, R_2^{load}, (R_2^{cust,n})_{n\in N})$ *if* $R_1^{rdc,cost} \leq R_2^{rdc,cost}$, $R_1^{load} \leq R_2^{load}$, *and* $R_1^{cust,n} \leq R_2^{cust,n}$ *holds for all* $n \in N$.

Duration Objective When considering the duration objective, additional information is required. We follow the ideas presented in Roberti and Ruthmair (2021). Hence, a label now comprises the resources $(R_i^{rdc,dur}, R_i^{dur}, R_i^{load}, (R_i^{cust,n})_{n\in N})$ defined as:

$R_i^{rdc,dur}$: the reduced cost based on the duration of path P_i;
R_i^{dur}: the duration for which the truck travels alone on the last open subtour of path P_i;

and R_i^{load} and $R_i^{cust,n}$ as defined in (4.4b) and (4.4c), respectively. A subtour is an *open* subtour if one or several alone arcs have been traversed without closing the subtour with a drone arc.

The initial label is now given by $\mathcal{L}_0 = (0, 0, 0, \mathbf{0})$. For an arc $[i, j, k] \in A$, the resources rdc, dur and dur are updated by:

$$R_j^{rdc,dur} = \begin{cases} R_i^{rdc,dur} - \pi_{j^{\mathrm{tr}}}, & \text{if } [i,j,\perp] \in A^{alone} \\ R_i^{rdc,dur} + t_{i^{\mathrm{tr}},j^{\mathrm{tr}}} - \pi_{j^{\mathrm{tr}}}, & \text{if } [i,j,\perp] \in A^{tgthr} \\ R_i^{rdc,dur} + \max\{t^{\mathrm{dr}}_{i^{\mathrm{dr}},k} + t^{\mathrm{dr}}_{k,j^{\mathrm{dr}}}, R_i^{dur}\} - \pi_k, & \text{if } [i,j,k] \in A^{drone} \end{cases} \tag{4.6a}$$

$$R_j^{dur} = \begin{cases} R_i^{dur} + t_{i^{\mathrm{tr}},j^{\mathrm{tr}}}, & \text{if } [i,j,\perp] \in A^{alone} \\ 0, & \text{if } [i,j,\cdot] \in A^{drone} \cup A^{tgthr} \end{cases} \tag{4.6b}$$

Compared to the cost objective, there are no further feasibility requirements in addition to (4.5).

Dominance Rule 2 *We consider two labels at the same vertex* $i = (i^{tr}, i^{dr})$ *of the artificial network. Label* $\mathcal{L}_1 = (R_1^{rdc,dur}, R_1^{dur}, R_1^{load}, (R_1^{cust,n})_{n\in N})$ *dominates another label* $\mathcal{L}_2 = (R_2^{rdc,dur}, R_2^{dur}, R_2^{load}, (R_2^{cust,n})_{n\in N})$ *if* $R_1^{load} \leq R_2^{load}$, $R_1^{cust,n} \leq R_2^{cust,n}$ *for all* $n \in N$, *and*

(1) $i^{tr} = i^{dr}$ *and* $R_1^{rdc,dur} \leq R_2^{rdc,dur}$,
(2) or: $i^{tr} \neq i^{dr}$ *and* $R_1^{rdc,dur} \leq R_2^{rdc,dur}$, $R_1^{rdc,dur} + R_1^{dur} \leq R_2^{rdc,dur} + R_2^{dur}$,
(3) or: $i^{tr} \neq i^{dr}$ *and* $R_1^{rdc,dur} + R_1^{dur} \leq R_2^{rdc,dur} + R_2^{dur}$ *and* $R_1^{rdc,dur} + \max_{k,v}\{t^{dr}_{i^{dr},k} + t^{dr}_{k,v}\} \leq R_2^{rdc,dur} + R_2^{dur}$, *where the indices* k *and* v *in the max-term run over* $k \in N$ *and* $v \in N \cup \{0'\}$ *with* $k \neq v$.

Note that in part (1) the precondition $i^{\mathrm{tr}} = i^{\mathrm{dr}}$ implies $R_1^{dur} = R_2^{dur} = 0$ because of (4.6b), which proves correctness of the dominance due to non-decreasing REFs (4.4b), (4.4c), (4.6a), and (4.6b). In part (2), the condition $R_1^{rdc,dur} + R_1^{dur} \leq R_2^{rdc,dur} + R_2^{dur}$ is fulfilled whenever $R_1^{rdc,dur} \leq R_2^{rdc,dur}$ and $R_1^{dur} \leq R_2^{dur}$, which shows that this dominance is stronger than a simpler pairwise comparison with $\leq$. The two conditions in part (3) guarantee that the reduced cost of the first path is not larger than that of the second when both are extended to a vertex j with $j^{\mathrm{tr}} = j^{\mathrm{dr}}$, either directly or with additional intermediate vertices. Such a vertex j with $j^{\mathrm{tr}} = j^{\mathrm{dr}}$ is certainly reached when arriving at the destination depot $0' = (0', 0')$. The validity of this Dominance Rule 2 is formally shown in Roberti and Ruthmair (2021).

Symmetry Considerations

By definition, the VRP-D is a symmetric routing problem, thus defined using an undirected graph. In this graph (V, E), the two depot vertices 0 and $0'$ can be swapped without changing the VRP-D instance. By reversing a route's truck path and drone subpaths (if any) and by swapping 0 and $0'$ afterwards, the resulting route is equivalent to the one considered originally. In particular, both routes serve the same customers and have identical routing cost and duration. In contrast, the artificial network $\mathscr{N} = (W, A)$ is intentionally defined as a directed multigraph. This is consistent with the idea that the second component i^{dr} of a vertex $i \in W$ indicates the location where the truck has released the drone in case that $i^{\mathrm{dr}} \neq i^{\mathrm{tr}}$ holds. This piece of information is asymmetric because the choice of the drone's later landing location is unspecified. Even more, a given route r and the corresponding reversed route r' are represented, in general, by paths that visit different vertices in the artificial network $\mathscr{N}$ as shown in the following example.

Example 4 (continued from Example 3) *The truck path* $P = (0, 1, 2, 4, 0')$ *and associated drone subpaths* $D = (\langle 1, 3, 2\rangle, \langle 2, 5, 0'\rangle)$ *have reversed counterparts* $P' = (0, 4, 2, 1, 0')$ *and* $D' = (\langle 0, 5, 2\rangle, \langle 2, 3, 1\rangle)$*, which are perfectly symmetric in the original network* (V, E)*. In the artificial network, however, the corresponding* 0-$0'$*-paths*

$$(0,0) = (1,1) - (2,1) \overset{3}{-} (2,2) - (4,2) - (0',2) \overset{5}{-} (0',0')$$

and

$$(0,0) - (4,0) - (2,0) \overset{5}{-} (2,2) - (1,2) \overset{3}{-} (1,1) = (0',0')$$

differ in the vertices

$$(1,2), (2,1), (4,2), (4,0), (0',2), \text{ and } (2,0).$$

□

The example shows that the reversal of paths in the asymmetric network is somewhat confusing and not straightforward. It seems rather unpromising to develop a bidirectional labeling algorithm over the artificial network. These observations may explain why Roberti and Ruthmair (2021) did not present a bidirectional approach for the TSP-D although the general methodology for an exact solution algorithm for TSP variants is known (Baldacci *et al.*, 2012; Tilk and Irnich, 2017; Lera-Romero *et al.*, 2022).

Implicit Backward and Bidirectional Labeling

For many SPPRCs, the number of labels generated by a monodirectional labeling algorithm increases rapidly when feasible partial paths grow longer. To mitigate this so-called *combinatorial explosion*, Righini and Salani (2006) introduced a bounded bidirectional labeling procedure for SPPRCs. In a bidirectional labeling, not only forward labels but also backward labels are created by propagating a label against the orientation of the arcs starting from the sink of the network. Both forward and backward labels are only extended up to a so-called *halfway point* (HWP). When forward and backward propagation is completed, suitable labels are merged to obtain complete feasible origin-destination-paths.

The selection of a monotone resource (often the load or time attribute) and the concrete choice of the HWP not exactly 'in the middle' can be important for the labeling algorithm's performance for asymmetric SPPRC instances (Tilk *et al.*, 2017). If the underlying network is however perfectly symmetric, an *implicit* bidirectional technique can be applied to further accelerate the labeling. Since forward and backward labeling create exactly the same partial paths, it suffices to propagate labels only in forward direction. These labels are combined in the merge procedure. Implicit bidirectional labeling has already been applied successfully in several works (e.g., Bode and Irnich, 2012; Goeke *et al.*, 2019; Heßler and Irnich, 2023).

We now show that implicit bidirectional labeling is possible for the artificial network of the VRP-D although it is asymmetric. Since the only constrained resource is the capacity of the trucks, we use the load resource R^{load} as the critical resource and fix the HWP to $Q/2$, i.e., only labels with accumulated load $R^{load} < Q/2$ are extended.

The following observation can be made from Example 4: The two 0-0′-paths that represent reversed routes differ only in vertices $i \in W$ with $i^{\mathrm{tr}} \neq i^{\mathrm{dr}}$. These vertices occur in segments of the routes where truck and drone move independently. In contrast, when truck and drone are at the same location $i^{\mathrm{tr}} = i^{\mathrm{dr}}$, segments are identical. In Example 4, the sequence $(0,0)$, $(1,1)$, $(2,2)$, $(0',0')$ is traversed in either this or the reversed order.

As a consequence, we apply two different merge procedures depending on whether

$i^{\mathrm{tr}} = i^{\mathrm{dr}}$ or $i^{\mathrm{tr}} \neq i^{\mathrm{dr}}$ holds at the vertex i of a label that could potentially be merged. The procedure for $i^{\mathrm{tr}} = i^{\mathrm{dr}}$ is rather standard, while the one for $i^{\mathrm{tr}} \neq i^{\mathrm{dr}}$ is more sophisticated.

Merge at Vertices where Truck and Drone are together ($i^{\mathbf{tr}} = i^{\mathbf{dr}}$) In this case, we consider two labels $\mathcal{L} = (R)$ and $\mathcal{L}' = (R')$ of partial paths that both end at the same vertex i with $i^{\mathrm{tr}} = i^{\mathrm{dr}}$ (this is the so-called *merge vertex*). These vertices exist for all $i^{\mathrm{tr}} \in V$. The merge condition is the one known from the CVRP, i.e., the resulting path must respect the truck's capacity and be elementary:

$$R^{load} + R'^{load} + q_{i^{\mathrm{tr}}} \leq Q \tag{4.7a}$$

$$R^{cust,n} + R'^{cust,n} \leq 1 \qquad \forall v \in N \setminus \{i^{\mathrm{tr}}\} \tag{4.7b}$$

The reduced cost of the route r resulting from the merge is

$$\tilde{c}_r^{cost} = R^{rdc,cost} + R'^{rdc,cost} + \pi_{i^{\mathrm{tr}}} \qquad \text{and} \qquad \tilde{c}_r^{dur} = R^{rdc,dur} + R'^{rdc,dur} + \pi_{i^{\mathrm{tr}}}. \tag{4.8a}$$

The route itself is $r = (P, \mathrm{rev}^-(P'))$ where P and P' are the partial paths associated with $\mathcal{L}$ and $\mathcal{L}'$, respectively, and the function $\mathrm{rev}^-(P')$ denotes the reversed path to P' (in the sense of Example 4; also replacing 0 by $0'$; details are provided below) from which the last vertex is removed (to avoid including the merge vertex twice).

Even if we do not need to know the reversed path for checking the merge conditions and computing the reduced cost, we must formally describe the reversal of the second partial path P' associated with $\mathcal{L}'$. For the reversal of an arbitrary path, it suffices to describe the reversal of a *segment* between two subsequent vertices i and j with $v = i^{\mathrm{tr}} = i^{\mathrm{dr}}$ and $w = j^{\mathrm{tr}} = j^{\mathrm{dr}}$. In general, such a segment in the artificial network is either trivial, i.e., it consists of a single together arc, or it is of the form

$$(v, v) - (v_1, v) - (v_2, v) - \cdots - (v_l, v) - (w, v) \overset{k}{-} (w, w)$$

with associated drone subpath $\langle v, k, w \rangle$. In this segment, all arcs are alone arcs except for the last arc which is the drone arc $[(w, v), (w, w), k]$. The reversed segment is

$$(w, w) - (v_l, w) - \cdots - (v_2, w) - (v_1, w) - (v, w) \overset{k}{-} (v, v),$$

i.e., (a) the sequence $v_1, v_2, \ldots, v_l$ of truck positions is reversed in the usual way, while the drone position v in the forward segment is replaced by w for all vertices in the sequence except the first and last vertex and (b) the second last vertex (w, v) in the forward segment becomes the second last vertex (v, w) in the reversed segment. Note also that in both segments, the customer k is served by the drone, i.e., the last arc in the reversed path is the drone arc $[(v, w), (v, v), k]$. Finally, multiple reversed segments must be concatenated again in the usual way.

Example 5 (continued from Example 4) *We assume that the path* $(0,0) = (1,1) - (2,1) \overset{3}{-} (2,2) - (4,2) - (0',2) \overset{5}{-} (0',0')$ *considered in Example 4 is feasible, i.e.,* $\sum_{l=1}^{5} q_l \leq Q$ *is fulfilled.*

Let vertex $(2,2)$ *be the merge vertex for the path, hence, it must hold* $q_1 < Q/2$ *and* $q_1 + q_3 \geq Q/2$*. In this case, the merge of the two partial paths*

$$(0,0) = (1,1) - (2,1) \overset{3}{-} (2,2) \quad \textit{and} \quad (0,0) - (4,0) - (2,0) \overset{5}{-} (2,2)$$

produces the given path. Note that these are exactly the partial paths (up to vertex $(2,2)$*) of the two partial paths shown in Example 4.*

Their labels $\mathcal{L} = (R)$ *and* $\mathcal{L}' = (R')$ *qualify for the merge, because* $R^{load} = q_1 + q_3 \geq Q/2$*. The merge conditions are also fulfilled, what can be seen as follows: First, the load-related feasibility test* (4.7a) *adds together* $R^{load} = q_1 + q_3$*,* $R'^{load} = q_4 + q_5$*, and* q_2 *for the merge vertex* $(2,2)$*, which is exactly the demand delivered by the resulting route. The assumption that the given route is feasible implies that* (4.7a) *is fulfilled.*

Second, regarding the served customers, $R^{cust,n} = 1$ *if and only if* $n \in \{1,2,3\}$*, and* $R'^{cust,n} = 1$ *if and only if* $n \in \{2,4,5\}$*. The elementarity check* (4.7b) *excludes* $n = 2$ *from the test* $R^{cust,n} + R'^{cust,n} \leq 1$*. Hence, labels* $\mathcal{L}$ *and* $\mathcal{L}'$ *passes all merge conditions.* □

Example 5 also provides a good example of why we must delay the addition of the customer demand in the REF (4.4b). In an alternative model, the demand would always be immediately added, i.e., $q_{j^{\text{tr}}}$ were added to R^{load} on all arcs $[(i^{\text{tr}}, i^{\text{dr}}), (j^{\text{tr}}, j^{\text{dr}}), \perp]$ with $i^{\text{tr}} \neq j^{\text{tr}}$. Then, the partial paths of both labels R and R' could sooner pass the HWP in which case they would not be extended to the merge point.

Example 6 (continued from Example 5) *We consider an instance of the VRP-D with demands* $q_1 = q_2 = q_4 = 10$ *and* $q_3 = q_5 = 1$ *and a vehicle capacity of* $Q = 38$ *as well as the same paths as in Example 5. Moreover, we assume a modified REF for the load resource that directly incorporates the demand of a new customer visited by the truck.*

The first path $(0,0) = (1,1) - (2,1) \overset{3}{-} (2,2)$ *would pass the HWP* $Q/2 = 19$ *already at vertex* $(2,1)$ *with an accumulated demand of* $q_1 + q_2 = 20 > Q/2$*. Similarly, the second path* $(0,0) - (4,0) - (2,0) \overset{5}{-} (2,2)$ *would pass the HWP already at vertex* $(2,0)$ *with an accumulated demand of* $q_4 + q_2 = 20 > Q/2$*. As a result, the merge procedure would not generate the route that has the two partial paths as forward and (reversed) backward partial path.* □

Merge at Pairs of Vertices where Truck and Drone are at Different Locations ($i^{\mathbf{tr}} \neq i^{\mathbf{dr}}$) This second case is more intricate: A label belonging to a vertex $(i^{\mathrm{tr}}, i^{\mathrm{dr}})$ with $i^{\mathrm{tr}} \neq i^{\mathrm{dr}}$ *must not* be merged with labels belonging to the same vertex. Instead, the second label must belong to a vertex with identical truck position i^{tr} but a different drone position $j^{\mathrm{dr}} \neq i^{\mathrm{dr}}$, i.e., a vertex of the type $(i^{\mathrm{tr}}, j^{\mathrm{dr}})$. As a consequence, the merge must be performed 'over an arc' (and $(i^{\mathrm{tr}}, i^{\mathrm{dr}})$ and $(i^{\mathrm{tr}}, j^{\mathrm{dr}})$ are the so-called merge vertices) so that the total length of the forward and backward partial paths is by one arc smaller than the length of the resulting merged route.

Example 7 (continued from Example 5) *We consider the same route as in Example 5, but assume that the merge involves the forward and backward partial paths* $(0,0) = (1,1) - (2,1) \overset{3}{-} (2,2) - (4,2)$ *and* $(0,0) - (4,0)$*, i.e., the first (forward) partial path with label* $\mathcal{L}$ *ends at vertex* $(i^{tr}, i^{dr}) = (4,2)$*, and the second (backward) partial path with label* $\mathcal{L}'$ *ends at vertex* $(i^{tr}, j^{dr}) = (4,0)$*. Note that, compared to Example 5, the forward path is one arc longer but the backward path is two arcs shorter.*

Clearly, the merged truck path (in forward direction) is $(0,1,2,4,0')$*. The first partial path has two drone subpath* $\langle 1,3,2\rangle$ *and* $\langle 2,\cdot,\cdot\rangle$*, where the latter is incomplete. Likewise, the second partial path has an incomplete subpath* $\langle 0,\cdot,\cdot\rangle$*. The two incomplete subpaths can be combined into one subpath* $\langle 2,\cdot,0'\rangle$ *(by reversing* $\langle 0,\cdot,\cdot\rangle$*). For the drone subpath to be valid, a drone customer* k^* *needs to be chosen from the set* N *so that it does not conflict with the elementarity constraints.* □

The example shows that an appropriate drone customer k must be chosen when merging two partial paths with an incomplete (last) drone subpath. Let label $\mathcal{L} = (R)$ represent a partial path ending at $i \in W$ with $i^{\mathrm{tr}} \neq i^{\mathrm{dr}}$ and label $\mathcal{L}' = (R')$ be a partial path ending at $j \in W$ with $j^{\mathrm{tr}} = i^{\mathrm{tr}}$ and $j^{\mathrm{dr}} \neq i^{\mathrm{dr}}$. When merging $\mathcal{L}$ and $\mathcal{L}'$, the completed drone subpath is $\langle i^{\mathrm{dr}}, k, j^{\mathrm{dr}}\rangle$ for a customer $k \in N$ served on the additional drone arc. The reduced cost of the resulting route r is

$$\tilde{c}_r^{cost} = R^{rdc,cost} + R'^{\,rdc,cost} + \pi_{i^{\mathrm{tr}}} + c^{\mathrm{dr}}_{i^{\mathrm{dr}},k} + c^{\mathrm{dr}}_{k,j^{\mathrm{dr}}} - \pi_k \quad \text{or} \tag{4.8b}$$

$$\tilde{c}_r^{dur} = R^{rdc,dur} + R'^{\,rdc,dur} + \pi_{i^{\mathrm{tr}}} + \max\left\{R^{dur} + R'^{\,dur}, t^{\mathrm{dr}}_{i^{\mathrm{dr}},k} + t^{\mathrm{dr}}_{k,j^{\mathrm{dr}}}\right\} - \pi_k \tag{4.8c}$$

depending on the objective, i.e., either cost or duration minimization. The route is feasible if

$$R^{load} + R'^{\,load} + q_{i^{\mathrm{tr}}} + q_k \leq Q \tag{4.9a}$$

$$R^{cust,n} + R'^{\,cust,n} \leq \begin{cases} 1, & n \in N \setminus \{i^{\mathrm{tr}}, k\} \\ 0, & n = k \end{cases} \tag{4.9b}$$

Accordingly, to obtain a minimum reduced-cost route, the drone customer k^* is chosen as

$$\begin{aligned} k^* &= \underset{k \in N(R,R',i^{\mathrm{tr}})}{\arg\min} \left\{ c^{\mathrm{dr}}_{i^{\mathrm{dr}},k} + c^{\mathrm{dr}}_{k,j^{\mathrm{dr}}} - \pi_k \right\} && \text{or} \\ k^* &= \underset{k \in N(R,R',i^{\mathrm{tr}})}{\arg\min} \left\{ \max\{ R^{dur} + R'^{dur}, t^{\mathrm{dr}}_{i^{\mathrm{dr}},k} + t^{\mathrm{dr}}_{k,j^{\mathrm{dr}}} \} - \pi_k \right\} \end{aligned} \tag{4.10}$$

where the subset of the customers to choose from is defined as $N(R, R', i^{\mathrm{tr}}) = \{k \in N : (4.9) \text{ is fulfilled}\}$. For convenience, we assume that the minimum is ∞ for $N(R, R', i^{\mathrm{tr}}) = \varnothing$. All routes with $\tilde{c}_r \geq 0$ are rejected in the merge procedure.

Finally, to avoid the generation of the same route multiple times, we restrict the merge to the cases when either $R^{load} > Q/2$ or $R'^{load} > Q/2$ or $i^{\mathrm{tr}} = 0'$ holds true.

ng-Route Relaxation

The elementarity constraints are those constraints that make many SPPRCs not only theoretically but also practically difficult to solve. The use of an SPPRC relaxation offers an easier solution of the pricing problems but comes at the cost of a generally weaker linear relaxation of the corresponding MP. For the CVRP and the *VRP with time windows* (VRPTW), Baldacci *et al.* (2011) introduced the *ng*-route relaxations to gradually control the trade-off between the strength of the LP-relaxation and the practical difficulty of the SPPRC pricing problems. The concrete *ng*-route relaxation is defined by neighborhoods $N_i \subseteq N$, one for each vertex i. A cycle $(i, i_1, \ldots, i_\ell, i)$ over vertex i (with $\ell \geq 1$) is feasible in the relaxation, if at least one vertex i_p (for some $p \in \{1, 2, \ldots, \ell\}$) fulfills $i \notin N_{i_p}$.

The *ng*-route relaxation is not directly applicable in the artificial network of the VRP-D. We adapt it (similar as Roberti and Ruthmair (2021)) as follows:

- For each customer $v \in V$, we define a neighborhood $N_v \subseteq N$ based on the n_{ng} customers closest to v (the parameter n_{ng} controls the size of the neighborhoods and, herewith, the strength and difficulty of the relaxation).
- We equip each artificial vertex $(v, w) \in W$ with the neighborhood N_v, i.e., the service tasks in the neighborhood depend only on the truck vertex.
- In particular, for $i^{\mathrm{tr}} \in N$, the neighborhood $N_{(i^{\mathrm{tr}},i^{\mathrm{dr}})}$ contains customer i^{tr}.

The REF (4.4c) is altered by setting $R^{cust,n}_j = 0$ if $n \notin N_j$. The BPC implementation used in the computational experiments presented in Section 4.5 uses $n_{\mathrm{ng}} = 3$ and extends the neighborhoods dynamically (see below). Note that the merge conditions $R^{cust,n} + R'^{cust,n} \leq 1$ and (4.9b) remain valid.

Monodirectional (forward) labeling and bidirectional labeling have slightly different *ng*-route relaxations even if the neighborhoods are chosen identically: The monodirectional relaxation can be weaker. The effect is observed for routes with two drone subpaths serving the same drone customer k. Such a non-elementary

route can be feasible in the *ng*-route relaxation when labeling is performed completely in forward direction, but the same route can be excluded in the merge.

Example 8 *We consider a route with truck path* $(0, 1, 2, 3, 0')$ *and two drone subpaths* $\langle 0, 4, 2\rangle$ *and* $\langle 2, 4, 0'\rangle$*, i.e., the drone customer* $k = 4$ *is served twice. In the artificial network* $\mathscr{N} = (W, A)$*, the associated route is*

$$(0,0) - (1,0) - (2,0) \overset{4}{-} (2,2) - (3,2) - (0',2) \overset{4}{-} (0',0').$$

If the neighborhood of vertex 3 *or* $0'$ *does not include* $k = 4$*, this route is feasible in the forward labeling algorithm.*

For the bidirectional case, we assume that the merge happens at vertex $(i^{tr}, i^{dr}) = (2, 2)$ *and that vertex 2 has* $k = 4$ *included in its neighborhood. Then, the two partial paths are*

$$(0,0) - (1,0) - (2,0) \overset{4}{-} (2,2) \quad \textit{and} \quad (0,0) - (3,0) - (2,0) \overset{4}{-} (2,2)$$

are both ng-feasible with resource values $R^{cust,4} = R'^{cust,4} = 1$ *at the merge vertex* $(2, 2)$*. However, the merge condition for the case* $i^{tr} = i^{dr}$ *requires* (4.7b)*, i.e.,* $R^{cust,4} + R'^{cust,4} \leq 1$*, which is clearly violated. Hence, the considered route is excluded in the implicit bidirectional labeling.* □

Finally, note that inconsistent results would be observed if the half-way point would be chosen dynamically (Tilk *et al.*, 2017). However, implicit bidirectional labeling requires a fixed half-way point exactly in the middle at $Q/2$.

Dynamic Neighborhood Extension (DNE) DNE is a technique to strengthen *ng*-route relaxations, which has been independently proposed by Roberti and Mingozzi (2014) and Bode and Irnich (2015). Neighborhoods N_i are extended depending on the solutions obtained in the MP. We adapt DNE as follows: When the root node of the branch-and-bound tree is solved, we check whether routes of the MP solution serve customers more than once (each double service creates a *task cycle*, see also Irnich and Desaulniers, 2005, Sect. 2.2). For each elementary task cycle, we determine its flow value, i.e., the sum of the λ_r^*-values over all routes r that contain the task cycle. We then choose the customer w^* and the task cycle Z^* that maximize the flow value divided by the length of the task cycle. For this cycle $Z^* = (w^*, v_1^*, \ldots, v_\ell^*, w^*)$, the *ng*-neighborhoods $N_{v_1^*}, \ldots, N_{v_\ell^*}$ are extended by w^*. This eliminates the task cycle Z^* in subsequent solutions of the pricing problem. Furthermore, the corresponding routes have to be removed from the RMP. After extending the neighborhood, the root node is solved again, and the check for task cycles is repeated for the resulting solution. We stop this procedure when all routes are elementary or 50 task cycles have been eliminated.

Acceleration Techniques

We now describe exact and heuristic acceleration techniques for the column-generation process.

Acceleration of the Merge Procedure The merge procedure is the bottleneck of the column-generation process. In comparison to classical VRPs (like CVRP, VRPTW, etc.), the artificial network contains a quadratic number $\mathcal{O}(m^2)$ of vertices, and for those vertices i with $i^{\text{tr}} \neq i^{\text{dr}}$ the merge must be performed 'over arcs'. All these arcs are drone arcs $[i, j, k]$ with $i^{\text{tr}} = j^{\text{tr}}$, $i^{\text{dr}} \neq j^{\text{dr}}$ and $k \in N$. As a result, the number of arcs to consider in the merge is $\mathcal{O}(m^4)$. This explains the empirical observation that the merge is very time consuming. Therefore, we stop the merge procedure as soon as 100 negative reduced-cost routes are found (we use the same termination condition in the forward labeling algorithm).

To reduce the computational effort of the merge procedure further, we apply two exact acceleration techniques: First, when the forward labeling process is finished, all labels $\mathcal{L}$ belonging to the same vertex $(i^{\text{tr}}, i^{\text{dr}})$ are sorted in non-decreasing order by their load resource R^{load}. If truck and drone are at the same position, i.e., both forward and backward label belong to a vertex i with $i^{\text{tr}} = i^{\text{dr}}$, the capacity constraint $R^{load} + R'^{load} + q_{i^{\text{tr}}} \leq Q$ can be exploited as follows. In two nested loops running over candidate labels R and R' with $R^{load} \leq R'^{load}$, the inner loop for R' can be discontinued as soon as $R^{load} + R'^{load} + q_{i^{\text{tr}}} > Q$. Moreover, the outer loop for R can be discontinued as soon as $2R^{load} + q_{i^{\text{tr}}} > Q$. If truck and drone are at different positions, then we know that the backward label must belong to another vertex j with $j^{\text{tr}} = i^{\text{tr}}$ and $j^{\text{dr}} \neq i^{\text{dr}}$. In this case, the capacity constraint can be further exploited: The load onboard is not smaller than $R^{load} + R'^{load} + q_{i^{\text{tr}}} + \min_{k \in N}\{q_k\}$. Hence, when considering the two nested loops, the outer over labels R at vertex $(i^{\text{tr}}, i^{\text{dr}})$ and the inner over labels R' at vertex $(i^{\text{tr}}, j^{\text{dr}})$, the inner loop can be discontinued if the above value exceeds Q.

Second, the determination of a best drone customer in (4.10) is another time-consuming task that has to be done numerous times in the merge procedure. To reduce the computational burden, we precompute the reduced cost for each possible drone subpath $\langle i^{\text{dr}}, k, j^{\text{dr}} \rangle$ before the actual procedure starts. For each combination $(i^{\text{dr}}, j^{\text{dr}}) \in V \times V$, the reduced cost of the drone subpath (i.e., $c^{\text{dr}}_{i^{\text{dr}},k} + c^{\text{dr}}_{k,j^{\text{dr}}} - \pi_k$ or $t^{\text{dr}}_{i^{\text{dr}},k} + t^{\text{dr}}_{k,j^{\text{dr}}} - \pi_k$) is stored in a list together with the respective drone customer k. The list is then sorted in non-decreasing order by reduced cost. When using the cost objective, a drone customer $k \in N(R, R', i^{\text{tr}}) \subset N$ for a merge with minimum reduced cost is found by inspecting the sorted list until a feasible merge is found (one that fulfills (4.9)). Moreover, the loop over $k \in N(R, R', i^{\text{tr}})$ can be stopped as soon as reduced cost can be estimated being non-negative. When using the duration objective, however, the reduced cost of the merge also depends on

the sum $R^{dur} + R'^{dur}$ and we cannot determine a merge with minimum reduced cost as the first one in the sorted list that is feasible. Moreover, the loop over $k \in N(R, R', i^{\text{tr}})$ cannot be stopped prematurely. However, inspecting potential drone customers in sorted ordering helps to find a negative reduced-cost merge sooner which accelerates the pricing also slightly in this case.

Partial Pricing with Reduced Networks We use a hierarchy of heuristics to quickly identify negative reduced-cost routes. The use of one or more heuristics is also known as *partial pricing* (Gamache *et al.*, 1999). In the context of dynamic programming-based labeling, partial pricing results from using the bidirectional labeling algorithm applied to an appropriate subgraph of the artificial network $\mathscr{N} = (W, A)$. For the VRP-D, we use two reduced networks. The first has up to four ingoing and four outgoing truck arcs ($A^{alone} \cup A^{tgthr}$) as well as at most two drone arcs (A^{drone}) per artificial vertex. The second reduced network doubles the number of arcs per vertex (eight and four). To build the networks, the arcs with the smallest reduced cost in the current pricing iteration are selected. Pricing always starts with the first network, and the pricing problem is solved exactly only when both heuristics fail to provide any routes with negative reduced cost.

4.4.3 Valid Inequalities

We strengthen the linear relaxation of formulation (4.2) with the help of *subset-row inequalities* (SRIs, Jepsen *et al.*, 2008) and non-robust *capacity cuts* (CCs, Baldacci *et al.*, 2008).

A SRI is defined for a subset $S \subset N$ of customers and non-negative weights for each $v \in S$. We restrict ourselves to subsets of size three and weight $1/2$, because these can be easily separated by straightforward enumeration and handled with a single binary attribute per active SRI in the labeling. The valid inequality reads:

$$\sum_{r \in \Omega} \left\lfloor \sum_{v \in S} \frac{b_{vr}}{2} \right\rfloor \lambda_r \leq 1 \tag{4.11}$$

A CC is also defined for a subset $C \subset N$ of customers. Let K_C be a lower bound on the number of routes needed to serve C. The valid inequality for (C, K_C) is

$$\sum_{r \in \Omega} \max_{v \in C} \{\min\{1, b_{vr}\}\} \lambda_r \geq K_C, \tag{4.12}$$

where we use the easy-to-compute lower bound $K_C = \lceil \sum_{v \in C} q_v / Q \rceil$ (the exact solution of a bin-packing problem like in Section 4.4.1 would sometimes create

stronger inequalities at the cost of a much more time consuming separation procedure).

Let $\mathcal{S}$ and $\mathcal{C}$ denote the sets of active SRIs and CCs, respectively. Further, let $\sigma_S < 0$ be the dual price of the SRI defined by $S \in \mathcal{S}$ and let $\tau_C > 0$ be the dual price of the CC defined by $(C, K_C) \in \mathcal{C}$. We define the additional binary resources $(R^{sri,S})_{S\in\mathcal{S}}$ and $(R^{cc,C})_{(C,K_C)\in\mathcal{C}}$. Initially, they are set to $(R^{sri,S}) = \mathbf{0}$ and $(R^{cc,C}) = \mathbf{0}$. For every second time a partial path serves a customer in S, the dual price σ_S has to be subtracted from the reduced cost. Similarly, the first time when a partial path serves a customer in C, the dual price τ_C has to be subtracted from the reduced cost. As for the load resource, it is important for a correct bidirectional labeling to delay the manipulation of the resource values when propagating over an arc $a = [i, j, k] \in A$. This is modeled in the following way:

$$R_j^{rdc,cost} = \begin{cases} R_i^{rdc,cost} + c_{i^{\mathrm{tr}},j^{\mathrm{tr}}} - \pi_{j^{\mathrm{tr}}} - \sum\limits_{\substack{S\in\mathcal{S}:R_i^{sri,S}=1,\\ i^{\mathrm{tr}}\in S}} \sigma_S - \sum\limits_{\substack{C\in\mathcal{C}:R_i^{cc,C}=0,\\ i^{\mathrm{tr}}\in C}} \tau_C, \\ \qquad\qquad \text{if } [i,j,\perp] \in A^{alone} \cup A^{tgthr} \\ R_i^{rdc,cost} + c^{\mathrm{dr}}_{i^{\mathrm{dr}},k} + c^{\mathrm{dr}}_{k,j^{\mathrm{dr}}} - \pi_k - \sum\limits_{\substack{S\in\mathcal{S}:R_i^{sri,S}=1,\\ k\in S}} \sigma_S - \sum\limits_{\substack{C\in\mathcal{C}:R_i^{cc,C}=0,\\ k\in C}} \tau_C, \\ \qquad\qquad \text{if } [i,j,k] \in A^{drone} \end{cases} \tag{4.4a'}$$

$$R_j^{sri,S} = \begin{cases} 1 - R_i^{sri,S}, & \text{if } ([i,j,\perp] \in A^{alone} \cup A^{tgthr} \text{ and } i^{\mathrm{tr}} \in S) \text{ or} \\ & \qquad ([i,j,k] \in A^{drone} \text{ and } k \in S) \\ R_i^{sri,S}, & \text{otherwise} \end{cases}$$

$$R_j^{cc,C} = \begin{cases} 1, & \text{if } ([i,j,\perp] \in A^{alone} \cup A^{tgthr} \text{ and } i^{\mathrm{tr}} \in C) \text{ or} \\ & \qquad ([i,j,k] \in A^{drone} \text{ and } k \in C) \\ R_i^{cc,C}, & \text{otherwise} \end{cases}$$

The new formula (4.4a') for the update of the reduced cost replaces the original

REF (4.4a) for the cost-minimization objective. Likewise,

$$R_j^{rdc,dur} = \begin{cases} R_i^{rdc,dur} - \pi_{j^{\mathrm{tr}}} - \sum\limits_{\substack{S\in\mathcal{S}: R_i^{sri,S}=1,\\ i^{\mathrm{tr}}\in S}} \sigma_S - \sum\limits_{\substack{C\in\mathcal{C}: R_i^{cc,C}=0,\\ i^{\mathrm{tr}}\in C}} \tau_C, & \text{if } [i,j,\perp]\in A^{alone} \\ R_i^{rdc,dur} + t_{i^{\mathrm{tr}},j^{\mathrm{tr}}} - \pi_{j^{\mathrm{tr}}} - \sum\limits_{\substack{S\in\mathcal{S}: R_i^{sri,S}=1,\\ i^{\mathrm{tr}}\in S}} \sigma_S - \sum\limits_{\substack{C\in\mathcal{C}: R_i^{cc,C}=0,\\ i^{\mathrm{tr}}\in C}} \tau_C, & \\ & \text{if } [i,j,\perp]\in A^{tgthr} \\ R_i^{rdc,dur} + \max\{t^{\mathrm{dr}}_{i^{\mathrm{dr}},k} + t^{\mathrm{dr}}_{k,j^{\mathrm{dr}}}, R_i^{dur}\} - \pi_k & \\ \qquad - \sum\limits_{\substack{S\in\mathcal{S}: R_i^{sri,S}=1,\\ k\in S}} \sigma_S - \sum\limits_{\substack{C\in\mathcal{C}: R_i^{cc,C}=0,\\ k\in C}} \tau_C, & \text{if } [i,j,k]\in A^{drone} \end{cases} \tag{4.6a'}$$

replaces the original REF (4.6a) for the duration objective. There are no feasibility constraints associated with new resources. The dominance Rules 1 and 2 have to be modified in the standard way as explained in (Jepsen *et al.*, 2008) and (Baldacci *et al.*, 2008). Moreover, in the merge procedure, reduced cost computations (4.8a) and (4.8b) have to be altered. For the merge of two labels R and R' at the same vertex i with $i^{\mathrm{tr}} = i^{\mathrm{dr}}$, the reduced cost becomes $\tilde{c}_r^{cost} = R^{rdc,cost} + R'^{rdc,cost} + \pi_{i^{\mathrm{tr}}} + (*)$ and $\tilde{c}_r^{dur} = R^{rdc,dur} + R'^{rdc,dur} + \pi_{i^{\mathrm{tr}}} + (*)$ where

$$(*) = \sum_{\substack{C\in\mathcal{C}:\\ R^{cc,C}=R'^{cc,C}=1}} \tau_C - \sum_{\substack{S\in\mathcal{S}:\\ R^{sri,S}=R'^{sri,S}=1}} \sigma_S - \sum_{\substack{S\in\mathcal{S}: i\in S\\ R^{sri,S}+R'^{sri,S}=1}} \sigma_S - \sum_{\substack{C\in\mathcal{C}: i\in C\\ R^{cc,C}=R'^{cc,C}=0}} \tau_C$$

(the first sum corrects dual prices of CCs whose customers have been visited in both partial paths, the second sum incorporates the dual prices of SRIs with an odd number of visits in both partial paths, the third incorporates dual prices of SRIs with $i \in S$, and the last sum incorporates CCs containing customer i that were not visited by poth partial paths).

For the merge of a label R at vertex i with $i^{\mathrm{tr}} \neq i^{\mathrm{dr}}$ and a label R' at vertex j with identical truck vertex $j^{\mathrm{tr}} = i^{\mathrm{tr}}$ but different drone vertex $j^{\mathrm{dr}} \neq j^{\mathrm{tr}}$ serving customer $k \in N$ on the drone arc, the reduced cost becomes

$$\begin{aligned} \tilde{c}_r^{cost} \;=\; & R^{rdc,cost} + R'^{rdc,cost} + \pi_{i^{\mathrm{tr}}} + c^{\mathrm{dr}}_{i^{\mathrm{dr}},k} + c^{\mathrm{dr}}_{k,j^{\mathrm{dr}}} - \pi_k \\ & - \sum_{\substack{S\in\mathcal{S}: k\in S, i^{\mathrm{tr}}\notin S\\ R^{sri,S}+R'^{sri,S}=1}} \sigma_S - \sum_{\substack{S\in\mathcal{S}: k\in S, i^{\mathrm{tr}}\in S\\ R^{sri,S}=R'^{sri,S}=0}} \sigma_S - \sum_{\substack{C\in\mathcal{C}: k\in C, i^{\mathrm{tr}}\notin S\\ R^{cc,C}=R'^{cc,C}=0}} \tau_C \;+\; (*) \end{aligned}$$

in case of the cost-based objective and

$$\begin{aligned} \tilde{c}_r^{dur} \;=\; & R^{rdc,dur} + R'^{rdc,dur} + \max\left\{R^{dur} + R'^{dur}, t^{\mathrm{dr}}_{i^{\mathrm{dr}},k} + t^{\mathrm{dr}}_{k,j^{\mathrm{dr}}}\right\} + \pi_{i^{\mathrm{tr}}} - \pi_k \\ & - \sum_{\substack{S\in\mathcal{S}: k\in S, i^{\mathrm{tr}}\notin S\\ R^{sri,S}+R'^{sri,S}=1}} \sigma_S - \sum_{\substack{S\in\mathcal{S}: k\in S, i^{\mathrm{tr}}\in S\\ R^{sri,S}=R'^{sri,S}=0}} \sigma_S - \sum_{\substack{C\in\mathcal{C}: k\in C, i^{\mathrm{tr}}\notin S\\ R^{cc,C}=R'^{cc,C}=0}} \tau_C \;+\; (*) \end{aligned}$$

in case of the duration objective. Note that CCs have a non-positive dual price τ_C so that they can lower the reduced cost when added in (*). Therefore, the acceleration of the merge procedure by stopping the loop over $k \in N(R, R', i^{\text{tr}})$ prematurely (see Section 4.4.2) must be altered: Only if the reduced cost plus the sum $\sum_C \tau_C$ is non-negative, the loop can be stopped prematurely. This is a rather weak condition.

Violated SRIs can be identified with a straightforward approach that enumerates all subsets S with $|S| = 3$ and then determines the left-hand side of (4.11), checks whether it is greater than 1, and stores S together with the degree of violation. Separating violated CCs requires more computational effort. For the capacitated arc routing problem, Martinelli *et al.* (2013) introduced a MIP-based CC separation algorithm that simultaneously computes C and K_C (for the above lower bound). The algorithm is directly applicable here.

Pretests have shown that adding many SRIs makes the labeling algorithm considerably more difficult. In contrast, the labeling algorithm can cope much better with CCs. Therefore, we always start with solving the separation problem for CCs and search for violated SRIs only if no CCs have been added. Moreover, we limit the total number of violated SRIs to be added to a maximum of 10 inequalities in total. Among the most violated inequalities, we allow no more than five per round and limit the number of SRIs per customer to not more than three. Violated inequalities are added only at the root node and we allow up to 100 CCs in total.

4.4.4 Branching

Let (λ_r^*, f^*) be a solution of the RMP. Whenever some component of (λ_r^*, f^*) is fractional, branching is necessary to obtain an integer solution. We apply the following four-level hierarchical branching scheme: (1) total number of routes, (2) number of drone visits at each customer, (3) flow of trucks on customer-to-customer arcs, and (4) flow on drone subpaths. At each branch-and-bound node, all routes that are incompatible with the current branching decision are temporarily removed from the RMP. The branch-and-bound tree is explored using a best-first search. Note that all constraints added to the RMP enforce additional dual prices in the labeling algorithm that all can be added on the corresponding arcs. We describe the four levels in detail now.

At the first level, when f^* is fractional, we branch on the total number of routes by creating two branches defined by $f \leq \lfloor f^* \rfloor$ and $f \geq \lceil f^* \rceil$, which is implemented by changing the bounds in (4.2d).

At the second level, we branch on the number of drone services performed for each customer (as proposed by Roberti and Ruthmair (2021) as a first-level branching decision in the algorithms for the TSP-D). To this end, let y_{kr} be the number of drone services performed by route r at customer k, which is the number of drone

arcs of type $[\cdot, \cdot, k]$ used in the artificial network. If $y_k^* = \sum_{r \in \Omega} y_{kr} \lambda_r^*$ is fractional for some customers $k \in N$, we choose a customer k^* with value $y_{k^*}^* - \lfloor y_{k^*}^* \rfloor$ closest to 0.5 and branch on $y_{k^*} = 0$ and $y_{k^*} = 1$. Instead of adding a constraint to the RMP, these decisions can be enforced directly on the artificial network: The zero-branch results from eliminating all drone arcs of type $[\cdot, \cdot, k]$ serving customer k^*. Moreover, k^* must be removed from the sets $N(R, R', i^{\text{tr}})$ that are used in the merge procedure. The one-branch results from eliminating all vertices representing that customer k^* is served by a truck, i.e., all vertices $(i^{\text{tr}}, i^{\text{dr}}) \in W$ with $i^{\text{tr}} = k^*$ or $i^{\text{dr}} = k^*$.

To finally ensure integrality, we use a similar branching strategy as proposed in Roberti and Ruthmair (2021), i.e., we branch on the total flow induced by truck movements between the two customers and the total flow on drone subpaths, respectively. However, instead of branching on the directed flow (as proposed in (Roberti and Ruthmair, 2021)), we branch on the total flow value.

At the third level, we consider pairs of customers $v, w \in N, v \neq w$ and the total flow induced by truck movements between the two customers. Formally, we consider the following subset of together arcs and alone arcs

$$A_{vw}^{\text{tr}} = \{[(v, v), (w, w), \perp], [(w, w), (v, v), \perp]\} \cup$$

$$\left\{[i, j, \perp], [j, i, \perp] \in A^{alone} : i^{\text{tr}} = v, j^{\text{tr}} = w, \text{and } i^{\text{dr}} = j^{\text{dr}} \in V\right\}$$

and determine the values $e_{vw}^* = \sum_{r \in \Omega} \sum_{a \in A_{vw}^{\text{tr}}} e_{ar} \lambda_r^*$ for each customer pair of v and w, where e_{ar} is the number of times that route r uses arc $a \in A$. Note that the symmetry in the definition of A_{vw}^{tr} implies $e_{wv}^* = e_{vw}^*$ for all $v, w \in N, v \neq w$. If at least one of these values is fractional, we choose a customer pair v^* and w^* with value $e_{v^*,w^*}^* - \lfloor e_{v^*,w^*}^* \rfloor$ closest to 0.5 and create the two branches $\sum_{r \in \Omega} \sum_{a \in A_{v^*,w^*}^{\text{tr}}} e_{ar} \lambda_r = 0$ and $\sum_{r \in \Omega} \sum_{a \in A_{v^*,w^*}^{\text{tr}}} e_{ar} \lambda_r = 1$. As in the second level, the zero-branch can be directly enforced by removing all arcs A_{v^*,w^*}^{tr} from the artificial network. The one-branch needs to be enforced by adding the above constraint to the RMP, in the same spirit as the one-branch needs to be enforced by an additional constraints when branching on undirected edges in other VRP variants.

In addition, it is feasible and advantageous to reduce the artificial network by removing further arcs that are incompatible with the branching constraint. In particular, the enforced truck customers v^* and w^* must never be served by a drone so that the arcs of type $[\cdot, \cdot, v^*] \in A^{drone}$ and $[\cdot, \cdot, w^*] \in A^{drone}$ can be eliminated. Likewise, v^* and w^* are disregarded as drone customers in the merge procedure. Branching decisions on the third level finally ensure unique truck paths. However, drone subpaths can still be fractional.

Hence, at the fourth level, we consider the total flow on drone subpaths for branching. Recall that any drone subpath $\langle v, k, w \rangle$ has a symmetric counter-

part $\langle w, k, v\rangle$, which are uniquely determined by the two drone arcs $[(w, v), (w, w), k]$ and $[(v, w), (v, v), k] \in A^{drone}$. We define

$$A^{\mathrm{dr}}_{\langle v,k,w\rangle} = \{[(w, v), (w, w), k], [(v, w), (v, v), k]\}$$

and determine the values $e^*_{\langle v,k,w\rangle} = \sum_{r\in\Omega}\sum_{a\in A^{\mathrm{dr}}_{\langle v,k,w\rangle}} e_{ar}\lambda^*_r$ for each drone subpath $\langle v, k, w\rangle$ with $v \neq k \neq w, v \neq w$. Note that the symmetry in the definition of $A^{\mathrm{dr}}_{\langle v,k,w\rangle}$ implies $e^*_{\langle v,k,w\rangle} = e^*_{\langle w,k,v\rangle}$. We apply a similar strategy as at the third level: For a subpath $\langle v^*, k^*, w^*\rangle$ with value $e^{\mathrm{dr}}_{\langle v^*,k^*,w^*\rangle} - \lfloor e^{\mathrm{dr}}_{\langle v^*,k^*,w^*\rangle}\rfloor$ closest to 0.5, two branches defined by $\sum_{r\in\Omega}\sum_{a\in A^{\mathrm{dr}}_{\langle v^*,k^*,w^*\rangle}} e_{ar}\lambda_r = 0$ and $\sum_{r\in\Omega}\sum_{a\in A^{\mathrm{dr}}_{\langle v^*,k^*,w^*\rangle}} e_{ar}\lambda_r = 1$, resp., are created. Again, the zero-branch results from removing the arcs $A^{\mathrm{dr}}_{\langle v^*,k^*,w^*\rangle}$ from the artificial network. The one-branch is enforced by adding the corresponding constraint to the RMP. As before, the artificial network can be reduced by eliminating arcs that are inconsistent with this branching decision: The vertices $(i^{\mathrm{tr}}, i^{\mathrm{dr}}) \in W$ with $i^{\mathrm{tr}} = k^*$ or $i^{\mathrm{dr}} = k^*$ can be removed because the customer k^* is not served by truck . Further, all drone arcs of type $[\cdot, \cdot, v^*]$ and $[\cdot, \cdot, w^*]$ can be eliminated because the drone never serves the truck customers v^* and w^*. Finally, v^* and w^* must be disregarded as drone customers in the merge procedure.

The branching decisions at the third and fourth level imply unique truck paths and drone subpaths. Since a solution is fully determined by both types of decisions, this four-level branching scheme is complete.

4.5 Computational Results

The following experiments were conducted on HPC cluster MOGON 2 of Johannes Gutenberg University Mainz using nodes equipped with Intel Xeon E5-2630v4 (Broadwell) processors running at 2.2GHz. Notice that the performance of a single thread of the cluster is slightly worse compared to that of a standard desktop processor. The BPC algorithm is coded in `C++` and compiled with GCC 11.2.0 in release mode into a 64-bit single-thread executable. The callable library of CPLEX 20.1.0 is used for the (re)optimization of the RMPs, for solving a primal MIP-based heuristic, and for separating CCs. CPLEX's default parameters are kept, except for setting the number of threads to one. The MIP-based heuristic solves a restricted integer MP using all columns generated up to this point. It is applied at the first and second level of the branch-and-bound-tree as well as if the BPC algorithm terminates without finding a feasible solution. The BPC computation time is limited to 3600 seconds per instance and additional 60 seconds are granted for the MIP-based heuristic at the end.

In what follows, Section 4.5.1 describes the VRP-D instances, Section 4.5.2 analyzes the impact of the acceleration techniques for the merge, Section 4.5.3 com-

pares the pure forward with the new implicit bidirectional labeling algorithm, and Section 4.5.4 analyzes what effects result from different routing costs and speeds of the drones. Pointers to further results that are presented in the eCompanion are given in Section 4.5.5.

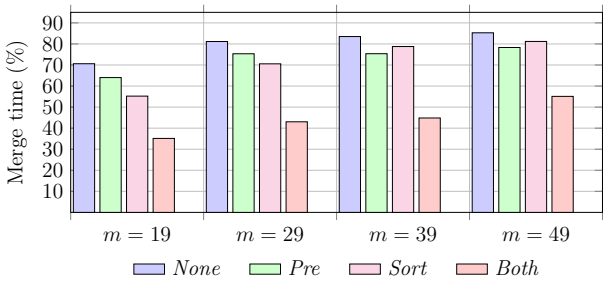
4.5.1 VRP-D Instances

Up to date, no suitable VRP-D benchmark set has been made available. Therefore, we created new VRP-D instances based on the *Capacitated Vehicle Routing Problem Library* (CVRPLIB) publicly available at `http://vrp.atd-lab.inf.puc-rio.br/index.php/en/` to properly evaluate the performance of our BPC algorithm. We use CVRPLIB instances from *set A* and *set B* (proposed by Augerat *et al.*, 1995) and consider the first 20, 30, 40, or 50 vertices, respectively. The first vertex always represents the depot. As a result, instances have $m = 19$, 29, 39, or 49 customers. For each size m, we construct 20 instances so that the VRP-D instance set comprises 80 instances.

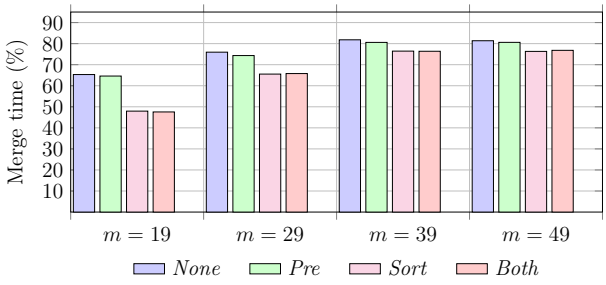
VRP-D-specific data is determined in the following way: All customers with a demand greater than $\bar{q} = Q/5$ are truck-only customers. The share of truck-only customers varies between 10% and 33% with an average of 20% over all instances. As a sidenote, if the value is kept in that interval, we could not see predictable differences in the difficulty of the instances. This statement can be proven with the instance-specific results presented in Tables 4.8–4.13 in the Appendix. The literature stresses that routing costs and travel times are lower for drones than for trucks. Hence, we compute c_{ij} and t_{ij} for trucks as Manhattan distances. In contrast, c_{ij}^{dr} and t_{ij}^{dr} for drones are computed as Euclidean distances divided by the given drone speed factor β. We set $\beta = 3$ as the default value and analyze the impact of the drones' speed in Section 4.5.4 using the alternative factors $\beta = 1$ and $\beta = 5$. All instances with details on how costs and times are computed can be found at `https://logistik.bwl.uni-mainz.de/research/benchmarks/`.

4.5.2 Acceleration of the Merge Procedure

When using implicit bidirectional labeling, the merge procedure is the bottleneck operation of the entire BPC algorithm. Therefore, we first examine the effect of the acceleration techniques presented in Section 4.4.2 on the time spent in the merge procedure. The following four algorithmic setups are considered: (*None*) no acceleration technique is used for the merge, (*Pre*) the reduced cost of each possible drone subpath is precomputed, (*Sort*) all labels belonging to the same vertex are sorted by their load resource, and (*Both*) both acceleration techniques are combined.



(a) Cost Objective.



(b) Duration Objective.

Figure 4.3: Average share (in percent) of the total computation time spent in the merge procedure for different acceleration techniques.

Figure 4.3 shows the percentage of the total BPC computation time spent on the merge operation depending on which of the above acceleration techniques is used. The results are aggregated over instances of the same size. For all setups, the share of the merge time increases with the size m of the instances. Similar values are obtained for both objectives and setup *None* where the merge accounts for 65 to 85% of the total BPC runtime.

The results differ significantly by objective, with a relatively strong positive effect for cost minimization (Figure 4.3a) and a moderate effect for duration minimization (Figure 4.3b). The precomputation of the reduced cost of drone subpaths (*Pre*) helps to reduce the merge time by about seven percentage points on average only for cost minimization, while the reduction is negligible for duration minimization. The latter can be explained by the fact that prematurely stopping the loop over the drone customers $k \in N(R, R', i^{\text{tr}})$ for the merge at pairs of vertices where truck and drone are at different locations is not possible in case of duration minimization. The effect of sorting (*Sort*) the labels by their load resource is beneficial for both objectives (more so for duration than for cost minimization), as it reduces the share by about 8.7 and 9.6 percentage points, respectively, compared to setup *None*. However, even though the absolute savings are greater for *Sort* than for *Pre*, the respective savings in relative computation times decrease with

the size m of the instances. The combination of both techniques (*Both*) reduces the merge times considerably: compared to setup *None*, they are halved in the best case of cost minimization and $m = 19$. In all the following experiments, the setup *Both* becomes the default.

4.5.3 Comparison of Forward and Bidirectional Labeling

We perform two types of computational experiments to compare the performance of the forward and the implicit bidirectional labeling strategies in the BPC algorithm. The first experiment focuses on quantifying their computational effort when applied to completely identical instances of the pricing problem. The second experiment then examines the impact on independent BPC runs.

In the first analysis, we consider the root node, i.e., the linear relaxation of the MP including the strengthening with the dynamic extension of the ng-neighborhoods, SRIs, and CCs. In each pricing iteration, the SPPRC subproblem is solved using both labeling strategies (forward and implicit bidirectional) applying the labeling algorithms consecutively. Since the dual prices and reduced costs used in both labeling algorithms are identical, a perfectly fair comparison is ensured. We add the negative reduced cost routes generated by the bidirectional algorithm to the RMP (this choice is arbitrary). As performance measures to compare the implicit bidirectional and forward labeling, we use the ratios of (1) the number of generated labels, (2) the number of processed labels, and (3) the total pricing time in the two labeling algorithms. The ratios are aggregated by taking the geometric mean over all pricing iterations of a VRP-D instance. Whenever the linear relaxation is not completely solved within the time limit, we omit the last iteration for both strategies. Table 4.1 reports the minimum (*min*), geometric mean (*mean*), and maximum (*max*) of each measure, aggregated over VRP-D instances with an identical number of customers. Additionally, the table lists the arithmetic mean (*avg*) of the number of labeling algorithms called while solving the LP-relaxation of the MP.

All *mean* and *avg* ratios presented in the table are less than one, which shows the implicit bidirectional labeling strategy is superior to its monodirectional counterpart. The number of calls is sufficiently large to provide statistically significant results. The number of generated (processed) labels reduces to 29% (23%) on average, i.e., by more than a factor of three (four). These reductions do not fully translate to the time consumed by the labeling algorithms, which is reduced to 58% on average.

In the second analysis, we compare the two fully-fledged BPC algorithms described in Section 4.4, where one uses the forward labeling and the other the implicit bidirectional labeling algorithm. Detailed results for each instance can be found in Tables 4.8–4.13 in the Appendix. Table 4.2 aggregates these re-

		Ratio of											
		generated labels			processed labels			pricing time			Num. calls		
Objective	m	min	mean	max	min	mean	max	min	mean	max	min	avg	max
Cost	19	0.29	0.32	0.36	0.22	0.26	0.30	0.39	0.44	0.56	46	211.6	336
	29	0.27	0.29	0.33	0.21	0.22	0.29	0.36	0.43	0.58	149	390.1	513
	39	0.26	0.27	0.32	0.19	0.21	0.25	0.33	0.45	0.73	385	471.0	549
	49	0.24	0.26	0.30	0.18	0.21	0.25	0.32	0.46	0.60	428	517.3	579
Subtotal			0.29			0.22			0.45			397.5	
Duration	19	0.28	0.32	0.36	0.21	0.25	0.31	0.40	0.57	0.70	108	292.2	401
	29	0.25	0.29	0.33	0.19	0.22	0.27	0.43	0.67	1.16	246	376.2	471
	39	0.24	0.28	0.34	0.18	0.22	0.28	0.47	0.82	1.46	338	448.4	556
	49	0.21	0.26	0.32	0.16	0.20	0.26	0.34	0.75	1.20	203	371.9	495
Subtotal			0.29			0.23			0.70			372.2	
Total			0.29			0.23			0.58			384.9	

Table 4.1: Comparison of forward and implicit bidirectional labeling using identical dual prices.

sults obtained with both BPC algorithms, again grouped by VRP-D instances with an identical number m of customers. The column entries have the following meaning: (#Inst) number of instances per group, (#Opt) number of instances solved to proven optimality within the time limit, (Gap_{LP}) average LP gap ($100 \cdot (opt - z_{LP})/z_{LP}$) in percent, ($\text{Gap}_{\text{cut}}$) average LP gap after extending the ng-neighborhoods and adding cutting planes in percent, (Gap) average relative difference between the upper bound (UB) and lower bound (LB) in percent at termination, i.e., $100 \cdot (UB - LB)/LB$ (computed only when an upper and lower bound was found), (Time) average solution time in seconds, and (#BB) average number of branch-and-bound nodes solved per VRP-D instance. Note that *Time* includes unsolved instances with 3600 seconds for the BPC and extra time for the MIP-based heuristic (see first paragraph of Section 4.5). For both Gap_{LP} and Gap_{cut}, we only consider instances with a known optimal solution *opt*.

Objective		Forward labeling						Implicit bidirectional labeling					
	m	#Opt	Gap_{LP}	Gap_{cut}	Gap	Time	#BB	#Opt	Gap_{LP}	Gap_{cut}	Gap	Time	#BB
Cost	19	20	3.19	0.27	0.00	206.4	42.9	20	2.92	0.30	0.00	22.3	37.7
	29	16	3.67	0.56	0.11	1,454.0	90.3	18	3.43	0.62	0.02	814.8	163.4
	39	11	2.76	0.22	0.84	2,638.1	84.0	14	2.59	0.26	0.16	1,634.5	141.1
	49	3	2.31	0.04	24.49	3,406.5	54.7	3	2.28	0.06	3.34	3,161.5	84.9
Subtotal		50	3.19	0.34	6.36	1,926.3	67.9	55	2.97	0.38	0.88	1,408.3	106.8
Duration	19	18	3.70	1.24	0.06	726.2	155.8	19	3.59	1.42	0.01	335.7	232.7
	29	7	2.14	0.96	3.52	2,900.3	142.4	11	2.11	1.00	1.27	2,486.0	325.6
	39	0	—	—	14.04	3,602.1	53.6	0	—	—	2.29	3,640.9	95.4
	49	0	—	—	37.56	3,601.7	13.8	0	—	—	5.92	3,655.2	33.3
Subtotal		25	3.13	1.14	12.54	2,707.6	91.4	30	3.04	1.26	2.13	2,529.5	171.7
Total		75			10.08	2,316.9	79.7	85			1.63	1,968.9	139.2

Table 4.2: Comparison of two BPC algorithms equipped with a forward or implicit bidirectional labeling algorithm.

The results of the second analysis are very consistent over different instance sizes. For all but six instances with duration objective, the BPC algorithm determines a feasible solution and herewith an upper bound UB. The implicit bidirectional labeling algorithm outperforms the forward labeling algorithm, i.e., more instances are solved to optimality (85 versus 75), average gaps and computation times are smaller, and more branch-and-bound nodes are processed (in shorter time). In total, the BPC algorithm with implicit bidirectional labeling is on average around 15% faster than the BPC with a forward labeling. The speedup for the smallest instances with $m = 19$ customers is more than by a factor of 9. The results for the instances with more customers are biased due to the one-hour time limit. Hence, the implicit bidirectional labeling is even more valuable than the 15% speedup suggests: For instances with $m = 29$ (39) customers, the number of proven optimal solutions for both objectives increases from 23 to 29 (11 to 14), and for the largest instances with $m = 49$ customers, approximately 75% more branch-and-bound nodes can be processed reducing the average gap from 10.08 to 1.63%. The results also confirm empirically that the same ng-route relaxation may have a stronger LP-relaxation when bidirectional labeling (instead of forward) is applied (see Example 8), i.e, Gap_{LP} is slightly smaller for the bidirectional labeling. To be more precise, for the cost (duration) objective 52 (22) out of 55 (30) instances solved to optimality have a stronger LP-relaxation. All experiments conducted in the following use the implicit bidirectional labeling algorithm to solve the pricing subproblems.

4.5.4 Effect of the Drones' Routing Cost and Speed

The routing cost and speed of drones compared to trucks have a strong influence on the structure of optimal and near-optimal solutions. We examine the impact in four analyses.

Conflicting Objectives First, we show that the two objectives are conflicting. To this end, we re-evaluate optimal and best-known solutions for one objective using the other objective. These values are then compared with best-known solutions for the other objective. Figure 4.4 shows the resulting average percentage increase in objective value for different values of β and m.

Considering the evaluation of the solutions optimized with respect to routing cost using the duration objective (labeled with 'cost $\rightarrow$ duration'), we find the highest average deviation of 23.7% for $\beta = 3$. Results for $m = 49$ should be treated with caution, since the number of optimal solutions is small so that best-known upper bounds are used instead.

For the reverse evaluation (labeled with 'duration $\rightarrow$ cost'), the cost increase for $\beta = 1$ stands out. Solutions optimized with respect to duration are (on average)

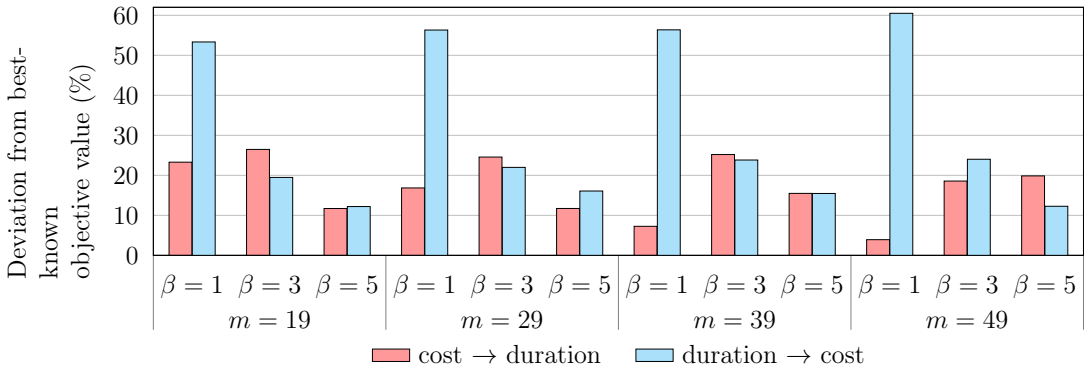


Figure 4.4: Increase of the objective value when changing the objective (cost versus duration minimizaion).

rather bad solutions for routing-cost minimization. An explanation is related to the analysis presented in the next paragraph (see Table 4.3 and Example 9) where we highlight that about 43% of all customers are served by a drone, when it comes to duration minimization. However, when the cost is minimized, drone flights are not profitable. Consequently, converting a duration solution into a corresponding cost solution will drastically increase the cost. For $\beta = 3$ and $\beta = 5$, the deviations resulting from the two re-evaluations are in a rather similar range for all instance sizes.

	Cost objective			Duration objective		
m	$\beta = 1$	$\beta = 3$	$\beta = 5$	$\beta = 1$	$\beta = 3$	$\beta = 5$
19	0%	27%	41%	29%	51%	55%
29	0%	27%	40%	25%	52%	55%
39	0%	22%	38%	26%	49%	52%
49	1%	23%	32%	26%	46%	48%

Table 4.3: Share of drone customers in optimal/best-known solutions.

Share of drone customers We now analyze the structure of optimal and best-known solutions. As shown in Table 4.3, the share of drone customers varies considerably depending on the objective, the size of the instance, and the β-value. On average, the share increases with β and with the substitution of cost by duration minimization. In turn, the share decreases slightly with increasing instance size. The influence of the parameter β is as expected, since cheaper and faster drones encourage their use.

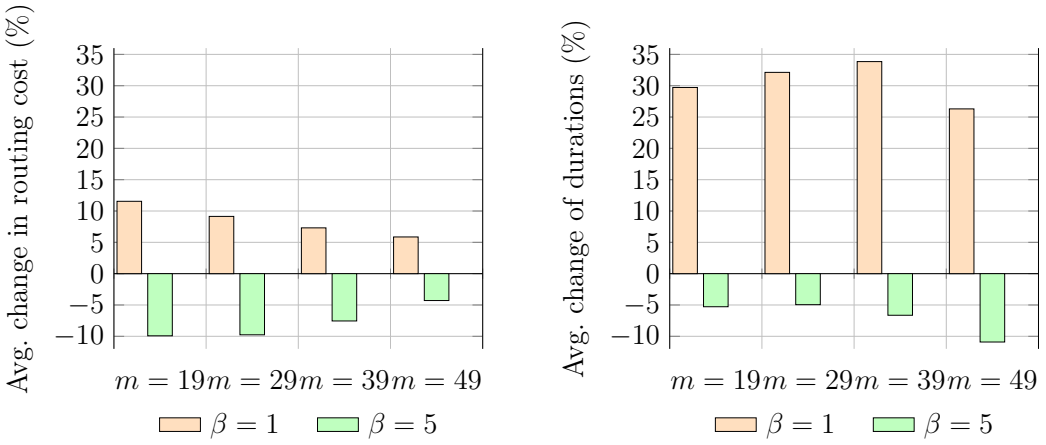
The most surprising and striking result is that drones are hardly ever used when the routing cost is minimized and trucks and drones have identical cost/speed characteristics ($\beta = 1$). The following example explains the above result.

Example 9 *Consider a tiny VRP-D instance with three customers $N = \{1, 2, 3\}$. The complete undirected graph is then $G = (V, E)$ with $V = \{0, 1, 2, 3, 0'\}$. Recall from Section 4.5.1 that routing costs and travel times are computed from Manhattan distances for trucks and Euclidean distances for drones. Accordingly, we define $t_{ij} = c_{ij} = 2$ and $t^{dr}_{ij} = c^{dr}_{ij} = 1.5/\beta$ for all $\{i, j\} \in E$. (Similar distances of 2 and $1.5 \approx \sqrt{2}$ occur when the coordinates in x- and y-directions differ by one.)*

When all three customers are served by the same truck, the corresponding route r has an objective value of $c_r = 8$, regardless of the order in which the customers are visited. This is true for both objectives. Alternatively, we assume that one of the customers is served by the drone and the other two by the same truck to which the drone is assigned. The corresponding route is denoted by r'. Its drone subpath has a routing cost of $2 \cdot 1.5/\beta$, and the duration increases by $\max\{2, 2 \cdot 1.5/\beta\}$. Hence, the routing cost accumulates to $c_{r'} = 6 + 3/\beta$, and the duration is $t_{r'} = 4 + \max\{2, 3/\beta\}$.

For $\beta < 1.5$, it follows that $c_r < c'_r$ for routing costs so that using the drone is not beneficial (in particular for $\beta = 1$). This relationship is reversed for $\beta > 1.5$ so that the use of the drone is cost-effective (in particular for $\beta \in \{3, 5\}$). This reasoning is empirically proven by the results shown on the left side of Table 4.3. For the duration objective, the first travel time $t_r = 8$ is never smaller than the second travel time $t_{r'} = 4 + \max\{2, 3/\beta\}$ (assuming that the drone speed is never lower than the truck speed, i.e., $\beta \geq 1$). Therefore, it is always advantageous to use the drone, as can be seen from the results on the right side of Table 4.3. □

Comparison of different β-Values Next, we compare optimal and best-known solutions for $\beta = 3$ with optimal and best-known solutions for the values $\beta = 1$ and $\beta = 5$, respectively. Figure 4.5 visualizes the average percentage changes relative to the baseline solutions for $\beta = 3$. First and foremost, the percentage changes are smaller for cost minimization (Figure 4.5a) and larger for the duration minimization (Figure 4.5b). For routing costs, increasing the drones' routing costs by a factor of 3 (from $\beta = 3$ to $\beta = 1$) results in a total cost increase between 5 and 12%, depending on the instance size. Reducing the routing cost of the drones (from $\beta = 3$ to $\beta = 5$) generates cost savings of about the same amount. For duration minimization, the corresponding changes are between 25 and 35% and between 5 and 11%, respectively. The latter result implies that drones should be faster than trucks, but further increasing their speed beyond $\beta = 5$ will not generate substantial additional savings.



(a) Routing cost. **(b)** Duration.

Figure 4.5: Average percentage change in routing cost and duration for different β-values.

Comparison with Approximation Results The above results are in line with those of Carlsson and Song (2018), namely, that the efficacy of the combined truck-and-drone delivery increases with the ratio of the speeds of the drones and trucks. Carlsson and Song (2018)'s most important finding is that 'the improvement in efficiency [resulting from the use of a drone] is proportional to the square root of the ratio of the speeds of the truck and the unmanned aerial vehicle' [drone]. Carlsson and Song (2018) do not use MIP-based optimization techniques, but a continuous approximation model (see Daganzo, 2005). They measure efficiency as the duration of a tour (a *max*-term as in the duration objective (4.1)). However, in their model, only the drone delivers to the customers and the truck serves as a support vehicle (as in case (1) in Section 4.1). When reducing the drone speed from $\beta = 3$ to $\beta = 1$, Carlsson and Song (2018)'s model predicts a 73% increase in duration ($\sqrt{3} = 1.73$). Our results show a relatively smaller duration increase of 26% on average, which can be partly explained by the fact that, in our setting, only about half of the customers can be served by the drones (see Table 4.3). Increasing the drone speed from $\beta = 3$ to $\beta = 5$, they predict a 23% reduction in duration ($\sqrt{3/5} = 0.77$). Our results are then rather consistent with the above numbers showing an average duration reduction of only 6.2%.

4.5.5 Further Results

Roberti and Ruthmair (2021) model and solve a variety of constraints and options that are meaningful for the TSP-D. They mention (a) loops (the drone is allowed

to return to the location where it was launched), (b) incompatible customers (customers that must be served by a truck), (c) the drone cannot land and wait for the truck, (d) launch and rendezvous times (additional time to be added when a drone is launched from or lands at a customer location), (e) a maximum number of customers per truck leg, and (f) a limited drone flight range. These extensions of the basic TSP-D are certainly also relevant for the multiple-truck case.

Since our artificial network is based on the network introduced by Roberti and Ruthmair (2021), the same DP adaptations can be used here. All variants except for (e) can be handled by restricting the label propagation and/or restricting the set of drone arcs, while (e) requires the introduction of an additional attribute into the labels. We have exemplarily tested our algorithm with a limited drone flight range. See Section 4.A in the Appendix for details.

Additionally, we have tested the performance of our BPC on the instance set proposed by Zhen *et al.* (2023). Note that Zhen *et al.* (2023) present one of the exact BPC algorithms that is applicable to the basic VRP-D and closely related variants. Results can be found in Section 4.B in the Appendix.

Finally, we considered the TSP-D instances of Roberti and Ruthmair (2021) and Blufstein *et al.* (2024) and solved them twice as VRP-D instances, once with a non-binding capacity and once with capacity of $Q = 8$ and unit demand for all customers. Results are presented in Section 4.C of the Appendix.

4.6 Conclusions

In this work, we presented a new exact solution algorithm for the basic VRP-D that is based on BPC. Its most important algorithmic component is a new implicit bidirectional labeling algorithm for solving pricing subproblems based on an artificial network originally introduced by Roberti and Ruthmair (2021) for TSP-D. In this artificial network, the same route, once considered in forward and once in backward direction, passes through different vertices. Our new implicit bidirectional labeling algorithm exploits the symmetry inherent in the VRP-D as it however does not distinguish forward and backward directions. Roberti and Ruthmair (2021) have shown how to modify the artificial network to cope with TSP-D variants that consider, e.g., drone loops in addition to drone subpaths, a (possibly weight-dependent) drone flight range, and a maximum number of customers per truck segment. Using the same ideas, our approach is adaptable to the corresponding VRP-D variants.

For the performance of the BPC algorithm, several known techniques had to be adapted to the VRP-D case: delayed propagation of some resource values is helpful to handle the non-robust cutting planes (subset-row inequalities and capacity cuts) that strengthen the linear relaxation of the column-generation MP. In addition, the

performance of the new bidirectional labeling algorithm is improved by problem-tailored preprocessing and acceleration techniques of the merge procedure, which is otherwise the bottleneck of the overall solution algorithm. At the end, the BPC algorithm is able to exactly minimize the routing cost for 14 (3) out of the 20 instances of the VRP-D with 39(49) customers. The minimization of the duration is more difficult such that only half of the instances with up to 29 customers can be solved to optimality. However, feasible solutions with an average gap of around 2.3% (5.9%) are found for instances with 39 (49) customers. Compared to other exact algorithms for variants of the VRP-D (see Section 4.2) that allow solution times of up to 3, 5, 10, and even 20 hours for instances with not more than 35 customers, our new BPC algorithm obtains the reported results within one hour.

Our results should also be of interest for other emerging variants of combined vehicle routing. For example, the VRP-D considered here shares many similarities with routing problems in last-mile delivery with trucks and delivery robots (see, e.g., Boysen *et al.*, 2018). For the future, the introduction of time windows and the consideration of time-related objective functions (such as total [weighted] tardiness or lateness, the makespan etc.) into VRP-D variants makes the truck-drone synchronization aspect even more challenging, especially, when considering the exact solution of these problems.

Acknowledgement

This research was supported by Deutsche Forschungsgemeinschaft (DFG) under project no. 418727865 grant IR 122/10-1. This support is gratefully acknowledged. In addition, the authors gratefully acknowledge the computing time granted on the supercomputer MOGON 2 at Johannes Gutenberg University Mainz (hpc.uni-mainz.de).

Bibliography

Augerat, P., Belenguer, J. M., Benavent, E., Corberán, A., Naddef, D., and Rinaldi, G. (1995). Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical report, Institut National Polytechnique, 38 - Grenoble (France).

Bakir, I. and Tiniç, G. Ö. (2020). Optimizing drone-assisted last-mile deliveries: The vehicle routing problem with flexible drones. Optimization online. `https://optimization-online.org/?p=16382`.

Baldacci, R., Christofides, N., and Mingozzi, A. (2008). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, **115**(2), 351–385.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Baldacci, R., Mingozzi, A., and Roberti, R. (2012). New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*, **24**(3), 356–371.

Blufstein, M., Lera-Romero, G., and Soulignac, F. J. (2024). Decremental state-space relaxations for the basic traveling salesman problem with a drone. *INFORMS Journal on Computing*, **36**(4), 1064–1083.

Bode, C. and Irnich, S. (2012). Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research*, **60**(5), 1167–1182.

Bode, C. and Irnich, S. (2015). In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. *Transportation Science*, **49**(2), 369–383.

Boysen, N., Schwerdfeger, S., and Weidinger, F. (2018). Scheduling last-mile deliveries with truck-based autonomous robots. *European Journal of Operational Research*, **271**(3), 1085–1099.

Carlsson, J. G. and Song, S. (2018). Coordinated logistics with a truck and a drone. *Management Science*, **64**(9), 4052–4069.

Chung, S. H., Sah, B., and Lee, J. (2020). Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions. *Computers & Operations Research*, **123**, 105004.

Costa, L., Contardo, C., and Desaulniers, G. (2019). Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, **53**, 946–985.

Daganzo, C. (2005). *Logistics Systems Analysis*. Springer, Berlin, 4th edition.

D'Andrea, R. (2014). Guest editorial Can drones deliver? *IEEE Transactions on Automation Science and Engineering*, **11**(3), 647–648.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Drexl, M. (2012). Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. *Transportation Science*, **46**(3), 297–316.

Gamache, M., Soumis, F., Marquis, G., and Desrosiers, J. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations Research*, **47**(2), 247–263.

Goeke, D., Gschwind, T., and Schneider, M. (2019). Upper and lower bounds for the vehicle-routing problem with private fleet and common carrier. *Discrete Applied Mathematics*, **264**, 43–61.

Goodchild, A. and Toy, J. (2018). Delivery by drone: An evaluation of unmanned aerial vehicle technology in reducing CO_2 emissions in the delivery service industry. *Transportation Research Part D: Transport and Environment*, **61**, 58–67.

Heßler, K. and Irnich, S. (2023). Partial dominance in branch-price-and-cut for the basic multicompartment vehicle-routing problem. *INFORMS Journal on Computing*, **35**(1), 50–65.

Irnich, S. (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

Irnich, S., Toth, P., and Vigo, D. (2014). The family of vehicle routing problems. In Toth and Vigo (2014), chapter 1, pages 1–33.

ITA (2023). Unmanned aircraft systems (UAS). International Trade Administration (ITA) `https://www.trade.gov/unmanned-aircraft-systems`, Last accessed 2023-05-03.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**(2), 497–511.

Joerss, M., Schröder, J., Neuhaus, F., Klink, C., and Mann, F. (2016). Parcel delivery: The future of last mile. McKinsey&Company, Brochure. `https://bdkep.de/files/bdkep-dateien/pdf/2016_the_future_of_last_mile.pdf`

Lera-Romero, G., Miranda Bront, J. J., and Soulignac, F. J. (2022). Dynamic programming for the time-dependent traveling salesman problem with time windows. *INFORMS Journal on Computing*, **34**(6), 3292–3308.

Li, H. and Wang, F. (2022). Branch-price-and-cut for the truck–drone routing problem with time windows. *Naval Research Logistics (NRL)*, **70**(2), 184–204.

Macrina, G., Di Puglia Pugliese, L., Guerriero, F., and Laporte, G. (2020). Drone-aided routing: A literature review. *Transportation Research Part C: Emerging Technologies*, **120**, 102762.

Madani, B. and Ndiaye, M. (2022). Hybrid truck-drone delivery systems: A systematic literature review. *IEEE Access*, **10**, 92854–92878.

Martinelli, R., Poggi, M., and Subramanian, A. (2013). Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, **40**(8), 2145–2160.

Moshref-Javadi, M. and Winkenbach, M. (2021). Applications and research avenues for drone-based models in logistics: A classification and review. *Expert Systems with Applications*, **177**, 114854.

Otto, A., Agatz, N., Campbell, J., Golden, B., and Pesch, E. (2018). Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks*, **72**(4), 411–458.

Poikonen, S., Wang, X., and Golden, B. (2017). The vehicle routing problem with drones: Extended models and connections. *Networks*, **70**(1), 34–43.

Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, **3**(3), 255–273.

Roberti, R. and Mingozzi, A. (2014). Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*, **48**(3), 413–424.

Roberti, R. and Ruthmair, M. (2021). Exact methods for the traveling salesman problem with drone. *Transportation Science*, **55**(2), 315–335.

Tamke, F. and Buscher, U. (2021). A branch-and-cut algorithm for the vehicle routing problem with drones. *Transportation Research Part B: Methodological*, **144**, 174–203.

Tilk, C. and Irnich, S. (2017). Dynamic programming for the minimum tour duration problem. *Transportation Science*, **51**(2), 549–565.

Tilk, C., Rothenbächer, A.-K., Gschwind, T., and Irnich, S. (2017). Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research*, **261**(2), 530–539.

Toth, P. and Vigo, D., editors (2014). *Vehicle Routing*, volume 18 of *MOS-SIAM Series on Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Valério de Carvalho, J. M. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, **86**, 629–659.

Wang, X., Poikonen, S., and Golden, B. (2017). The vehicle routing problem with drones: several worst-case results. *Optimization Letters*, **11**(4), 679–697.

Wang, Z. and Sheu, J.-B. (2019). Vehicle routing problem with drones. *Transportation Research Part B: Methodological*, **122**, 350–364.

Xia, Y., Zeng, W., Zhang, C., and Yang, H. (2023). A branch-and-price-and-cut algorithm for the vehicle routing problem with load-dependent drones. *Transportation Research Part B: Methodological*, **171**, 80–110.

Zhen, L., Gao, J., Tan, Z., Wang, S., and Baldacci, R. (2023). Branch-price-and-cut for trucks and drones cooperative delivery. *IISE Transactions*, **55**(3), 271–287.

Zhou, H., Qin, H., Cheng, C., and Rousseau, L.-M. (2023). An exact algorithm for the two-echelon vehicle routing problem with drones. *Transportation Research Part B: Methodological*, **168**, 124—150.

Appendix

4.A Results for VRP-D with a Limited Drone Flight Range

The drone flight range constraint, occurs in two forms (see Roberti and Ruthmair, 2021, Sect. 3.2 and 4.2): an upper bound U on the flight time given by $t^{\mathrm{dr}}_{i^{\mathrm{dr}},k} + t^{\mathrm{dr}}_{k,j^{\mathrm{dr}}}$ for a drone arc $[i, j, k]$, or an upper bound on the weight dependent flight range. We analyze the former restriction of a limited flying time. For the sake of simplicity, we generate new instances using a single parameter (percentage rate) $u \in [0, 100]$. The instance-specific limit is chosen as the largest value such that $t^{\mathrm{dr}}_{i^{\mathrm{dr}},k} + t^{\mathrm{dr}}_{k,j^{\mathrm{dr}}} \leq U$ is fulfilled for not more than $u|A^{drone}|/100$ drone arcs. Thus, the instance can be constructed by eliminating the other $100 - u$ percent of the ('longer') drone arcs.

Table 4.4 shows results for $u = 100$, 50, and 30 aggregated over instances with an identical number of customers. The column entries have the following meaning: (u) percentage rate of customers that can be served by drone considering the drone's flight time, (m) number of customers in the instance group, (#Opt) number of instances solved to proven optimality within the time limit, ($\mathrm{Gap_{LP}}$) average LP gap ($100 \cdot (opt - z_{LP})/z_{LP}$) in percent, ($\mathrm{Gap_{cut}}$) average LP gap after extending the ng-neighborhoods and adding cutting planes in percent, (Gap) average relative difference between the upper bound (UB) and lower bound (LB) in percent at termination, i.e., $100(UB - LB)/LB$ (computed only when an upper and lower bound was found), (Time) average solution time in seconds, and (#BB) average number of branch-and-bound nodes solved per VRP-D instance. For both $\mathrm{Gap_{LP}}$ and $\mathrm{Gap_{cut}}$, we only consider instances with a known optimal solution *opt*.

Independent from the considered objective function, the general trend is that more constrained flight times lead to instances that are easier to solve, i.e., more provably optimal solutions (*#Opt*), and shorter total BPC solution times (*Time*). Independent from the considered objective function, the general trend is that more constrained flight times lead to instances that are easier to solve, i.e., more provably optimal solutions (*#Opt*), and shorter total BPC solution times (*Time*). Since gaps (*Gap*) are computed only when a feasible solution is found, the corresponding values do not show a clear trend when comparing different percentage rates u.

		Cost Objective						Duration Objective					
u	m	#Opt	Gap_{LP}	Gap_{cut}	Gap	Time	#BB	#Opt	Gap_{LP}	Gap_{cut}	Gap	Time	#BB
100	19	20	2.92	0.30	0.00	22.3	37.7	19	3.59	1.42	0.01	335.7	232.7
	29	18	3.43	0.62	0.02	814.8	163.4	11	2.11	1.00	1.27	2,486.0	325.6
	39	14	2.59	0.26	0.16	1,634.5	141.1	0	—	—	2.29	3,640.9	95.4
	49	3	2.28	0.06	3.34	3,161.5	84.9	0	—	—	5.92	3,655.2	33.3
Subtotal		55	2.97	0.38	0.88	1,408.3	106.8	30	3.04	1.26	2.13	2,529.5	171.7
50	19	20	2.99	0.23	0.00	12.0	32.7	20	5.96	1.04	0.00	75.4	85.0
	29	19	3.61	0.56	0.01	442.6	126.9	14	3.41	0.90	0.58	1,793.2	315.5
	39	17	2.76	0.28	0.09	1,399.6	154.5	2	1.64	0.12	2.61	3,346.0	134.0
	49	3	2.28	0.02	2.21	3,153.7	96.9	0	—	—	8.54	3,647.1	53.7
Subtotal		59	3.09	0.34	0.58	1,252.0	102.7	36	4.73	0.94	2.30	2,215.4	147.0
30	19	20	3.24	0.21	0.00	11.0	32.9	20	5.94	0.67	0.00	55.8	130.1
	29	20	3.76	0.55	0.00	338.3	115.0	13	4.21	0.78	0.23	1,848.1	312.7
	39	18	2.81	0.28	0.10	1,113.2	146.4	5	2.63	0.36	3.19	3,076.3	136.7
	49	5	2.73	0.05	2.64	3,010.8	97.6	0	—	—	14.31	3,644.1	55.6
Subtotal		63	3.24	0.32	0.69	1,118.3	98.0	38	4.91	0.66	3.92	2,156.1	158.7

Table 4.4: Aggregated results for different drone flying ranges for cost and duration minimization.

4.B Results for VRP-D Instances by Zhen *et al.* (2023)

Zhen *et al.* (2023) define a variant of the VRP-D similar to ours, but with two additional aspects. First, the variant includes a weight-dependent drone flight range of the form $g^{\text{on}}_{i^{\text{dr}},k} + g^{\text{off}}_{k,j^{\text{dr}}} \leq U$, for which Roberti and Ruthmair (2021) argue that these constraints can be handled similar to the drone flight ranges considered above. Here, $g^{\text{on}}_{i^{\text{dr}},k}$ typically measures the energy consumption of the drone when flying the first leg between i^{dr} and k and thus taking into account the demand q_k of the drone customer, and $g^{\text{off}}_{k,j^{\text{dr}}}$ is another energy consumption for the second leg between k and j^{dr}. Second, this variant includes waiting costs when the truck has to wait for its drone, i.e., $t^{\text{dr}}_{i^{\text{dr}},k} + t^{\text{dr}}_{k,j^{\text{dr}}} > R_i^{dur}$ in (4.6a). For further details, we refer to Zhen *et al.* (2023). For the following comparison, we ignore both types of extensions in our BPC algorithm.

The test set of Zhen *et al.* (2023) contains ten instances with $m = 9$ and 14 customers, respectively. Table 4.5 compares the results of our BPC algorithm with the results reported in Zhen *et al.* (2023) while Table 4.6 contains instance-wise results of our algorithm. Among the 20 optimal solutions obtained for the instances, only five contain a single drone subpath while the other 15 solutions do not use the drone to serve customers. For the solutions containing drone flights, the trucks never wait for the drone (so that the second extension is irrelevant). The paper by Zhen *et al.* (2023) allows for different interpretations of the rounding rules used to compute routing costs, travel times, and other coefficients. Table 4.5 should therefore be seen as a rough performance comparison, taking into accout that the BPC of Zhen *et al.* (2023) and our BPC algorithm actually solve different VRP-D variants. For the same reasons, Table 4.6 present the unrounded upper bound UB (since all instances are optimally solved, this is the optimal objective value) and the rounded UB as presented by Zhen *et al.* (2023). Our bounds differ by $+1$ for the three rounded UB-values printed in **bold**.

	Zhen *et al.* (2023)			Our algorithm				
m	#Opt	Gap	Time	#Opt	Gap_{LP}	Gap_{cut}	Gap	Time
9	10	0.00	75.4	10	37.89	0.08	0.00	3.5
14	10	0.00	4347.6	10	9.91	0.16	0.00	10.5
Total	20	0.00	2211.5	20	23.90	0.12	0.00	7.0

Table 4.5: Comparison on instance set introduced by Zhen *et al.* (2023).

Instance	Opt	UB	UB (rounded)	Gap_{LP}	Gap_{cut}	Gap	Time	#BB
9-1	*	323.76	324	38.83	0.00	0.00	0.6	7
9-2	*	331.12	331	37.96	0.00	0.00	0.8	10
9-3	*	332.90	**333**	29.70	0.00	0.00	0.4	8
9-4	*	312.10	312	35.81	0.00	0.00	0.9	8
9-5	*	306.51	**307**	39.39	0.00	0.00	1.6	4
9-6	*	343.30	343	34.31	0.77	0.00	1.8	23
9-7	*	299.00	299	44.56	0.00	0.00	15.3	18
9-8	*	290.17	290	34.30	0.00	0.00	2.2	17
9-9	*	354.94	355	35.71	0.00	0.00	0.9	12
9-10	*	305.46	305	48.29	0.00	0.00	10.5	25
14-1	*	352.26	352	5.93	0.70	0.00	15.5	36
14-2	*	354.96	355	10.72	0.00	0.00	9.8	21
14-3	*	362.87	363	5.44	0.00	0.00	2.6	7
14-4	*	346.86	347	11.30	0.00	0.00	2.1	6
14-5	*	328.98	329	8.59	0.00	0.00	10.7	12
14-6	*	340.00	340	25.59	0.88	0.00	27.4	43
14-7	*	355.60	356	3.73	0.00	0.00	4.8	11
14-8	*	370.97	371	12.54	0.00	0.00	11.5	13
14-9	*	348.99	**349**	11.55	0.00	0.00	14.8	25
14-10	*	329.89	330	3.77	0.00	0.00	6.0	20

Table 4.6: Detailed results on the instances used in Zhen *et al.* (2023).

4.C Results for TSP-D Instances

We also conduct two types of experiments on the TSP-D instances used by Roberti and Ruthmair (2021, RR) and Blufstein *et al.* (2024, BLU). We consider six subgroups of instances differing in the number of customers $m \in \{19, 29\}$ and the speed factor $\beta \in \{1, 2, 3\}$, consisting of 25 instances each.

For the first experiment, we use our VRP-D-tailored BPC algorithm to solve the TSP-D instances without a vehicle capacity (setting Q to a high non-binding value). Non surprisingly, results are rather poor as can be seen from the columns headed with *TSP-D* in Table 4.7. Compared to the excellent results of Roberti and Ruthmair (2021) and Blufstein *et al.* (2024), our BPC algorithm cannot exploit any specific property of the TSP-D as, e.g., using a stagewise labeling in terms of the number of visited customers. In addition, the dominance relations (Dominance Rules 1 and 2) still incorporate the resource R^{load} for the load.

In a second experiment, we considered the same instances as VRP-D instances with the duration objective and setting the capacity of the trucks to $Q = 8$. Unity demand $q_v = 1$ is assumed for each customer $v \in N$. The results are shown in the columns headed with *VRP-D* in Table 4.7. Our BPC algorithm performs much better when it is used for capacitated instances, i.e., it is able to optimally solve many of these VRP-D instances, in particular when the drone is much faster than a truck (for $\beta = 3$).

		Time				#Opt			
		RR	BLU	our algorithm		RR	BLU	our algorithm	
m	β			TSP-D	VRP-D			TSP-D	VRP-D
19	1	47.4	—	1,095.1	876.6	25	—	23	21
	2	20.7	—	974.5	73.8	25	—	19	25
	3	17.6	—	1,573.9	12.2	25	—	15	25
29	1	1,209.0	7.6	3,637.9	2,675.5	19	25	0	12
	2	554.3	7.1	3,433.2	1,760.5	24	25	4	15
	3	537.0	11.8	3,466.3	522.9	23	25	2	24

Table 4.7: Aggregated results on TSP-D instances.

4.D Detailed Results per Instance

Tables 4.8–4.13 present detailed computational results for the complete VRP-D instance set. Tables 4.8–4.10 report the results for routing-cost minimization and Tables 4.11–4.13 for duration minimization.

All tables are structured according to the following scheme: The first two columns state the instance name (*Instance*) and the number of truck-only customers (#To) considered in the instances. The third column (*Opt*) is indicated with an asterisk * if the algorithm could solve the instance to proven optimality within the given time limit. The next five columns report the following values: The computed upper bound (UB), lower bound at the root node (LB_{LP}), lower bound at the root node after adding cutting planes (LB_{cut}), lower bound when reaching the time limit (LB), and the average relative difference between the upper bound UB and lower bound LB in percent at termination (*Gap*). Please note, that the gap is only computed when an upper and lower bound was found. The next three columns show the number of capacity cuts ($\#CC$), subset row inequalities ($\#SR$), and rounds of dynamic ng extensions ($\#dyn_{ng}$) added. The last columns report the solution time in seconds (*Time*) and the number of solved Branch-and-Bound nodes ($\#BB$). Cells filled with '*TL*' indicate that the time limit of 3,600 seconds for the BPC algorithm was reached.

Table 4.8: Detailed results for $\beta = 1$ and cost minimization.

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
A-n32-k5-20	4	*	6,700	6,700	6,700	6,700	0.00	0	0	0	0.6	1
A-n33-k5-20	3	*	5,900	5,877	5,900	5,900	0.00	0	0	2	0.8	3
A-n33-k6-20	5	*	5,980	5,717	5,931	5,980	0.00	6	7	14	19.5	80
A-n34-k5-20	5	*	6,200	6,055	6,180	6,200	0.00	14	3	10	6.3	27
A-n36-k5-20	3	*	6,560	6,453	6,560	6,560	0.00	4	10	10	14.5	18
A-n37-k5-20	4	*	6,120	5,897	6,020	6,120	0.00	5	10	13	60.8	63
A-n37-k6-20	3	*	6,557	6,309	6,557	6,557	0.00	3	10	15	8.2	21
A-n38-k5-20	2	*	5,880	5,840	5,880	5,880	0.00	2	0	4	1.8	7
A-n39-k5-20	5	*	5,040	4,813	5,025	5,040	0.00	6	10	18	31.7	29
A-n39-k6-20	6	*	7,040	6,688	7,040	7,040	0.00	6	10	12	4.0	22
A-n44-k7-20	2	*	6,000	6,000	6,000	6,000	0.00	0	0	0	0.3	1
A-n45-k6-20	3	*	6,780	6,317	6,620	6,780	0.00	9	10	9	85.8	158
A-n45-k7-20	3	*	6,840	6,440	6,699	6,840	0.00	9	10	12	62.0	103
A-n46-k7-20	3	*	6,280	6,080	6,280	6,280	0.00	2	0	3	1.5	6
A-n48-k7-20	6	*	7,400	7,155	7,263	7,400	0.00	0	10	7	49.6	73
A-n53-k7-20	5	*	6,500	6,500	6,500	6,500	0.00	0	0	0	1.0	1
A-n54-k7-20	3	*	6,280	6,058	6,280	6,280	0.00	9	10	7	7.4	17
A-n55-k9-20	4	*	5,940	5,850	5,940	5,940	0.00	0	6	0	1.3	3
A-n60-k9-20	3	*	5,840	5,772	5,840	5,840	0.00	1	3	11	3.5	14
A-n61-k9-20	4	*	5,040	4,763	4,950	5,040	0.00	3	10	8	23.4	49
A-n32-k5-30	6	*	9,540	9,263	9,538	9,540	0.00	6	10	25	131.6	38
A-n33-k5-30	4	*	7,680	7,292	7,548	7,680	0.00	9	10	19	199.2	48
A-n33-k6-30	10	*	8,880	8,700	8,810	8,880	0.00	0	3	16	87.5	108
A-n34-k5-30	7	*	8,480	8,209	8,480	8,480	0.00	13	10	14	35.5	29
A-n36-k5-30	3	*	8,438	8,316	8,360	8,438	0.00	1	10	13	404.3	78
A-n37-k5-30	5	*	7,116	6,945	7,064	7,116	0.00	1	10	19	423.7	41
A-n37-k6-30	3	*	9,220	9,007	9,219	9,220	0.00	6	10	25	73.0	33
A-n38-k5-30	4	*	7,920	7,521	7,920	7,920	0.00	9	10	24	62.6	35
A-n39-k5-30	7	*	7,920	7,864	7,920	7,920	0.00	2	3	11	25.9	14
A-n39-k6-30	8	*	8,560	8,338	8,560	8,560	0.00	14	10	18	36.1	31
A-n44-k7-30	4	*	8,560	8,433	8,531	8,560	0.00	6	10	16	73.6	37
A-n45-k6-30	7	*	8,380	8,249	8,380	8,380	0.00	4	0	12	23.2	14
A-n45-k7-30	6	*	9,720	9,490	9,698	9,720	0.00	4	10	29	54.3	38
A-n46-k7-30	5	*	8,320	8,169	8,320	8,320	0.00	8	10	11	29.2	18
A-n48-k7-30	6	*	9,600	9,526	9,600	9,600	0.00	1	0	10	16.3	12

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
A-n53-k7-30	6	*	8,060	7,765	8,034	8,060	0.00	12	10	13	199.7	35
A-n54-k7-30	5	*	9,200	8,819	9,070	9,200	0.00	10	10	26	781.4	52
A-n55-k9-30	8	*	8,320	7,963	8,190	8,320	0.00	14	10	11	125.5	87
A-n60-k9-30	5	*	7,960	7,728	7,939	7,960	0.00	12	10	24	143.8	44
A-n61-k9-30	5	*	6,760	6,519	6,707	6,760	0.00	8	10	16	88.2	60
A-n44-k7-40	7		10,820	10,522	10,599	10,729	0.85	5	10	14	TL	619
A-n45-k6-40	9	*	10,940	10,599	10,860	10,940	0.00	18	10	24	447.9	75
A-n45-k7-40	7		12,120	11,627	11,908	12,008	0.93	12	10	37	TL	529
A-n46-k7-40	6	*	10,000	9,735	9,945	10,000	0.00	15	10	32	1,010.5	75
A-n48-k7-40	8	*	11,640	11,484	11,640	11,640	0.00	6	0	24	183.9	27
A-n53-k7-40	9	*	9,800	9,470	9,751	9,800	0.00	13	10	27	1,224.0	39
A-n54-k7-40	5	*	10,440	10,257	10,440	10,440	0.00	16	10	43	506.2	57
A-n55-k9-40	11	*	10,240	9,920	10,149	10,240	0.00	15	10	22	1,924.2	586
A-n60-k9-40	5	*	10,040	9,729	10,040	10,040	0.00	20	10	32	431.2	50
A-n61-k9-40	6	*	9,360	8,994	9,195	9,360	0.00	9	10	21	1,407.3	199
A-n62-k8-40	4	*	10,320	10,049	10,257	10,320	0.00	4	10	21	2,940.0	61
A-n63-k9-40	9	*	13,720	13,443	13,696	13,720	0.00	16	10	39	421.0	56
A-n63-k10-40	11	*	11,680	11,450	11,640	11,680	0.00	5	10	33	402.5	55
A-n64-k9-40	10		11,622	11,190	11,288	11,408	1.88	3	10	17	TL	261
A-n65-k9-40	8		10,000	9,599	9,769	9,906	0.95	7	10	18	TL	538
A-n69-k9-40	5		9,280	8,913	9,114	9,199	0.88	12	10	36	TL	158
A-n80-k10-40	10		13,900	13,417	13,703	13,812	0.64	6	10	22	TL	199
B-n41-k6-40	7	*	10,220	9,980	10,215	10,220	0.00	10	10	38	356.6	53
B-n43-k6-40	8		8,883	7,977	8,125	8,178	8.62	7	10	50	TL	115
B-n44-k7-40	8	*	10,500	9,953	10,440	10,500	0.00	15	10	49	1,846.0	89
A-n53-k7-50	13	*	12,240	12,094	12,220	12,240	0.00	12	10	31	1,984.8	60
A-n54-k7-50	7		13,440	12,847	13,120	13,163	2.10	16	10	48	TL	94
A-n55-k9-50	13	*	12,700	12,381	12,643	12,700	0.00	7	10	38	910.0	85
A-n60-k9-50	9		13,800	13,339	13,589	13,643	1.15	34	10	45	TL	128
A-n61-k9-50	6	*	10,880	10,610	10,817	10,880	0.00	19	10	34	2,472.0	143
A-n62-k8-50	9		13,320	12,936	13,124	13,192	0.97	11	10	48	TL	79
A-n63-k9-50	12	*	16,240	16,151	16,240	16,240	0.00	13	0	25	542.9	30
A-n63-k10-50	13		13,960	13,610	13,810	13,846	0.82	22	10	47	TL	153
A-n64-k9-50	10		13,400	12,981	13,129	13,149	1.91	8	10	41	TL	64
A-n65-k9-50	10	*	11,640	11,422	11,640	11,640	0.00	24	10	37	700.1	56
A-n69-k9-50	6	*	10,600	10,440	10,600	10,600	0.00	9	5	32	778.0	39
A-n80-k10-50	12		15,640	15,168	15,318	15,355	1.86	14	10	27	TL	52
B-n50-k7-50	9		10,551	8,803	9,331	9,332	13.06	23	0	50	TL	67
B-n50-k8-50	13		17,040	16,056	16,153	16,166	5.41	9	10	50	TL	79
B-n51-k7-50	10		13,344	12,175	12,926	12,927	3.23	48	10	50	TL	74
B-n52-k7-50	9	*	7,980	7,703	7,978	7,980	0.00	3	10	36	1,221.6	43
B-n56-k7-50	7		12,225	7,364	7,499	7,499	63.02	4	0	31	TL	32
B-n57-k7-50	9		13,060	11,908	13,017	13,017	0.33	30	0	50	TL	59
B-n57-k9-50	10		18,360	17,325	17,945	17,945	2.31	39	10	50	TL	68
B-n63-k10-50	12		16,153	14,752	15,006	15,023	7.52	9	10	47	TL	74

Table 4.9: Detailed results for $\beta = 3$ and cost minimization.

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
A-n32-k5-20	4	*	6,618	6,611	6,618	6,618	0.00	1	0	1	0.8	3
A-n33-k5-20	3	*	5,597	5,476	5,597	5,597	0.00	0	0	15	2.6	16
A-n33-k6-20	5	*	5,292	5,226	5,292	5,292	0.00	0	0	10	2.4	11
A-n34-k5-20	5	*	5,596	5,482	5,596	5,596	0.00	10	0	31	9.3	41
A-n36-k5-20	3	*	5,937	5,634	5,835	5,937	0.00	5	10	35	105.6	109
A-n37-k5-20	4	*	4,947	4,933	4,947	4,947	0.00	0	0	2	1.6	3
A-n37-k6-20	3	*	5,808	5,658	5,799	5,808	0.00	6	8	26	17.3	38
A-n38-k5-20	2	*	5,266	5,087	5,266	5,266	0.00	3	5	39	20.9	43
A-n39-k5-20	5	*	4,336	4,206	4,330	4,336	0.00	3	8	26	14.7	34
A-n39-k6-20	6	*	6,169	5,899	6,123	6,169	0.00	5	8	27	9.4	43
A-n44-k7-20	2	*	5,565	5,481	5,565	5,565	0.00	0	0	12	2.7	13
A-n45-k6-20	3	*	5,937	5,733	5,892	5,937	0.00	6	10	26	34.4	52
A-n45-k7-20	3	*	5,769	5,644	5,769	5,769	0.00	1	0	26	9.2	28
A-n46-k7-20	3	*	5,326	5,015	5,253	5,326	0.00	17	10	27	101.9	93
A-n48-k7-20	6	*	6,760	6,569	6,760	6,760	0.00	11	5	30	12.6	42
A-n53-k7-20	5	*	6,285	6,132	6,285	6,285	0.00	2	0	27	14.9	30
A-n54-k7-20	3	*	5,770	5,579	5,711	5,770	0.00	7	10	36	56.8	73
A-n55-k9-20	4	*	5,186	5,049	5,186	5,186	0.00	0	0	17	2.9	18
A-n60-k9-20	3	*	5,522	5,226	5,522	5,522	0.00	2	0	49	23.8	52
A-n61-k9-20	4	*	4,264	4,161	4,264	4,264	0.00	0	0	11	2.8	12
A-n32-k5-30	6	*	9,124	8,826	9,038	9,124	0.00	7	3	50	587.6	145
A-n33-k5-30	4	*	6,961	6,752	6,878	6,961	0.00	5	10	45	180.4	80
A-n33-k6-30	10	*	8,423	8,067	8,274	8,423	0.00	17	5	47	246.2	222
A-n34-k5-30	7	*	7,908	7,681	7,898	7,908	0.00	20	10	35	86.8	53
A-n36-k5-30	3	*	7,310	7,052	7,290	7,310	0.00	20	10	50	317.8	77

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
A-n37-k5-30	5	*	6,350	6,244	6,350	6,350	0.00	10	10	44	170.5	54
A-n37-k6-30	3	*	8,141	7,926	8,128	8,141	0.00	7	10	47	141.9	60
A-n38-k5-30	4	*	7,140	6,815	7,099	7,140	0.00	9	10	50	324.3	77
A-n39-k5-30	7	*	7,236	6,981	7,146	7,236	0.00	47	10	40	1,466.1	197
A-n39-k6-30	8	*	7,615	7,381	7,581	7,615	0.00	10	10	40	99.5	58
A-n44-k7-30	4	*	7,287	7,130	7,287	7,287	0.00	12	5	50	84.4	61
A-n45-k6-30	7	*	7,903	7,756	7,903	7,903	0.00	14	3	50	73.2	59
A-n45-k7-30	6	*	9,128	8,823	9,063	9,128	0.00	5	10	50	234.5	95
A-n46-k7-30	5	*	7,589	7,299	7,493	7,589	0.00	13	10	35	810.5	273
A-n48-k7-30	6		9,054	8,668	8,929	9,046	0.09	10	8	50	TL	690
A-n53-k7-30	6	*	7,641	7,265	7,585	7,641	0.00	19	10	50	2,641.1	111
A-n54-k7-30	5		8,806	8,481	8,648	8,781	0.28	4	10	50	TL	713
A-n55-k9-30	8	*	7,286	7,184	7,286	7,286	0.00	0	0	17	9.1	18
A-n60-k9-30	5	*	7,480	7,049	7,400	7,480	0.00	17	10	50	1,519.4	160
A-n61-k9-30	5	*	6,134	5,895	6,097	6,134	0.00	15	10	36	98.5	64
A-n44-k7-40	7	*	9,800	9,592	9,774	9,800	0.00	13	10	50	531.5	102
A-n45-k6-40	9	*	10,006	9,830	10,006	10,006	0.00	1	5	50	215.3	53
A-n45-k7-40	7	*	11,311	11,063	11,246	11,311	0.00	21	10	50	1,463.5	190
A-n46-k7-40	6	*	9,054	8,832	9,054	9,054	0.00	18	10	50	426.1	62
A-n48-k7-40	8	*	10,809	10,569	10,809	10,809	0.00	13	0	50	420.6	57
A-n53-k7-40	9	*	9,285	9,018	9,259	9,285	0.00	13	10	50	1,694.0	66
A-n54-k7-40	5		10,135	9,768	10,032	10,101	0.34	30	10	50	TL	166
A-n55-k9-40	11	*	9,760	9,479	9,703	9,760	0.00	29	10	46	878.4	262
A-n60-k9-40	5		9,344	8,854	9,135	9,190	1.68	24	10	50	TL	178
A-n61-k9-40	6	*	8,394	8,105	8,370	8,394	0.00	19	10	50	626.1	79
A-n62-k8-40	4	*	9,501	9,266	9,485	9,501	0.00	3	10	50	974.7	62
A-n63-k9-40	9		12,882	12,626	12,821	12,877	0.04	14	10	50	TL	308
A-n63-k10-40	11	*	11,331	10,972	11,247	11,331	0.00	13	10	50	882.1	136
A-n64-k9-40	10		10,826	10,524	10,774	10,816	0.08	17	10	50	TL	229
A-n65-k9-40	8	*	9,553	9,304	9,514	9,553	0.00	12	10	50	608.9	82
A-n69-k9-40	5	*	7,962	7,752	7,940	7,962	0.00	21	10	50	1,171.4	72
A-n80-k10-40	10	*	12,939	12,656	12,930	12,939	0.00	9	10	50	753.5	63
B-n41-k6-40	7		9,839	9,416	9,710	9,757	0.84	29	10	50	TL	503
B-n43-k6-40	8	*	7,853	7,675	7,853	7,853	0.00	3	5	50	400.2	53
B-n44-k7-40	8		10,267	9,762	10,198	10,248	0.19	30	10	50	TL	98
A-n53-k7-50	13		11,883	11,450	11,665	11,700	1.56	22	10	50	TL	106
A-n54-k7-50	7		12,663	12,219	12,508	12,550	0.90	22	10	50	TL	105
A-n55-k9-50	13	*	12,073	11,877	12,056	12,073	0.00	16	10	50	487.7	67
A-n60-k9-50	9		12,831	12,370	12,607	12,663	1.33	22	10	50	TL	128
A-n61-k9-50	6	*	9,992	9,765	9,990	9,992	0.00	16	10	50	592.6	64
A-n62-k8-50	9		13,044	12,310	12,607	12,665	2.99	26	10	50	TL	74
A-n63-k9-50	12		15,769	15,405	15,645	15,678	0.58	30	10	50	TL	84
A-n63-k10-50	13		13,195	12,740	12,949	12,998	1.52	27	10	50	TL	163
A-n64-k9-50	10		13,255	12,128	12,390	12,398	6.90	19	10	50	TL	72
A-n65-k9-50	10		11,292	10,981	11,175	11,240	0.46	25	10	50	TL	166
A-n69-k9-50	6		10,191	9,476	9,791	9,791	4.09	38	10	50	TL	72
A-n80-k10-50	12		15,001	14,395	14,599	14,600	2.75	22	10	50	TL	65
B-n50-k7-50	9	*	8,626	8,385	8,626	8,626	0.00	1	0	50	538.0	52
B-n50-k8-50	13		15,774	15,547	15,697	15,698	0.48	46	10	50	TL	68
B-n51-k7-50	10		12,732	11,669	12,116	12,116	5.08	51	0	50	TL	69
B-n52-k7-50	9		7,929	7,506	7,830	7,830	1.26	20	0	50	TL	58
B-n56-k7-50	7		8,070	6,927	7,291	7,291	10.68	40	0	50	TL	65
B-n57-k7-50	9		13,372	11,571	12,082	12,083	10.67	47	0	50	TL	62
B-n57-k9-50	10		19,770	16,973	17,265	17,265	14.51	58	0	50	TL	67
B-n63-k10-50	12		14,361	13,901	14,175	14,202	1.12	22	10	50	TL	91

Table 4.10: Detailed results for $\beta = 5$ and cost minimization.

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
A-n32-k5-20	4	*	6,398	6,340	6,398	6,398	0.00	0	0	11	3.7	12
A-n33-k5-20	3	*	5,266	5,084	5,266	5,266	0.00	0	0	27	5.9	28
A-n33-k6-20	5	*	4,816	4,775	4,802	4,816	0.00	9	0	21	11.1	43
A-n34-k5-20	5	*	5,185	5,062	5,151	5,185	0.00	21	10	27	39.8	63
A-n36-k5-20	3	*	4,852	4,769	4,852	4,852	0.00	0	5	26	20.5	28
A-n37-k5-20	4	*	4,576	4,538	4,576	4,576	0.00	0	0	16	6.0	17
A-n37-k6-20	3	*	5,075	4,994	5,075	5,075	0.00	13	9	24	17.0	39
A-n38-k5-20	2	*	4,813	4,583	4,710	4,813	0.00	10	10	50	162.0	135
A-n39-k5-20	5	*	3,983	3,832	3,968	3,983	0.00	0	4	30	16.7	37
A-n39-k6-20	6	*	5,580	5,350	5,485	5,580	0.00	10	10	40	27.6	87
A-n44-k7-20	2	*	4,796	4,796	4,796	4,796	0.00	0	0	1	1.1	2
A-n45-k6-20	3	*	5,220	5,124	5,220	5,220	0.00	9	5	38	16.1	47
A-n45-k7-20	3	*	5,154	5,078	5,146	5,154	0.00	13	10	23	22.8	44
A-n46-k7-20	3	*	4,847	4,533	4,605	4,847	0.00	12	10	50	2,439.9	1668
A-n48-k7-20	6	*	6,162	6,092	6,157	6,162	0.00	3	10	22	13.9	33

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
A-n53-k7-20	5	*	5,762	5,575	5,653	5,762	0.00	14	10	27	101.1	78
A-n54-k7-20	3	*	5,168	5,071	5,141	5,168	0.00	5	10	31	45.4	63
A-n55-k9-20	4	*	4,479	4,449	4,479	4,479	0.00	0	0	13	3.3	14
A-n60-k9-20	3	*	4,863	4,625	4,834	4,863	0.00	2	0	42	55.0	60
A-n61-k9-20	4	*	3,847	3,784	3,847	3,847	0.00	0	0	13	4.0	14
A-n32-k5-30	6	*	8,343	8,171	8,343	8,343	0.00	0	0	28	47.0	29
A-n33-k5-30	4	*	6,377	6,206	6,355	6,377	0.00	6	10	50	181.9	61
A-n33-k6-30	10	*	7,690	7,494	7,614	7,690	0.00	14	10	50	744.0	470
A-n34-k5-30	7	*	7,479	7,255	7,435	7,479	0.00	26	10	45	262.1	94
A-n36-k5-30	3	*	6,295	6,120	6,244	6,295	0.00	8	10	45	386.2	72
A-n37-k5-30	5	*	5,656	5,610	5,656	5,656	0.00	0	2	37	140.9	39
A-n37-k6-30	3	*	7,287	7,063	7,250	7,287	0.00	4	10	50	268.4	70
A-n38-k5-30	4	*	6,295	6,187	6,264	6,295	0.00	3	10	50	191.6	64
A-n39-k5-30	7	*	6,335	6,242	6,325	6,335	0.00	3	10	35	119.8	48
A-n39-k6-30	8	*	6,855	6,650	6,826	6,855	0.00	0	7	32	86.1	59
A-n44-k7-30	4	*	6,572	6,419	6,507	6,572	0.00	8	10	50	500.6	194
A-n45-k6-30	7	*	7,233	7,076	7,193	7,233	0.00	12	10	48	331.3	87
A-n45-k7-30	6	*	8,397	8,289	8,391	8,397	0.00	10	10	38	87.7	50
A-n46-k7-30	5	*	6,740	6,552	6,644	6,740	0.00	4	10	47	938.9	307
A-n48-k7-30	6	*	8,107	7,994	8,107	8,107	0.00	2	5	50	138.0	54
A-n53-k7-30	6	*	6,852	6,638	6,799	6,852	0.00	10	9	50	1,275.0	150
A-n54-k7-30	5	*	7,993	7,849	7,982	7,993	0.00	8	10	50	135.5	60
A-n55-k9-30	8	*	6,783	6,621	6,713	6,783	0.00	13	8	37	421.8	362
A-n60-k9-30	5		6,771	6,435	6,632	6,732	0.58	18	10	50	TL	384
A-n61-k9-30	5	*	5,370	5,279	5,365	5,370	0.00	4	10	38	72.4	46
A-n44-k7-40	7	*	8,992	8,855	8,940	8,992	0.00	23	10	50	1,772.6	295
A-n45-k6-40	9	*	9,299	9,057	9,238	9,299	0.00	6	10	50	1,316.2	143
A-n45-k7-40	7	*	10,235	10,172	10,235	10,235	0.00	0	0	29	170.6	30
A-n46-k7-40	6		8,278	8,046	8,124	8,188	1.10	6	10	50	TL	522
A-n48-k7-40	8	*	10,009	9,853	9,985	10,009	0.00	2	8	50	1,015.5	74
A-n53-k7-40	9		8,760	8,497	8,708	8,739	0.24	7	10	50	TL	81
A-n54-k7-40	5		9,332	9,132	9,262	9,323	0.10	13	10	50	TL	180
A-n55-k9-40	11	*	9,129	8,959	9,075	9,129	0.00	18	10	50	698.1	212
A-n60-k9-40	5		8,378	8,043	8,215	8,285	1.12	27	10	50	TL	134
A-n61-k9-40	6		7,552	7,337	7,479	7,521	0.41	27	10	50	TL	142
A-n62-k8-40	4		8,499	8,347	8,457	8,496	0.04	6	10	50	TL	97
A-n63-k9-40	9	*	11,814	11,618	11,759	11,814	0.00	8	10	50	2,171.7	310
A-n63-k10-40	11	*	10,531	10,363	10,461	10,531	0.00	10	10	50	890.8	147
A-n64-k9-40	10		10,065	9,759	9,964	9,993	0.72	24	10	50	TL	120
A-n65-k9-40	8	*	9,051	8,856	9,029	9,051	0.00	10	10	50	395.7	75
A-n69-k9-40	5		7,116	6,913	7,036	7,086	0.42	9	10	50	TL	95
A-n80-k10-40	10	*	12,098	11,919	12,098	12,098	0.00	0	0	27	169.5	28
B-n41-k6-40	7		9,241	8,927	9,119	9,147	1.03	17	10	50	TL	314
B-n43-k6-40	8	*	7,509	7,427	7,509	7,509	0.00	0	0	28	263.2	29
B-n44-k7-40	8	*	9,795	9,388	9,775	9,795	0.00	23	10	50	2,265.9	97
A-n53-k7-50	13		10,991	10,692	10,844	10,876	1.06	6	10	50	TL	121
A-n54-k7-50	7		11,662	11,421	11,587	11,621	0.35	10	10	50	TL	110
A-n55-k9-50	13	*	11,502	11,345	11,481	11,502	0.00	21	10	50	1,139.4	88
A-n60-k9-50	9		11,782	11,526	11,724	11,777	0.03	25	10	50	TL	114
A-n61-k9-50	6		9,094	8,872	8,983	9,024	0.78	33	10	50	TL	131
A-n62-k8-50	9		12,726	11,400	11,586	11,587	9.83	22	10	50	TL	65
A-n63-k9-50	12		14,969	14,310	14,531	14,658	2.12	29	10	50	TL	93
A-n63-k10-50	13		12,308	12,087	12,218	12,263	0.37	17	10	50	TL	185
A-n64-k9-50	10		13,577	11,290	11,549	11,560	17.45	18	10	50	TL	66
A-n65-k9-50	10		10,715	10,426	10,580	10,626	0.84	18	10	50	TL	159
A-n69-k9-50	6		8,820	8,587	8,723	8,736	0.95	25	10	50	TL	75
A-n80-k10-50	12		13,737	13,522	13,714	13,726	0.08	16	10	50	TL	67
B-n50-k7-50	9	*	8,304	8,100	8,304	8,304	0.00	3	5	50	701.5	54
B-n50-k8-50	13		15,040	14,861	14,988	14,997	0.28	22	10	50	TL	81
B-n51-k7-50	10		12,085	11,433	11,768	11,768	2.69	54	10	50	TL	71
B-n52-k7-50	9		8,054	7,274	7,574	7,574	6.34	14	0	50	TL	56
B-n56-k7-50	7		10,382	6,580	6,955	6,955	49.25	22	0	50	TL	57
B-n57-k7-50	9		13,406	11,276	11,524	11,524	16.32	51	0	50	TL	63
B-n57-k9-50	10		18,453	16,196	16,429	16,429	12.32	54	10	50	TL	70
B-n63-k10-50	12		13,421	13,175	13,339	13,358	0.46	28	10	50	TL	90

Table 4.11: Detailed results for $\beta = 1$ and duration minimization.

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
A-n32-k5-20	4	*	6,042	5,883	6,008	6,042	0.00	8	10	27	301.5	43
A-n33-k5-20	3	*	4,932	4,754	4,932	4,932	0.00	0	0	20	20.9	21
A-n33-k6-20	5	*	4,672	4,628	4,672	4,672	0.00	0	0	12	31.3	13
A-n34-k5-20	5	*	5,017	4,802	4,980	5,017	0.00	16	10	30	329.3	60
A-n36-k5-20	3	*	5,240	4,916	5,199	5,240	0.00	15	10	30	2,336.8	58

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
A-n37-k5-20	4	*	4,584	4,383	4,540	4,584	0.00	3	5	30	1,297.6	65
A-n37-k6-20	3	*	5,380	5,058	5,316	5,380	0.00	10	9	30	211.3	61
A-n38-k5-20	2	*	4,560	4,493	4,560	4,560	0.00	0	0	7	42.9	8
A-n39-k5-20	5	*	3,945	3,788	3,945	3,945	0.00	2	9	30	344.2	35
A-n39-k6-20	6	*	5,630	5,323	5,452	5,630	0.00	6	7	30	477.6	220
A-n44-k7-20	2	*	5,017	4,881	5,017	5,017	0.00	0	0	21	45.2	22
A-n45-k6-20	3	*	5,219	4,995	5,128	5,219	0.00	1	10	29	554.9	77
A-n45-k7-20	3	*	5,254	5,037	5,254	5,254	0.00	11	5	30	491.5	38
A-n46-k7-20	3	*	4,802	4,486	4,671	4,802	0.00	16	10	30	3,331.9	308
A-n48-k7-20	6	*	5,994	5,742	5,994	5,994	0.00	14	9	29	89.9	40
A-n53-k7-20	5	*	5,637	5,292	5,488	5,637	0.00	8	10	30	1,990.4	185
A-n54-k7-20	3		5,524	5,001	5,154	5,263	4.96	7	10	30	TL	204
A-n55-k9-20	4	*	4,615	4,426	4,557	4,615	0.00	2	10	30	143.1	60
A-n60-k9-20	3		5,165	4,620	5,010	5,144	0.41	14	10	30	TL	176
A-n61-k9-20	4	*	3,869	3,669	3,734	3,869	0.00	0	3	4	542.3	714
A-n32-k5-30	6		8,227	7,825	8,012	8,013	2.67	0	0	28	TL	28
A-n33-k5-30	4		6,385	5,969	6,199	6,259	2.01	16	10	30	TL	86
A-n33-k6-30	10	*	7,321	7,159	7,309	7,321	0.00	16	10	30	1,240.0	47
A-n34-k5-30	7		7,105	6,980	7,025	7,104	0.01	0	5	22	TL	398
A-n36-k5-30	3		6,944	6,397	6,518	6,519	6.52	3	0	30	TL	31
A-n37-k5-30	5		6,665	5,545	5,585	5,586	19.32	0	0	6	TL	6
A-n37-k6-30	3		7,627	7,170	7,380	7,381	3.33	15	10	30	TL	44
A-n38-k5-30	4		7,225	6,040	6,266	6,267	15.29	13	0	30	TL	37
A-n39-k5-30	7	*	6,401	6,302	6,397	6,401	0.00	3	10	30	2,463.5	38
A-n39-k6-30	8		7,200	6,580	6,794	6,807	5.77	18	10	30	TL	48
A-n44-k7-30	4		7,049	6,476	6,646	6,648	6.03	15	10	30	TL	45
A-n45-k6-30	7		7,167	6,831	7,036	7,036	1.86	23	5	30	TL	40
A-n45-k7-30	6		8,524	8,068	8,215	8,216	3.75	13	10	30	TL	40
A-n46-k7-30	5		7,278	6,406	6,603	6,620	9.94	19	10	30	TL	49
A-n48-k7-30	6		8,131	7,663	7,883	7,884	3.13	13	0	30	TL	36
A-n53-k7-30	6		9,420	6,393	6,487	6,488	45.19	0	0	22	TL	22
A-n54-k7-30	5	*	7,708	7,521	7,675	7,708	0.00	6	10	30	3,001.3	41
A-n55-k9-30	8	*	6,411	6,308	6,400	6,411	0.00	11	10	30	384.0	44
A-n60-k9-30	5		6,614	6,443	6,609	6,609	0.08	15	5	30	TL	40
A-n61-k9-30	5		5,445	5,179	5,384	5,404	0.76	16	10	30	TL	56
A-n44-k7-40	7		8,772	8,584	8,697	8,697	0.86	3	0	30	TL	31
A-n45-k6-40	9		10,246	8,930	9,050	9,051	13.20	0	0	25	TL	25
A-n45-k7-40	7		10,558	10,159	10,200	10,201	3.50	0	0	19	TL	19
A-n46-k7-40	6		8,671	7,905	7,964	7,964	8.88	0	0	12	TL	12
A-n48-k7-40	8		10,007	9,472	9,507	9,507	5.26	0	0	5	TL	5
A-n53-k7-40	9		11,653	8,012	8,024	8,024	45.23	0	0	2	TL	2
A-n54-k7-40	5		11,600	8,872	8,885	8,886	30.54	0	0	8	TL	8
A-n55-k9-40	11		8,657	8,368	8,515	8,534	1.44	14	10	30	TL	61
A-n60-k9-40	5		9,236	8,006	8,037	8,038	14.90	0	0	5	TL	5
A-n61-k9-40	6		8,230	7,248	7,348	7,348	12.00	0	0	28	TL	28
A-n62-k8-40	4		9,417	—	—	—	—	0	0	0	TL	0
A-n63-k9-40	9		12,267	11,370	11,452	11,452	7.12	0	0	24	TL	24
A-n63-k10-40	11		10,580	9,924	10,107	10,108	4.67	13	0	30	TL	36
A-n64-k9-40	10		10,571	9,508	9,557	9,558	10.60	0	0	11	TL	11
A-n65-k9-40	8		9,072	8,285	8,394	8,395	8.06	8	0	30	TL	33
A-n69-k9-40	5		9,523	6,916	6,916	6,917	37.68	0	0	1	TL	1
A-n80-k10-40	10		12,959	11,406	11,538	11,538	12.32	0	0	4	TL	4
B-n41-k6-40	7		9,630	8,666	8,872	8,872	8.54	28	0	30	TL	39
B-n43-k6-40	8		8,648	7,312	7,348	7,349	17.68	0	0	18	TL	18
B-n44-k7-40	8		10,740	9,184	9,380	9,380	14.50	4	0	30	TL	31
A-n53-k7-50	13		11,140	—	—	—	—	0	0	0	TL	0
A-n54-k7-50	7		12,801	11,101	11,101	11,101	15.31	0	0	1	TL	1
A-n55-k9-50	13		10,877	10,577	10,666	10,666	1.98	0	0	21	TL	21
A-n60-k9-50	9		12,540	—	—	—	—	0	0	0	TL	0
A-n61-k9-50	6		10,053	8,740	8,746	8,747	14.93	0	0	4	TL	4
A-n62-k8-50	9		13,483	—	—	—	—	0	0	0	TL	0
A-n63-k9-50	12		14,709	14,117	14,117	14,117	4.19	0	0	2	TL	2
A-n63-k10-50	13		13,068	11,666	11,683	11,683	11.85	0	0	3	TL	3
A-n64-k9-50	10		14,337	—	—	—	—	0	0	0	TL	0
A-n65-k9-50	10		10,580	9,798	9,828	9,829	7.64	0	0	2	TL	2
A-n69-k9-50	6		11,351	—	—	—	—	0	0	0	TL	0
A-n80-k10-50	12		15,950	—	—	—	—	0	0	0	TL	0
B-n50-k7-50	9		—	—	—	—	—	0	0	0	TL	0
B-n50-k8-50	13		15,664	—	—	—	—	0	0	0	TL	0
B-n51-k7-50	10		12,760	10,899	10,912	10,912	16.94	0	0	4	TL	4
B-n52-k7-50	9		—	—	—	—	—	0	0	0	TL	0
B-n56-k7-50	7		—	—	—	—	—	0	0	0	TL	0
B-n57-k7-50	9		14,017	—	—	—	—	0	0	0	TL	0
B-n57-k9-50	10		20,081	15,902	15,902	15,902	26.27	0	0	2	TL	2
B-n63-k10-50	12		14,048	—	—	—	—	0	0	0	TL	0

Table 4.12: Detailed results for $\beta = 3$ and duration minimization.

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
A-n32-k5-20	4	*	5,139	4,996	5,139	5,139	0.00	14	5	30	61.6	44
A-n33-k5-20	3	*	4,320	4,001	4,082	4,320	0.00	10	10	30	771.4	726
A-n33-k6-20	5	*	3,544	3,474	3,500	3,544	0.00	8	0	26	24.2	57
A-n34-k5-20	5		4,430	4,195	4,314	4,418	0.27	30	10	30	TL	2455
A-n36-k5-20	3	*	3,653	3,470	3,607	3,653	0.00	5	10	30	315.8	61
A-n37-k5-20	4	*	3,836	3,674	3,773	3,836	0.00	3	10	30	702.6	160
A-n37-k6-20	3	*	4,089	4,016	4,066	4,089	0.00	4	10	30	46.2	53
A-n38-k5-20	2	*	3,295	3,221	3,293	3,295	0.00	0	8	29	22.7	34
A-n39-k5-20	5	*	3,240	3,110	3,240	3,240	0.00	0	0	26	21.1	27
A-n39-k6-20	6	*	4,627	4,328	4,436	4,627	0.00	13	10	30	210.0	345
A-n44-k7-20	2	*	3,512	3,511	3,512	3,512	0.00	0	0	11	6.9	12
A-n45-k6-20	3	*	3,967	3,753	3,918	3,967	0.00	19	10	30	79.4	64
A-n45-k7-20	3	*	3,840	3,762	3,832	3,840	0.00	2	10	20	29.0	34
A-n46-k7-20	3	*	3,535	3,417	3,479	3,535	0.00	4	10	30	38.7	44
A-n48-k7-20	6	*	4,689	4,494	4,551	4,689	0.00	7	10	30	94.1	153
A-n53-k7-20	5	*	4,309	4,147	4,237	4,309	0.00	13	10	25	145.7	101
A-n54-k7-20	3	*	4,080	3,907	3,996	4,080	0.00	9	10	30	399.9	165
A-n55-k9-20	4	*	3,280	3,266	3,280	3,280	0.00	0	0	6	2.7	7
A-n60-k9-20	3	*	3,470	3,328	3,415	3,470	0.00	9	10	30	106.9	64
A-n61-k9-20	4	*	3,080	3,026	3,066	3,080	0.00	7	10	30	31.2	47
A-n32-k5-30	6		6,468	6,404	6,422	6,459	0.14	1	10	30	TL	364
A-n33-k5-30	4	*	5,063	4,939	4,972	5,063	0.00	5	10	30	1,430.5	297
A-n33-k6-30	10	*	6,028	5,855	5,948	6,028	0.00	12	10	30	971.9	222
A-n34-k5-30	7		6,573	6,267	6,432	6,446	1.97	25	10	30	TL	287
A-n36-k5-30	3	*	4,612	4,516	4,571	4,612	0.00	11	10	30	2,493.1	97
A-n37-k5-30	5		4,738	4,362	4,441	4,456	6.33	11	10	30	TL	47
A-n37-k6-30	3	*	5,357	5,231	5,328	5,357	0.00	10	10	30	453.3	54
A-n38-k5-30	4	*	4,819	4,735	4,785	4,819	0.00	8	10	30	607.3	64
A-n39-k5-30	7		5,520	5,204	5,296	5,338	3.41	35	10	30	TL	194
A-n39-k6-30	8		5,520	5,356	5,418	5,477	0.79	14	10	30	TL	355
A-n44-k7-30	4		5,100	4,736	4,843	4,877	4.57	24	10	30	TL	90
A-n45-k6-30	7	*	5,419	5,300	5,356	5,419	0.00	8	10	30	3,067.5	345
A-n45-k7-30	6	*	6,814	6,763	6,780	6,814	0.00	2	10	30	2,947.3	570
A-n46-k7-30	5	*	4,956	4,832	4,876	4,956	0.00	5	10	30	1,213.3	230
A-n48-k7-30	6		6,362	6,141	6,184	6,256	1.69	12	10	30	TL	100
A-n53-k7-30	6		5,332	4,991	5,081	5,134	3.86	12	10	30	TL	104
A-n54-k7-30	5	*	6,578	6,497	6,564	6,578	0.00	8	10	30	356.0	40
A-n55-k9-30	8	*	5,300	5,193	5,218	5,300	0.00	2	10	28	3,375.0	2807
A-n60-k9-30	5		4,991	4,789	4,829	4,867	2.55	14	10	30	TL	193
A-n61-k9-30	5	*	4,071	3,971	4,050	4,071	0.00	9	10	30	303.6	51
A-n44-k7-40	7		6,832	6,650	6,699	6,729	1.53	7	10	30	TL	81
A-n45-k6-40	9		6,843	6,776	6,805	6,840	0.04	3	10	30	TL	217
A-n45-k7-40	7		8,652	8,342	8,450	8,451	2.38	43	5	30	TL	50
A-n46-k7-40	6		6,130	5,898	5,950	5,968	2.71	20	10	30	TL	76
A-n48-k7-40	8		7,890	7,691	7,735	7,735	2.00	16	5	30	TL	40
A-n53-k7-40	9		6,519	6,418	6,465	6,472	0.73	8	10	30	TL	44
A-n54-k7-40	5		7,762	7,344	7,399	7,400	4.89	25	10	30	TL	48
A-n55-k9-40	11		7,392	7,255	7,298	7,341	0.69	4	10	30	TL	560
A-n60-k9-40	5		6,049	5,880	5,946	5,967	1.37	17	10	30	TL	52
A-n61-k9-40	6		5,737	5,635	5,682	5,695	0.74	16	10	30	TL	128
A-n62-k8-40	4		—	6,262	6,373	6,374	—	21	0	30	TL	37
A-n63-k9-40	9		9,963	9,496	9,562	9,593	3.86	16	10	30	TL	70
A-n63-k10-40	11		8,766	8,556	8,702	8,725	0.47	34	10	30	TL	62
A-n64-k9-40	10		7,927	7,704	7,742	7,771	2.01	11	10	30	TL	78
A-n65-k9-40	8		7,342	7,175	7,316	7,317	0.34	5	10	30	TL	69
A-n69-k9-40	5		5,517	5,121	5,175	5,180	6.51	11	10	30	TL	42
A-n80-k10-40	10		10,122	9,328	9,462	9,462	6.98	19	0	30	TL	39
B-n41-k6-40	7		7,692	7,450	7,611	7,628	0.84	19	10	30	TL	86
B-n43-k6-40	8		7,183	6,797	6,846	6,846	4.92	23	5	30	TL	45
B-n44-k7-40	8		8,367	8,131	8,320	8,332	0.42	13	10	30	TL	84
A-n53-k7-50	13		9,250	8,604	8,785	8,786	5.28	0	0	24	TL	24
A-n54-k7-50	7		9,576	9,108	9,153	9,153	4.62	16	0	30	TL	37
A-n55-k9-50	13		9,640	9,441	9,467	9,491	1.57	5	10	30	TL	127
A-n60-k9-50	9		9,474	9,339	9,392	9,393	0.86	10	10	30	TL	39
A-n61-k9-50	6		7,165	6,847	6,874	6,875	4.22	19	5	30	TL	40
A-n62-k8-50	9		9,880	8,815	8,852	8,852	11.61	0	0	22	TL	22
A-n63-k9-50	12		12,139	11,748	11,790	11,797	2.90	6	10	30	TL	43
A-n63-k10-50	13		10,422	10,156	10,260	10,260	1.58	28	0	30	TL	39
A-n64-k9-50	10		9,825	8,751	8,779	8,779	11.90	6	0	30	TL	32
A-n65-k9-50	10		8,973	8,394	8,507	8,507	5.48	17	0	30	TL	38
A-n69-k9-50	6		—	6,379	6,420	6,421	—	0	0	21	TL	21
A-n80-k10-50	12		11,945	10,611	10,639	10,640	12.27	0	0	15	TL	15
B-n50-k7-50	9		—	6,842	6,861	6,861	—	0	0	18	TL	18
B-n50-k8-50	13		14,674	13,024	13,109	13,110	11.93	0	0	25	TL	25
B-n51-k7-50	10		10,782	9,614	9,897	9,898	8.93	17	0	30	TL	38
B-n52-k7-50	9		—	5,940	5,952	5,952	—	0	0	7	TL	7

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
B-n56-k7-50	7		—	5,389	5,409	5,410	—	0	0	11	TL	11
B-n57-k7-50	9		—	10,189	10,197	10,197	—	0	0	18	TL	18
B-n57-k9-50	10		13,815	13,499	13,576	13,576	1.76	21	5	30	TL	38
B-n63-k10-50	12		11,341	10,830	10,912	10,913	3.92	10	0	30	TL	33

Table 4.13: Detailed results for $\beta = 5$ and duration minimization.

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
A-n32-k5-20	4	*	5,100	4,853	5,086	5,100	0.00	58	10	30	1,712.3	90
A-n33-k5-20	3	*	4,120	3,871	4,003	4,120	0.00	11	10	30	416.6	227
A-n33-k6-20	5	*	3,480	3,392	3,426	3,480	0.00	26	10	30	62.0	76
A-n34-k5-20	5		4,400	4,187	4,307	4,367	0.76	46	10	30	TL	1117
A-n36-k5-20	3	*	3,077	3,033	3,077	3,077	0.00	0	0	25	26.4	26
A-n37-k5-20	4	*	3,742	3,608	3,740	3,742	0.00	14	10	30	755.9	45
A-n37-k6-20	3	*	3,848	3,765	3,837	3,848	0.00	13	10	30	58.1	59
A-n38-k5-20	2	*	2,942	2,931	2,939	2,942	0.00	1	10	17	12.0	23
A-n39-k5-20	5	*	3,208	2,923	3,204	3,208	0.00	3	10	30	85.9	49
A-n39-k6-20	6	*	4,218	4,035	4,107	4,218	0.00	3	10	30	55.8	108
A-n44-k7-20	2	*	3,480	3,423	3,465	3,480	0.00	9	10	30	56.1	49
A-n45-k6-20	3	*	3,866	3,593	3,766	3,866	0.00	18	10	30	598.3	147
A-n45-k7-20	3	*	3,560	3,447	3,463	3,560	0.00	0	8	24	264.3	249
A-n46-k7-20	3	*	3,520	3,302	3,353	3,520	0.00	2	10	30	543.4	529
A-n48-k7-20	6	*	3,934	3,898	3,934	3,934	0.00	0	5	16	4.0	18
A-n53-k7-20	5	*	4,100	3,930	3,992	4,100	0.00	4	10	30	221.8	101
A-n54-k7-20	3	*	3,851	3,503	3,716	3,851	0.00	14	10	30	3,473.9	480
A-n55-k9-20	4	*	3,212	3,151	3,170	3,212	0.00	9	10	30	94.7	91
A-n60-k9-20	3	*	3,082	2,938	3,019	3,082	0.00	5	10	30	90.7	70
A-n61-k9-20	4	*	3,080	2,884	3,027	3,080	0.00	31	7	30	511.0	348
A-n32-k5-30	6		6,215	6,020	6,196	6,201	0.23	41	10	30	TL	97
A-n33-k5-30	4	*	4,723	4,630	4,719	4,723	0.00	14	10	30	395.8	43
A-n33-k6-30	10	*	5,866	5,722	5,850	5,866	0.00	12	10	30	142.2	44
A-n34-k5-30	7		6,600	6,060	6,229	6,243	5.72	21	10	30	TL	295
A-n36-k5-30	3		4,376	4,024	4,096	4,157	5.27	20	10	30	TL	60
A-n37-k5-30	5		4,520	4,156	4,218	4,226	6.96	17	10	30	TL	49
A-n37-k6-30	3		4,953	4,786	4,870	4,901	1.06	12	10	30	TL	89
A-n38-k5-30	4	*	4,434	4,345	4,383	4,434	0.00	11	10	30	623.7	67
A-n39-k5-30	7		5,058	4,979	5,005	5,028	0.60	18	10	30	TL	331
A-n39-k6-30	8		5,330	4,895	4,942	5,093	4.65	9	10	30	TL	726
A-n44-k7-30	4		5,016	4,543	4,679	4,718	6.32	12	10	30	TL	111
A-n45-k6-30	7		5,202	5,091	5,136	5,197	0.10	20	10	30	TL	242
A-n45-k7-30	6		6,461	6,305	6,411	6,424	0.58	23	10	30	TL	227
A-n46-k7-30	5		4,880	4,587	4,643	4,723	3.32	23	10	30	TL	331
A-n48-k7-30	6		5,712	5,513	5,566	5,619	1.66	14	10	30	TL	101
A-n53-k7-30	6		5,340	4,752	4,863	4,927	8.38	21	10	30	TL	80
A-n54-k7-30	5	*	6,140	6,033	6,133	6,140	0.00	13	10	30	536.3	46
A-n55-k9-30	8	*	5,080	4,947	4,955	5,080	0.00	4	10	30	2,697.1	1123
A-n60-k9-30	5	*	4,488	4,398	4,449	4,488	0.00	5	10	30	1,303.5	120
A-n61-k9-30	5		3,800	3,656	3,682	3,743	1.52	7	10	30	TL	326
A-n44-k7-40	7		6,471	6,275	6,307	6,343	2.02	4	10	30	TL	210
A-n45-k6-40	9		6,488	6,431	6,461	6,474	0.22	12	10	30	TL	66
A-n45-k7-40	7	*	7,611	7,558	7,596	7,611	0.00	10	10	30	3,353.7	116
A-n46-k7-40	6		5,748	5,530	5,555	5,588	2.86	12	10	30	TL	90
A-n48-k7-40	8		7,168	6,944	7,052	7,062	1.50	17	10	30	TL	44
A-n53-k7-40	9		6,320	6,203	6,238	6,251	1.10	13	10	30	TL	49
A-n54-k7-40	5		6,869	6,744	6,794	6,799	1.03	12	10	30	TL	44
A-n55-k9-40	11		7,040	6,979	6,993	7,038	0.03	0	10	30	TL	667
A-n60-k9-40	5		5,362	5,265	5,307	5,332	0.56	18	10	30	TL	59
A-n61-k9-40	6		5,271	5,190	5,207	5,256	0.29	8	10	30	TL	147
A-n62-k8-40	4		—	5,655	5,697	5,697	—	15	10	30	TL	40
A-n63-k9-40	9		9,076	8,736	8,783	8,819	2.91	23	10	30	TL	69
A-n63-k10-40	11		8,433	8,141	8,325	8,349	1.01	19	10	30	TL	115
A-n64-k9-40	10		7,715	7,414	7,436	7,469	3.29	12	10	30	TL	108
A-n65-k9-40	8		7,260	6,922	7,161	7,161	1.38	36	0	30	TL	58
A-n69-k9-40	5		5,136	4,670	4,762	4,794	7.13	6	10	30	TL	55
A-n80-k10-40	10		9,877	9,120	9,264	9,266	6.59	24	10	30	TL	46
B-n41-k6-40	7	*	6,724	6,681	6,724	6,724	0.00	21	10	30	525.1	45
B-n43-k6-40	8		6,609	6,505	6,537	6,542	1.02	14	10	30	TL	49
B-n44-k7-40	8		8,035	7,558	7,851	7,856	2.28	23	10	30	TL	61
A-n53-k7-50	13		9,001	8,345	8,558	8,558	5.18	12	0	30	TL	34
A-n54-k7-50	7		8,685	8,400	8,450	8,450	2.78	19	5	30	TL	39
A-n55-k9-50	13		9,179	9,090	9,107	9,136	0.47	11	10	30	TL	168
A-n60-k9-50	9		8,873	8,697	8,729	8,729	1.64	25	10	30	TL	48
A-n61-k9-50	6		6,337	6,214	6,227	6,241	1.52	10	10	30	TL	47
A-n62-k8-50	9		—	8,154	8,171	8,171	—	8	0	30	TL	36

Instance	#To	Opt	UB	LB_{LP}	LB_{cut}	LB	Gap	#CC	#SR	$\#dyn_{ng}$	Time	#BB
A-n63-k9-50	12		11,460	10,914	11,002	11,002	4.16	21	10	30	TL	47
A-n63-k10-50	13		10,266	9,733	9,779	9,779	4.97	31	0	30	TL	42
A-n64-k9-50	10		—	8,335	8,349	8,349	—	20	10	30	TL	41
A-n65-k9-50	10		8,606	8,082	8,252	8,252	4.29	18	5	30	TL	40
A-n69-k9-50	6		—	5,928	5,954	5,954	—	0	0	28	TL	28
A-n80-k10-50	12		11,559	9,993	10,171	10,171	13.65	12	0	30	TL	35
B-n50-k7-50	9		—	6,740	6,756	6,756	—	3	0	30	TL	31
B-n50-k8-50	13		13,152	12,425	12,583	12,583	4.51	15	0	30	TL	34
B-n51-k7-50	10		—	9,492	9,797	9,797	—	23	5	30	TL	41
B-n52-k7-50	9		—	5,365	5,393	5,393	—	0	0	19	TL	19
B-n56-k7-50	7		4,955	4,775	4,787	4,787	3.49	0	0	22	TL	22
B-n57-k7-50	9		—	9,912	9,958	9,958	—	10	0	30	TL	34
B-n57-k9-50	10		—	13,215	13,324	13,324	—	32	0	30	TL	42
B-n63-k10-50	12		11,182	10,231	10,343	10,343	8.10	28	0	30	TL	38

Chapter 5

The Vehicle Routing Problem with Drones and Time Windows: Minimizing Route Duration

Jeanette Schmidt

Abstract

The vehicle routing problem with drones and time windows (VRP-DTW) is a generalization of the vehicle routing problem with drones, in which each customer is associated with a predefined time window that specifies the delivery time for the customer's shipment. Most literature on the VRP-DTW considers the minimization of the total routing cost as objective function, which is the standard objective for vehicle routing problems. However, the presence of time windows gives rise to the analysis of alternative objective functions, such as minimizing the completion time or route duration. We present a branch-price-and-cut algorithm to solve the VRP-DTW with the objective to minimize the sum of route durations over all routes. The column-generation pricing problem is modeled as a shortest path problem with resource constraints (SPPRC) that can handle the implications of non-robust cuts and branching decisions. The SPPRC is solved using an effective dynamic programming labeling algorithm on an artificial network. Non-robust capacity cuts and dynamic ng-neighborhood extensions are used to strengthen the linear relaxation. In a computational study, we investigate the algorithmic components and show that the proposed algorithm can solve instances with up to 30 customers to proven optimality within two hours of computation time. The remaining gap between primal and dual bound over all considered instances is less than 0.5% on average. We also present managerial insights that analyze the impact of combined truck-and-drone routing compared to classical truck routing, and show that minimizing the route duration leads to considerable time savings compared to minimizing the completion time.

5.1 Introduction

In synchronized truck-and-drone routing problems, one or several trucks are equipped with a single or multiple *unmanned aerial vehicles* (UAVs, in the following referred to as *drones*) to fulfill a set of transportation requests. Drones can usually reach customers faster, easier, and more environmentally friendly than a truck, but they have the disadvantages of a limited flying range and payload, i.e., they cannot serve long-haul transportation requests or deliver heavy packages (Joerss *et al.*, 2016; Goodchild and Toy, 2018). However, the combination of two different types of vehicles with complementary characteristics allows the reduction of delivery times and transportation costs, can simplify access to certain customers, and increases the efficiency of performing transportation requests (Wang *et al.*, 2017; Poikonen *et al.*, 2017; Carlsson and Song, 2018). Thus, this topic attracts the research community (Otto *et al.*, 2018; Chung *et al.*, 2020; Macrina *et al.*, 2020; Moshref-Javadi and Winkenbach, 2021; Madani and Ndiaye, 2022) as well as several delivery companies such as Amazon or UPS (Amazon Prime Air, 2023; Federal Aviation Administraion, 2023).

The (basic) *vehicle routing problem with drones* (VRP-D) is one example of a synchronized truck-and-drone routing problem. Herein, a fleet of homogeneous trucks –each equipped with a single drone– is stationed at a depot and can be used to serve a given set of customers. Therefore, a truck and its drone can operate either together or separately. When operating *together*, the drone is inside or on top of the truck, and the truck is responsible for serving the customer. Whenever it is beneficial, the truck can release its drone (at a customer location or the depot) so that both vehicles can serve customers in parallel. While the drone is airborne, the truck continues its route *alone* to serve one or several customers. Meanwhile, the drone serves exactly one customer and *returns* to the truck. In the basic version, it is assumed that the assignment of trucks and drones is fixed, i.e., a drone always has to return to the truck from which it was released. Additionally, a drone is not allowed to *loop*, i.e., the truck must continue its journey while the drone is airborne. The VRP-D aims to find a cost-minimal set of truck-and-drone routes that start and end at the depot, visit each customer exactly once, and respect the capacity of the truck and drone on each route (see Chapter 4). The basic VRP-D can be generalized in several ways, e.g., by relaxing the loop constraint (Zhou *et al.*, 2023), by relaxing the fixed assignment of trucks and drones (Bakir and Tiniç, 2020), or by limiting the flying range of the drones (Zhen *et al.*, 2023).

In this paper, we consider the *vehicle routing problem with drones and time windows* (VRP-DTW), which is a restricted variant of the VRP-D where the service at each customer must start within a predefined time interval, called a *time window*. An additional time window at the depot spans the planning horizon and limits the departure and arrival times for truck and drone. Time windows are

present in many routing problems, as they can represent real-life situations such as working hours, opening hours, or agreed delivery times. The number of publications concerning a routing problem with time windows is huge, but there are only a few publications considering time windows in a VRP-D variant. Almost all of these publications aim at solving the VRP-DTW with the objective to minimize the total routing cost, including different cost components such as fixed and variable costs for trucks and/or drones, costs for waiting times, or CO_2 emissions. However, the introduction of time windows allows the investigation of alternative objective functions such as minimizing the *completion time* or the *route duration* (Savelsbergh, 1992). When considering the completion time of a route, truck and drone leave the depot directly upon opening the time window to start the delivery process, generating an *as-early-as-possible schedule*. In contrast, when considering the route duration, the departure time at the depot is not fixed. Both types of vehicles can depart at any time within the depot's time window to deliver the customers, creating an *as-late-as-possible schedule*. Minimizing the completion time may not always be appropriate for truck-and-drone routing, as it may result in long waiting times at customer locations. Especially, if drones have a limited flying range (in the sense that they cannot be separated from the truck longer than a specific amount of time) long waiting times can make a drone flight infeasible.

We develop a BPC algorithm to solve the VRP-DTW with the objective to minimize the sum of route duration over all routes. Our BPC algorithm is based on a set-partitioning formulation, as usually done in BPC approaches for vehicle routing problems. The column-generation pricing problem is modeled as a shortest path problem with resource constraints (SPPRC, Irnich and Desaulniers, 2005) and solved by means of a dynamic programming labeling algorithm. The labeling algorithm runs on a *multi-digraph*, originally proposed by Roberti and Ruthmair (2021), in which vertices represent possible truck-and-drone positions and arcs represent potential movements of truck and drone. To track Pareto-optimal schedules, we use resources that are pairwise interdependent and coupled with a max-term (Irnich, 2008; Tilk and Irnich, 2017).

Our contributions are as follows:

- To the best of our knowledge, we developed the first exact algorithm to solve the VRP-DTW with the objective to minimize the sum of route duration over all routes contained in a solution. We equip our BPC-algorithm with non-robust capacity cuts (Baldacci *et al.*, 2008) and dynamic *ng*-neighborhood extensions (Roberti and Mingozzi, 2014; Bode and Irnich, 2015) to strengthen the linear relaxation of the set-partitioning formulation. Integer solutions are obtained by means of a four-stage branching strategy. Our SPPRC pricing problem provides the flexibility to handle all implications of the non-robust cuts and branching decisions.

- Synchronizing truck and drone to obtain a route with minimal duration is a non-trivial task. Neither the as-late-as-possible schedule nor the as-early-as-possible schedule are appropriate. We propose a method on how to properly synchronize both types of vehicles to obtain a route with minimal duration.
- We present an extensive computational study on newly generated instances that evaluates the algorithmic components and demonstrates the efficacy of the overall BPC algorithm. Finally, our algorithm is able to solve instances with up to 30 customers to proven optimality within a time limit of two hours. The remaining gap between primal and dual bound over all instances is less than 0.5% on average.
- We also provide managerial insights that analyze the impact of combined truck-and-drone routing compared to classical truck routing, as well as the impact of different types of drones that differ in their flying range. In addition, we analyze the time savings that can be achieved by minimizing the route duration rather than the completion time.

The remainder of this paper is organized as follows: At first, we provide a brief overview of the related scientific literature in Section 5.2. Afterwards, we formally define the VRP-DTW in Section 5.3. Section 5.4 describes the BPC algorithm to solve the VRP-DTW with a specific focus on the solution of the pricing problem. Computational results and managerial insights are presented in Section 5.5. Finally, in Section 5.6, we conclude and suggest future research directions.

5.2 Literature Review

Synchronized truck-and-drone routing is a popular research topic. The review articles by Otto *et al.* (2018); Chung *et al.* (2020); Macrina *et al.* (2020); Moshref-Javadi and Winkenbach (2021); Madani and Ndiaye (2022) show the huge amount of publications in the last years and new publications continue to appear. It would go far beyond the scope of this paper to list them all. Therefore, we focus only on publications that consider a variant of the VRP-DTW. For a review of VRP-D variants, we refer the reader to Chapter 4, and for a more general overview of (synchronized) truck-and-drone routing, we refer to one of the review articles mentioned above.

The first work considering time windows in a VRP-D variant was published by Pugliese and Guerriero in 2017. The authors developed a mixed-integer linear program (MIP) to solve the VRP-DTW with multiple drones per truck with the objective of minimizing the total routing cost. The commercial MIP solver CPLEX can solve small instances of five and ten customers in a matter of seconds. Their work was later extended by Pugliese *et al.* (2020), in which the authors investigated

several variants of the problem, such as relaxed time window constraints for each customer served by a drone, or a limit on the number of drones associated with each truck. In addition to a MIP formulation, the authors developed a heuristic algorithm that embeds a two-phase strategy in a multi-start framework. The MIP can be solved to proven optimality for instances with up to 15 customers within a computation time of 30 minutes. The multi-start heuristic can find almost all optimal solutions obtained by the MIP in a considerably shorter computation time. In addition, large instances with up to 100 customers are solved heuristically. In a second follow-up paper, Pugliese *et al.* (2021) focused on different energy consumption functions in case of adverse weather conditions while a drone is airborne. They proposed a Benders' decomposition approach to solve the VRP-DTW under uncertain energy consumption and presented results on instances with 10 and 15 customers. Coindreau *et al.* (2021) investigated a version of the VRP-DTW that allows a drone to loop. They developed a MIP formulation and an adaptive large neighborhood search (ALNS) to minimize the total routing cost. CPLEX solves instances with up to 20 customers within a time limit of 10 hours, while the proposed ALNS solves instances with up to 100 customers in less than a minute of computation time. In the publication by Das *et al.* (2021), each customer can have multiple, prioritized time windows that are assumed to be soft. They developed an ant colony optimization heuristic and a genetic algorithm, both aimed at solving a multi-objective function. More precisely, the heuristics aim to minimize the total routing cost and maximize the customer service level in terms of on-time deliveries. Both heuristics can solve instances with 25 and 50 customers from the Solomon benchmark set (Solomon, 1987) in less than 10 minutes of computation time. Kuo *et al.* (2022) presented another MIP formulation and a variable neighborhood search (VNS) to solve the VRP-DTW with the objective to minimize the total routing cost. The Gurobi MIP solver solves instances with a maximum of 10 customers within a computation time of two hours. The VNS is used to tackle larger instances with up to 50 customers within a time limit of 10 minutes.

To the best of our knowledge, Li and Wang (2022) developed the first exact BPC algorithm to solve the VRP-DTW with the objective to minimize the total routing cost. It solves instances with up to 50 customers to proven optimality within a time limit of 20 hours. They also combine the BPC algorithm with an ALNS to tackle large-sized instances with up to 100 customers. The time limit for the combined BPC algorithm is set to 24 hours. Another exact algorithm was proposed by Yin *et al.* (2023). They consider a variant of the VRP-DTW in which the drone is allowed to serve more than one customer, as long as it does not exceed its capacity or maximum flying range. They developed a full-fledged BPC algorithm that uses a bidirectional labeling algorithm to solve the pricing problem to minimize the total routing cost. Instances with up to 35 (45) customers can be

solved within a computation time of two (three) hours.

Li *et al.* (2020) and Zhou *et al.* (2023) studied another variant of the VRP-DTW, in which each truck is equipped with more than one drone. Whenever the truck releases one or several drones at a customer's location, it must wait at that customer's location until all the launched drones have returned. The service at the next customer is performed subsequently. In such a setting, the synchronization constraints between a truck and its drones are different compared to all VRP-DTW variants mentioned above. Li *et al.* (2020) presented a MIP formulation together with an ALNS. Small instances with up to 12 customers can be solved with CPLEX within four hours of computation time. For the same set of instances, the proposed ALNS finds the optimal solution in less than 10 seconds. Additionally, the authors run their ALNS algorithm on instances with up to 100 customers and obtain solutions in less than 1500 seconds. Zhou *et al.* (2023) developed a BPC algorithm that uses a bidirectional labeling algorithm to solve the pricing problem. Instances with up to 35 customers can be solved to proven optimality within a time limit of three hours.

The works of Chen *et al.* (2021a) and Chen *et al.* (2021b) consider delivery robots instead of drones, which share many similarities with the VRP-DTW variant proposed in Li *et al.* (2020) and Zhou *et al.* (2023). Chen *et al.* (2021a) developed an ALNS to solve this problem heuristically. The authors show that their algorithm is competitive with other heuristics from the literature on the Solomon benchmark set (Solomon, 1987) and on the self-generated *Cardiff* instance set with up to 200 customers. Chen *et al.* (2021b) proposed a MIP formulation together with a matheuristic. Instances with up to 50 customers can be solved within a time limit of 30 minutes.

5.3 The Vehicle Routing Problem with Drones and Time Windows

The VRP-DTW that we consider in the following is defined on a complete directed graph $G = (V, A)$ with vertex set V and arc set A. The vertex set $V = N \cup \{0, 0'\}$ comprises the set of *customers* N and the *start* and *end depot* $\{0, 0'\}$. Associated with each customer $v \in N$ is an integer *demand* $q_v > 0$ and a *time window* $[e_v, l_v]$ in which the *service* of length $\sigma_v > 0$ has to start. We consider the time windows to be hard, i.e., if a vehicle (independent of its type) arrives earlier than e_v to serve customer $v \in N$ the start of the service is delayed to time e_v. It is not allowed to start the service later than l_v. The time windows at the start and end depot $[e_0, l_0] = [e_{0'}, l_{0'}]$ represent the planning horizon. W.l.o.g, we assume that there is no service and demand at the depot, i.e., $q_0 = q_{0'} = \sigma_0 = \sigma_{0'} = 0$. The

depot houses a homogeneous fleet of K trucks. Each truck has a *capacity* Q and is equipped with a single drone. The *capacity of a drone* is limited by Q^{dr} and in the sense that it can only serve one customer v with $q_v \leq Q^{\mathrm{dr}}$ before returning to the truck. Each drone has a maximum *flying range* δ, i.e., the drone cannot remain airborne for more than δ time units. We assume that a drone always waits on the ground rather than hovering, so there is no energy consumption while waiting that affects its flying range. Both types of vehicles can be used to serve the set of customers. For this purpose, each truck and its drone can operate together or separately. To operate *together*, the drone is either inside or on top of the truck while the truck *serves* the customer. To operate *separately*, the truck can release a drone at any customer location (or at the start depot) $v \in N \cup \{0\}$. While the drone is airborne, the truck continues its journey *alone* to serve one or several customers. After serving exactly one customer, the drone must *return* to the same truck from which it was released, at the truck's current location $w \in N \cup \{0'\}$ with $w \neq v$. We assume that take-off and landing of a drone is completely autonomous and does not require any interaction from the truck driver. This allows for take-off and landing at any time, even while the truck is serving a customer. In addition, take-off and landing are independent of the customer time windows $[e_v, l_v]$ and $[e_w, l_w]$ at the release and landing positions v and w with $w \neq v$, as they are only relevant for the truck serving that customers. The drone can take off as soon as the truck arrives at a customer, without waiting for the truck to finish its service or for the time window to open.

Associated with each arc $(v, w) \in A$ is a *travel time* $t_{vw} > 0$ for the truck and $t^{\mathrm{dr}}_{vw} > 0$ for the drone. It is assumed that the triangle inequality holds for all travel times t_{vw} and t^{dr}_{vw}, respectively. Times for take-off and landing are considered to be negligible.

A *route* $r = (P, D)$ is defined as a pair of a *truck path* P and a sequence of (possibly empty) *drone subpaths* D together with corresponding *time schedules* $S = (T(P), U(D))$. A truck path $P = (0, v_1, \ldots, v_m, 0')$ with $m \geq 1$ is a path in G, starting at the start depot 0, visiting a sequence of customers $N(P) = \{v_1, \ldots, v_m\}, N(P) \subseteq N$, and returning to the end depot $0'$. The drone subpaths $D = (D^1, \ldots, D^b)$ specify unique drone flights along the truck path P. Each drone subpath $D^s = \langle v^s, k^s, w^s \rangle$ for all $s \in \{1, \ldots, b\}$ is represented by a triplet: at $v^s \in \{0\} \cup N$ the drone leaves the truck, $k^s \in N$ specifies the customer that is served by the drone, and $w^s \in N \cup \{0'\}$ indicates where truck and drone meet each other again.
For each drone subpath D^s, we define two indices $g_s, h_s \in \{0, 1, \ldots, m, m+1\}$ with $g_s < h_s$. For convenience, the set $I(s) = \{g_s, g_s + 1, \ldots, h_s - 1\}$ defines the indices of the truck path P where the drone leaves the truck (g_s) and the truck serves a customer alone ($g_s + 1, \ldots, h_s - 1$). Note that the position where the drone

returns to the truck is not included. For two drone subpaths $D^s, D^{s'}$ with $s, s' \in \{1, \ldots, b\}$ $g_s < h_s \leq g_{s'} < h_{s'}$ must hold true. For each truck-and-drone path, there exist a corresponding time schedule. The schedule $T(P) = (T_0, T_1, \ldots, T_m, T_{0'})$ with $m \geq 1$ represents the service start times at each customer of $N(P)$ together with the departure and arrival times at the start and end depot, respectively. Additionally, there exists a time schedule $U(D) = (U(D^1), \ldots, U(D^b))$ for each drone subpath. Each drone time schedule $U(D^s) = \langle U_{v^s}, U_{k^s}, U_{w^s} \rangle$ for all $s \in \{1, \ldots, b\}$ is represented by an additional triplet: the drone takes off at time U_{v^s} from its current position v^s, the service at customer k^s starts at U_{k^s}, and the drone meets the truck at time U_{w^s}. Recall that by the above given definition of the VRP-DTW, U_{v^s} and U_{w^s} are not restricted by the time windows of both customers v^s and w^s.

A route $r = (P, D)$ is *elementary* if all elements in $N(P)$ and all k^s for all $s \in \{1, \ldots, b\}$ are different and it is *feasible* if the following conditions hold:

(i) $\sum_{p=1}^{m} q_{v_p} + \sum_{p=s}^{b} q_{k^s} \leq Q$;

(ii) $q_{k^s} \leq Q^{\mathrm{dr}}$ for all $s \in \{1, \ldots, b\}$;

(iii) $T_p \in [e_{v_p}, l_{v_p}]$ for all $p \in \{0, \ldots, m+1\}$;

(iv) $U_{k^s} \in [e_{k^s}, l_{k^s}]$ for all $s \in \{1, \ldots, b\}$;

(v) $T_{p-1} + \sigma_{p-1} + t_{v_{p-1}, v_p} \leq T_p$ for all $p \in \{1, \ldots, m+1\}$;

(vi) $U_{v^s} + t^{\mathrm{dr}}_{v^s, k^s} \leq U_{k^s}$ and $U_{k^s} + \sigma_{k^s} + t^{\mathrm{dr}}_{k^s, w^s} \leq U_{w^s}$ for all $s \in \{1, \ldots, b\}$;

(vii) $t^{\mathrm{dr}}_{v^s, k^s} + t^{\mathrm{dr}}_{k^s, w^s} \leq \delta$ for all $s \in \{1, \ldots, b\}$;

(viii) $I(s) \cap I(s') = \varnothing$ for $1 \leq s < s' \leq b$ with $b \geq 2$;

(ix) $T_{g_{s-1}} + \sigma_{g_{s-1}} + t_{g_{s-1} g_s} \leq U_{v^s}$ for all $s \in \{1, \ldots, b\}$;

(x) $U_{v^s} \leq T_{g_{s+1}} - t_{g_s g_{s+1}}$ for all $s \in \{1, \ldots, b\}$;

(xi) $U_{w^s} \leq T_{v_{h+1}} - t_{v_h, v_{h+1}}$ for all $s \in \{1, \ldots, b\}$.

These conditions ensure that each route respects the capacity of the truck and the drone ((i) and (ii)), the customer time windows ((iii) and (iv)), and travel times ((v) and (vi)). Condition (vii) ensures that a drone is never airborne longer than allowed. Additionally, constraints (viii) to (xi) ensure a synchronization of truck and drone in space (viii) and time ((ix) to(xi)).

The *route duration* is defined by the *time span* between the first departure x of truck or drone at the start depot, i.e., $x = \min(T_0, U_{v^1})$, and the arrival of the

second vehicle at the end depot, i.e., $y = \max(T_{0'}, U_{w^d})$. Therefore, the duration of a route r is defined as $d_r = y - x$.

The VRP-DTW asks for a set of at most K feasible routes such that each customer is served by a single vehicle and the total duration over all routes $\sum_{r\in\Omega} d_r$ is minimized.

Example 10 *In the following, we consider a feasible route* $r = (P, D)$ *with* $P = (0, 1, 2, 3, 5, 0')$ *and* $D = (\langle 2, 4, 3\rangle, \langle 3, 6, 0'\rangle)$. *Time windows* $[e_v, l_v]$ *along with travel times* t_{vw} *and* t^{dr}_{vw} *are given in Table 5.1 (columns 2+3 and 7+8). Further, we assume a demand* $q_v = 2$ *for all* $v \in N = \{1, \ldots, 6\}$ *and a capacity* $Q = 12$ *for the truck. The drone's capacity is set to* $Q^{dr} = 5$ *and its flying range to* $\delta = 5$ *so that each customer can be served by a drone. The service time for each customer* $v \in N$ *is set to* $\sigma_v = 1$.

P	$[e_v, l_v]$	t_{vw}	$T(P)_{alap}$	$T(P)_{aeap}$	D	$[e_{k^s}, l_{k^s}]$	t^{dr}_{vw}	$U(D)_{alap}$	$U(D)_{aeap}$
0	$[0, 25]$	1	5	0	2	—	2	10	7
1	$[3, 6]$	3	6	3	4	$[10, 12]$	3	12	10
2	$[8, 10]$	3	10	8	3	—	—	16	14
3	$[12, 14]$	3	14	12					
5	$[17, 20]$	3	19	17	3	—	2	16	14
$0'$	$[0, 25]$	—	23	21	6	$[18, 20]$	3	18	18
					$0'$	—	—	22	22

Table 5.1: VRP-DTW instance with a feasible solution.

The columns $T(P)_{alap}$ *and* $U(D)_{alap}$ *show the truck-and-drone schedules for the given route according to an as-late-as-possible schedule. The truck leaves the depot at time 5 for customer 1. It arrives there at time 6, which is the latest possible time to start service. After completing the service, the truck drives towards customer 2, where it arrives at time 10, which is again the latest possible time to start the service. Now, truck and drone separate. The truck continues its journey alone to customer 3, arrives at time 14 and performs service until time 15. In parallel, the drone takes off from customer 2, serves customer 4, and meets the truck at customer 3 at time 16. Note that the drone is allowed to meet the truck later than* $l_3 = 14$ *because it is not restricted to that time window. The truck has to wait for the drone, before they separate again. The truck continues serving customer 5 at time 19 and arrives at the depot at time 23. Meanwhile, the drone serves customer 6 and arrives at the depot* $0'$ *at time 22. The duration of this route is* $\max(23, 22) - 5 = 18$.

In contrast, an as-early-as-possible schedule can be determined as follows: The truck leaves the depot immediately at time $e_0 = 0$ *and arrives at customer 1 at*

time 1. It must wait until time 3 to start service. The truck then continues to customer 2, where it has to wait for one time unit to start service. While waiting for service, the drone is permitted to take off to customer 4. Note that the drone is allowed to do so because it is not restricted to the time window at customer 2. Starting at time 7, serving customer 4 at time 10 (including one unit of waiting time), the drone lands at customer 3 at time 14. While the drone is airborne, the truck serves customers 2 and 3 and waits for one time unit for the drone to arrive. After both vehicles are synchronized, the truck continues to customer 5 and $0'$, where it arrives at time 21. The drone arrives at customer 6 at time 16 and has to wait for 2 time units to start service at time 18. After serving the customer, the drone reaches the depot at time 22. The completion time, according to an as-early-as-possible schedule is $\max(21, 22) = 22$*. The example shows that starting later than e_0 reduces the duration by 4 time units, which is a time saving of approx. 18%.*

5.4 Branch-Price-and-Cut Algorithm

This section describes the BPC algorithm for the solution of the VRP-DTW. Section 5.4.1 presents a straightforward set-partitioning formulation that will be used as the *master problem* (MP) in the column-generation process. Section 5.4.2 discusses the modeling and solution of the column-generation subproblem. In particular, we present the underlying network, discuss the labeling algorithm, and present effective dominance rules in Section 5.4.3. Afterwards, Section 5.4.4 shows implemented acceleration techniques for the overall BPC algorithm. Valid inequalities used to strengthen the linear relaxation of the MP are presented in Section 5.4.5. Finally, Section 5.4.6 discusses the branching rules to obtain integer solutions.

5.4.1 Route-based Formulation

Let Ω denote the set of all VRP-DTW-routes that are feasible with respect to the capacity, time window, and synchronization constraints. Further, let d_r denote the duration of a route r as defined in Section 5.3 and λ_r be a binary variable equal to 1 if and only if the route $r \in \Omega$ is selected in the solution. The coefficient a_{vr} indicates the number of times customer $v \in N$ is served by route r. The VRP-DTW can be stated as follows:

$$\min \quad \sum_{r \in \Omega} d_r \lambda_r \tag{5.1a}$$

$$\text{subject to} \quad \sum_{r \in \Omega} a_{vr} \lambda_r = 1 \qquad \forall v \in N \tag{5.1b}$$

$$L \leq \sum_{r \in \Omega} \lambda_r \leq K \tag{5.1c}$$

$$\lambda_r \in \{0, 1\} \qquad \forall r \in \Omega \tag{5.1d}$$

The objective function (5.1a) minimizes the total duration over all routes. The set-partitioning constraints (5.1b) ensure that each customer is served exactly once. Due to (5.1c), the number of trucks in use ranges between a lower bound (L, see Section 5.4.6) and the fleet size K. The domain of all route variables is specified in (5.1d).

The model defined by (5.1) contains a huge number of feasible routes so that it is almost impossible (and not necessary) to generate all such routes beforehand. Instead, the linear relaxation of this model, i.e., constraints (5.1d) are replaced by $\lambda_r \geq 0$, is solved by means of *column generation* (Desaulniers *et al.*, 2005). Its basic idea is to work with a *restricted master problem* (RMP) that considers only a (small) subset of feasible routes. It can be obtained by replacing Ω with a subset $\bar{\Omega} \subset \Omega$ of generated routes. Missing routes are generated by solving the column-generation *subproblem*, also called *pricing problem*. A solution of the linear relaxation of the MP can be obtained by iteratively re-optimizing the RMP and solving the pricing problem(s), until no more beneficial routes, i.e., routes with negative reduced cost, can be added to the RMP.

5.4.2 Column Generation

The column-generation pricing problem aims to generate beneficial routes or to prove that no such route exists. A route is beneficial if it is feasible and its reduced cost $\tilde{c}_r$ is negative. Let $(\pi_v)_{v \in N}$ denote the dual prices of the set-partitioning constraints (5.1b) and let μ be the sum of the dual prices of the two fleet-size constraints (5.1c), positive or negative depending on whether $L \leq \sum_{r \in \Omega} \lambda_r$ or $\sum_{r \in \Omega} \lambda_r \leq K$ is constraining. The *reduced cost* of an arbitrary route $r \in \Omega$ can be calculated as

$$\tilde{c}_r = d_r - \sum_{v \in N} a_{vr} \pi_v - \mu.$$

This pricing problem can be modeled as *shortest path problem with resource constraints* (SPPRC, Irnich and Desaulniers, 2005) and solved with a labeling algorithm.

Artificial Network

We run our labeling algorithm on an artificial network originally introduced by Roberti and Ruthmair (2021) to solve the *traveling salesman problem with drone*

(TSP-D). Chapter 4 shows that this network is also suitable to solve the multiple-vehicle version of this problem, the VRP-D. For the sake of completeness, we explain the most important characteristics of this network and refer the reader to Roberti and Ruthmair (2021) or Chapter 4 for further details. From now on, we stay close to the notation used in Chapter 4.

The *artificial network* $\mathcal{N} = (\mathcal{W}, \mathcal{A})$ contains a set of vertices $\mathcal{W}$ and arcs $\mathcal{A}$. The vertex set

$$\mathcal{W} = \{(0,0) \cup (\{0\} \cup N) \times (N \cup \{0'\}) \cup (0',0')\} \subset V \times V$$

represents all possible truck-and-drone positions. Each vertex $i = (i^{\text{tr}}, i^{\text{dr}}) \in \mathcal{W}$ contains two pieces of information: the current truck position i^{tr} and the current drone position i^{dr}.

The arc set $\mathcal{A}$ represents movements of truck and drone between two vertices $i = (i^{\text{tr}}, i^{\text{dr}})$ and $j = (j^{\text{tr}}, j^{\text{dr}})$ with $i, j \in \mathcal{W}, i \neq j$ together with an (possible) additional service by the drone, denoted by $k \in N^{dr} = \{v \in N : q_v \leq Q^{\text{dr}}\} \cup \{\perp\}$, where $\perp$ represents that no drone service is performed. Since the artificial network is a directed *multi-graph*, each arc $\mathcal{A}$ can be uniquely described by a triplet $[i, j, k] = [(i^{\text{tr}}, i^{\text{dr}}), (j^{\text{tr}}, j^{\text{dr}}), k]$. All arcs of the artificial network $\mathcal{N}$ can be divided into three disjoint categories:

(i) *together arcs*

$$\mathcal{A}^{tog} = \left\{[i, j, \perp] \in \mathcal{W} \times \mathcal{W} \times \{\perp\} : i^{\text{tr}} = i^{\text{dr}} \neq j^{\text{tr}} = j^{\text{dr}}\right\}$$

represent that truck and drone are moving together from $i^{\text{tr}} = i^{\text{dr}}$ to a new position $j^{\text{tr}} = j^{\text{dr}}$;

(ii) *truck alone arcs*

$$\mathcal{A}^{alone} = \left\{[i, j, \perp] \in \mathcal{W} \times \mathcal{W} \times \{\perp\} : i^{\text{dr}} = j^{\text{dr}} \text{ and } i^{\text{tr}} \neq j^{\text{tr}} \text{ and } i^{\text{dr}} \neq j^{\text{tr}}\right\}$$

represent movements of the truck from customer i^{tr} to another customer j^{tr} while the drone is airborne (release position $i^{\text{dr}} = j^{\text{dr}}$); and

(iii) *drone arcs*

$$\mathcal{A}^{drone} = \left\{[i, j, k] \in \mathcal{W} \times \mathcal{W} \times N^{dr} : i^{\text{tr}} = j^{\text{tr}} = j^{\text{dr}} \neq i^{\text{dr}} \text{ and } k \notin \{i^{\text{dr}}, i^{\text{tr}}, \perp\}\right\}$$

represent drone movements. Here, the drone starts from its release position i^{dr} to serve customer k and meets the truck at the truck's current position $j^{\text{tr}} = j^{\text{dr}} \neq i^{\text{dr}}$.

Every path in $\mathcal{N}$, starting at $(0,0)$, visiting a sequence of vertices, and ending at $(0',0')$ represents a joint truck-and-drone route. Note that this route is not necessarily elementary or feasible. The labeling algorithm presented in Section 5.4.2 ensures final elementary and feasibility.

Pre-Processing Before running the labeling algorithm on $\mathcal{N}$, the graph can be pre-processed by removing all capacity and time window infeasible arcs that cannot be part of a feasible solution. First, if customers i and j are both served by a truck, we check if serving customer j after customer i is feasible with respect to their time windows. Otherwise the arc $[i, j, \perp]$ can be eliminated from $\mathcal{A}$ so that

$$\mathcal{A}^{tog} \subset \{[i, j, \perp] \in \mathcal{W} \times \mathcal{W} \times \{\perp\} : e_{i^{\text{tr}}} + \sigma_{i^{\text{tr}}} + t_{i^{\text{tr}} j^{\text{tr}}} \leq l_{j^{\text{tr}}}\} \text{ and}$$

$$\mathcal{A}^{alone} \subset \{[i, j, \perp] \in \mathcal{W} \times \mathcal{W} \times \{\perp\} : e_{i^{\text{tr}}} + \sigma_{i^{\text{tr}}} + t_{i^{\text{tr}} j^{\text{tr}}} \leq l_{j^{\text{tr}}}\}.$$

Unfortunately, it is not possible to eliminate drone arcs in the same way, since the drone is allowed to start earlier than $e_{i^{\text{dr}}}$ at its release position i^{dr}. Second, if the demand of all customers along an arc $[i, j, k] \in \mathcal{A}$ is greater than the truck's capacity, the arc can be eliminated due to capacity restrictions, i.e.,

$$\mathcal{A}^{tog} \subset \{[i, j, \perp] \in \mathcal{W} \times \mathcal{W} \times \{\perp\} : q_{i^{\text{tr}}} + q_{j^{\text{tr}}} \leq Q\},$$

$$\mathcal{A}^{alone} \subset \{[i, j, \perp] \in \mathcal{W} \times \mathcal{W} \times \{\perp\} : q_{i^{\text{tr}}} + q_{j^{\text{tr}}} \leq Q\}, \text{ and}$$

$$\mathcal{A}^{drone} \subset \left\{[i, j, k] \in \mathcal{W} \times \mathcal{W} \times N^{dr} : q_{i^{\text{tr}}} + q_{j^{\text{tr}}} + q_k \leq Q\right\}.$$

In addition, all drone arcs that do not respect the drone's flying range δ can also be eliminated, i.e.,

$$\mathcal{A}^{drone} \subset \left\{[i, j, k] \in \mathcal{W} \times \mathcal{W} \times N^{dr} : t^{\text{dr}}_{i^{\text{dr}},k} + t^{\text{dr}}_{k,j^{\text{dr}}} \leq \delta\right\}.$$

Labeling Algorithm

In our labeling algorithm for the VRP-DTW, each label $\mathcal{L}_i$ represents a partial path $\mathcal{P}_i = ((0, 0), \ldots, i = (i^{tr}, i^{dr}))$ in $\mathcal{N}$ that starts at the depot $(0, 0)$ and ends at some artificial vertex $i = (i^{tr}, i^{dr}) \in \mathcal{N}$. Each element stored in a label is called an *attribute* R_i and represents the resource consumption along the partial path. Starting from an initial label $\mathcal{L}_0$ at the depot $0 = (0, 0)$, a labeling algorithm propagates labels toward the depot $(0', 0')$ over all three types of arcs $\mathcal{A} = \mathcal{A}^{tog} \cup \mathcal{A}^{alone} \cup \mathcal{A}^{drone}$ with the help of *resource extension functions* (REFs, Irnich, 2008). To avoid enumerating all feasible paths, some labels can be eliminated by a dominance criterion.

A label $\mathcal{L}_i$ at an artificial vertex $i = (i^{\text{tr}}, i^{\text{dr}})$ comprises the following attributes:

R_i^{rdc}: the negative accumulated *dual prices* along $\mathcal{P}$;
R_i^{load}: the accumulated *load* along $\mathcal{P}$;
$R_i^{tme,ar}$: the earliest *arrival time* at vertex i;
$R_i^{tme,dp}$: the earliest *departure time* at vertex i, i.e., including possible waiting times and service;

$R_i^{dur,ar}$: the minimum *route duration* along $\mathcal{P}$ up to the arrival on vertex i such that each customer in $\mathcal{P}$ is served within its time window;
$R_i^{dur,dp}$: the minimum *route duration* along $\mathcal{P}$, when leaving vertex i, i.e., including possible waiting times and service;
R_i^{strt}: the negative of the *latest possible departure time* at $(0,0)$ such that each customer in $\mathcal{P}$ can be served within its time window; and
$R_i^{cust,n}$: the number of times that customer $n \in N$ is served along the path.

The attributes R_i^{rdc}, R_i^{load} and $R_i^{cust,n}$ are standard resources in a *vehicle routing problem with time windows* (VRPTW, Desaulniers *et al.*, 2014). The attributes $R_i^{tme,ar}$ and $R_i^{tme,dp}$ are used to generate the as-early-as-possible schedule, while the resources $R_i^{dur,ar}, R_i^{dur,dp},$ and R_i^{strt} are used to generate the as-late-as-possible schedule (Tilk and Irnich (2017); Irnich (2008)). In contrast to the VRPTW, we explicitly distinguish between the arrival and the departure at each vertex i to model the situation where truck and drone have different departure times at a vertex i, e.g., whenever the truck arrives at customer i^{tr}, its earliest departure time is at $e_{i^{\text{tr}}} + \sigma_{i^{\text{tr}}}$, while the drone can leave as soon as the truck arrives (even before the time window opens). We do not distinguish between arrival and departure for the resource R_i^{strt}, since the latest possible departure time at the depot is independent from possible waiting and service times at the current vertex i. Note that the resource R_i^{strt} is defined to be negative to get a non-decreasing REF (Tilk and Irnich, 2017). To properly synchronize truck and drone at a later vertex $j = (j^{\text{tr}}, j^{\text{dr}}) \in \mathcal{W}$ with $j^{\text{tr}} = j^{\text{dr}}$ it is important to remember the resource consumption whenever truck and drone separate. Therefore, we use an additional set of attributes $\mathscr{R}_i$ whenever the truck operates alone $\mathscr{R}_i = (\mathscr{R}_i^{tme,ar}, \mathscr{R}_i^{tme,dp}, \mathscr{R}_i^{dur,ar}, \mathscr{R}_i^{dur,dp}, \mathscr{R}_i^{strt})$. The meaning of all attributes in $\mathscr{R}_i$ is the same as for the attributes in R_i, e.g., $\mathscr{R}_i^{tme,ar}$ is the earliest arrival time at vertex i when the *truck travels alone*. The initial label at the depot $0 = (0,0)$ is given by

$$\begin{aligned}
\mathcal{L}_0 &= (R_0^{rdc}, R_0^{load}, R_0^{tme,ar}, R_0^{tme,dp}, R_0^{dur,ar}, R_0^{dur,dp}, R_0^{strt}, (R_0^{cust,n})_{n\in N}, \mathscr{R}_0) \\
&= (0, 0, e_0, e_0, 0, 0, -l_0, \mathbf{0}, \mathscr{R}_0) \text{ with} \\
\mathscr{R}_0 &= (\mathscr{R}_0^{tme,ar}, \mathscr{R}_0^{tme,dp}, \mathscr{R}_0^{dur,ar}, \mathscr{R}_0^{dur,dp}, \mathscr{R}_0^{strt}) \\
&= (e_0, e_0, 0, 0, -l_0)
\end{aligned}$$

Extending an arbitrary label $\mathcal{L}_i$ over an arc $a = [i, j, k] \in \mathcal{A}$ creates a new label $\mathcal{L}_j$ for the corresponding partial path $\mathcal{P}_j = ((0,0), \ldots, i, j = (j^{\text{tr}}, j^{\text{dr}}))$ depending on the particular arc types $\mathcal{A}^{tog}, \mathcal{A}^{alone}$, and $\mathcal{A}^{drone}$. Thus, we distinguish three different types of REFs:

(i) propagating a label $\mathcal{L}_i$ over a *together arc* $a = [i, j, \perp] \in \mathcal{A}^{tog}$ results in label

$\mathcal{L}_j$ with

$$
\begin{aligned}
R_j^{rdc} &= R_i^{rdc} - \pi_{j^{\mathrm{tr}}}, && (5.2a)\\
R_j^{load} &= R_i^{load} + q_{j^{\mathrm{tr}}}, && (5.2b)\\
R_j^{tme,ar} &= R_i^{tme,dp} + t_{i^{\mathrm{tr}}j^{\mathrm{tr}}}, && (5.2c)\\
R_j^{tme,dp} &= \max\{R_j^{tme,ar}, e_{j^{\mathrm{tr}}}\} + \sigma_{j^{\mathrm{tr}}}, && (5.2d)\\
R_j^{dur,ar} &= R_i^{dur,dp} + t_{i^{\mathrm{tr}}j^{\mathrm{tr}}}, && (5.2e)\\
R_j^{dur,dp} &= \max\{R_j^{dur,ar}, R_i^{strt} + e_{j^{\mathrm{tr}}}\} + \sigma_{j^{\mathrm{tr}}}, && (5.2f)\\
R_j^{strt} &= \max\{R_j^{dur,ar} - l_{j^{\mathrm{tr}}}, R_i^{strt}\}, && (5.2g)\\
R_j^{cust,n} &= \begin{cases} R_i^{cust,n} + 1, & \text{if } n = j^{\mathrm{tr}} \\ R_i^{cust,n}, & \text{otherwise.} \end{cases} && (5.2h)
\end{aligned}
$$

Additionally, all attributes in $\mathscr{R}_j$ are set to zero: $\mathscr{R}_j^{tme,ar} = \mathscr{R}_j^{tme,dp} = \mathscr{R}_j^{dur,ar} = \mathscr{R}_j^{dur,dp} = \mathscr{R}_j^{strt} = \#$;

(ii) propagating a label $\mathcal{L}_i$ over a *truck alone arc* $a = [i, j, \perp] \in \mathcal{A}^{alone}$ results in label $\mathcal{L}_j$ with

$$
\begin{aligned}
R_j^{rdc} &= R_i^{rdc} - \pi_{j^{\mathrm{tr}}}, && (5.3a)\\
R_j^{load} &= R_i^{load} + q_{j^{\mathrm{tr}}}, && (5.3b)\\
\mathscr{R}_j^{tme,ar} &= \begin{cases} R_i^{tme,dp} + t_{i^{\mathrm{tr}}j^{\mathrm{tr}}}, & \text{if } \mathscr{R}_i^{tme,dp} = \# \\ \mathscr{R}_i^{tme,dp} + t_{i^{\mathrm{tr}}j^{\mathrm{tr}}}, & \text{if } \mathscr{R}_i^{tme,dp} \neq \# \end{cases} && (5.3c)\\
\mathscr{R}_j^{tme,dp} &= \max\{\mathscr{R}_j^{tme,ar}, e_{j^{\mathrm{tr}}}\} + \sigma_{j^{\mathrm{tr}}}, && (5.3d)\\
\mathscr{R}_j^{dur,ar} &= \begin{cases} R_i^{dur,dp} + t_{i^{\mathrm{tr}}j^{\mathrm{tr}}}, & \text{if } \mathscr{R}_i^{tme,dp} = \# \\ \mathscr{R}_i^{dur,dp} + t_{i^{\mathrm{tr}}j^{\mathrm{tr}}}, & \text{if } \mathscr{R}_i^{tme,dp} \neq \# \end{cases} && (5.3e)\\
\mathscr{R}_j^{dur,dp} &= \begin{cases} \max\{\mathscr{R}_j^{dur,ar}, R_i^{strt} + e_{j^{\mathrm{tr}}}\} + \sigma_{j^{\mathrm{tr}}}, & \text{if } \mathscr{R}_i^{tme,dp} = \# \\ \max\{\mathscr{R}_j^{dur,ar}, \mathscr{R}_i^{strt} + e_{j^{\mathrm{tr}}}\} + \sigma_{j^{\mathrm{tr}}}, & \text{if } \mathscr{R}_i^{tme,dp} \neq \# \end{cases} && (5.3f)\\
\mathscr{R}_j^{strt} &= \begin{cases} \max\{\mathscr{R}_j^{dur,ar} - l_{j^{\mathrm{tr}}}, R_i^{strt}\}, & \text{if } \mathscr{R}_i^{tme,dp} = \# \\ \max\{\mathscr{R}_j^{dur,ar} - l_{j^{\mathrm{tr}}}, \mathscr{R}_i^{strt}\}, & \text{if } \mathscr{R}_i^{tme,dp} \neq \# \end{cases} && (5.3g)\\
R_j^{cust,n} &= \begin{cases} R_i^{cust,n} + 1, & \text{if } n = j^{\mathrm{tr}} \\ R_i^{cust,n}, & \text{otherwise.} \end{cases} && (5.3h)
\end{aligned}
$$

Whenever truck and drone separate, the attributes in $\mathscr{R}_j$ are used and the attributes in R_j remain unchanged, i.e., $R_j^{tme,ar} = R_i^{tme,ar}$, $R_j^{tme,dp} = R_i^{tme,dp}$,

$R_j^{dur,ar} = R_i^{dur,ar}$, $R_j^{dur,dp} = R_i^{dur,dp}$, and $R_j^{strt} = R_i^{strt}$. Thus, the resource consumption up to the drone's release position is stored and whenever the drone returns to the truck, we resume this status; and

(iii) propagating a label $\mathcal{L}_i$ over a *drone arc* $a = [i, j, k] \in \mathcal{A}^{drone}$ results in label $\mathcal{L}_j$ with

$$
\begin{aligned}
R_j^{rdc} &= R_i^{rdc} - \pi_k, && (5.4a)\\
R_j^{load} &= R_i^{load} + q_k, && (5.4b)\\
R_j^{tme,ar} &= \max\left\{\max\{R_i^{tme,ar} + t_{i^{\mathrm{dr}}k}, l_k\} + \sigma_k + t_{kj^{\mathrm{dr}}}, \mathscr{R}_j^{tme,ar}\right\}, && (5.4c)\\
R_j^{tme,dp} &= \max\left\{\max\{R_i^{tme,ar} + t_{i^{\mathrm{dr}}k}, l_k\} + \sigma_k + t_{kj^{\mathrm{dr}}}, \mathscr{R}_j^{tme,dp}\right\}, && (5.4d)\\
R_j^{strt} &= \max\left\{\max\{R_i^{dur,ar} + t_{i^{\mathrm{dr}}k} - l_k, R_i^{strt}\}, \mathscr{R}_i^{strt}\right\}, && (5.4e)\\
R_j^{cust,n} &= \begin{cases} R_i^{cust,n} + 1, & \text{if } n = k \\ R_i^{cust,n}, & \text{otherwise.} \end{cases} && (5.4f)
\end{aligned}
$$

While propagating over a drone arc, the drone returns to the truck and all resources in $\mathscr{R}_j$ are set to # again. The REFs for $R_j^{dur,ar}$ and $R_j^{dur,dp}$ are defined in paragraph *Synchronizing the duration of trucks and drones.*

We remark that the resources $R^{tme,dp}$ and $\mathscr{R}^{tme,dp}$ are not necessary in the labeling algorithm. They do not affect the feasibility of a path (feasibility is ensured by $R^{tme,ar}$ and $\mathscr{R}^{tme,ar}$, respectively) and can always be calculated from the resources $R^{tme,ar}$ and $\mathscr{R}^{tme,ar}$. They are only considered for the sake of a simpler notation.

A new label $\mathcal{L}_j$ is feasible if the capacity (5.5a), time-window (5.5b)–(5.5d), and elementary (5.5e) constraints are fulfilled, i.e.,

$$
\begin{aligned}
R_j^{load} &\le Q && \text{if } a \in \mathcal{A}^{alone} \cup \mathcal{A}^{tog} \cup \mathcal{A}^{drone} && (5.5a)\\
R_j^{tme,ar} &\le l_{j^{\mathrm{tr}}}, && \text{if } a \in \mathcal{A}^{tog} && (5.5b)\\
\mathscr{R}_j^{tme,ar} &\le l_{j^{\mathrm{tr}}}, && \text{if } a \in \mathcal{A}^{alone} && (5.5c)\\
R_i^{tme,ar} + t_{i^{\mathrm{dr}}k} &\le l_k, && \text{if } a \in \mathcal{A}^{drone} && (5.5d)\\
R_j^{cust,n} &\le 1, && \text{for all } n \in N. && (5.5e)
\end{aligned}
$$

Synchronizing the duration of trucks and drones

Synchronizing the duration of truck and drone after propagating a label $\mathcal{L}_i$ over a drone arc $\mathcal{A}^{drone}$ according to an intuitive REF with a max-term, i.e.,

$$R_j^{dur,ar} = \max\left\{\max\{R_i^{dur,ar} + t_{i^{\mathrm{dr}}k}, R_i^{strt} + e_k\} + \sigma_k + t_{k,j^{\mathrm{dr}}}, \mathscr{R}_j^{dur,ar}\right\}, \quad (5.6a)$$

$$R_j^{dur,dp} = \max\left\{\max\{R_i^{dur,ar} + t_{i^{\mathrm{dr}}k}, R_i^{strt} + e_k\} + \sigma_k + t_{kj^{\mathrm{dr}}}, \mathscr{R}_i^{dur,dp}\right\} \quad (5.6b)$$

does not necessarily result in a partial path with minimum duration. Especially not when truck and drone leave the depot 0 at different times, i.e., $\max\{R_i^{dur,ar} + t_{i^{\mathrm{dr}}k} - l_k, R_i^{strt}\} \neq \mathscr{R}_i^{strt}$ in REF (5.4e). If either the truck or the drone has to leave earlier due to (5.4e), the REFs (5.6a)–(5.6b) do not consider this additional duration for the corresponding vehicle. However, adding the difference of $\max\{R_i^{dur,ar} + t_{i^{\mathrm{dr}}k} - l_k, R_i^{strt}\}$ and $\mathscr{R}_i^{strt}$ to the corresponding duration in (5.6a)–(5.6b) does also not always lead to a proper duration. Since both vehicles are scheduled according to an as-late-as-possible schedule, shifting the departure to an earlier point in time may result in unnecessary waiting times at one or several customers along the path if the service still starts at the latest feasible point in time. Example 11 visualizes both situations.

Example 11 *Given a partial route* $r = (P = (0, 1, 3, \ldots), D = (\langle 0, 2, 3\rangle), \ldots))$ *with time windows* $[e_0, l_0] = [0, 25], [e_1, l_1] = [10, 13], [e_2, l_2] = [6, 8]$, *and* $[e_3, l_3] = [15, 20]$. *The service times are assumed to be* $\sigma_1 = \sigma_2 = 1$ *and* $\sigma_3 = 3$. *All travel times for truck and drone are set to 2.*

Figure 5.1 depicts two timelines that represent the truck-and-drone movements in route r according to an as-late-as-possible schedule. The bar above each timeline, represents the truck path P and the bar below depicts the drone subpath in D. For the sake of clarity, both paths distinguish between travel times (colored in dark blue), service times (colored in light blue), and waiting times (colored in red). The customer time windows are sketched by curly brackets.

The first timeline shows that using REFs (5.6a)–(5.6b) *would lead to an incorrect duration of the path* $R_3^{dur,dp} = 8$ *that corresponds only to the trucks' duration. It ignores the additional duration caused by leaving the depot at time 6 (shown as a red dashed arc). The second timeline shows that only shifting the truck, i.e., letting the truck start earlier by the difference of* $\max\{R_i^{dur,ar} + t_{i^{dr}k} - l_k, R_i^{strt}\}$ *and* $\mathscr{R}_i^{strt}$, *leads to unnecessary waiting at customer 1. Although the truck arrives before the time window opens, the service is performed at the latest possible time, resulting in a path duration of* $R_3^{dur,dp} = 13$.

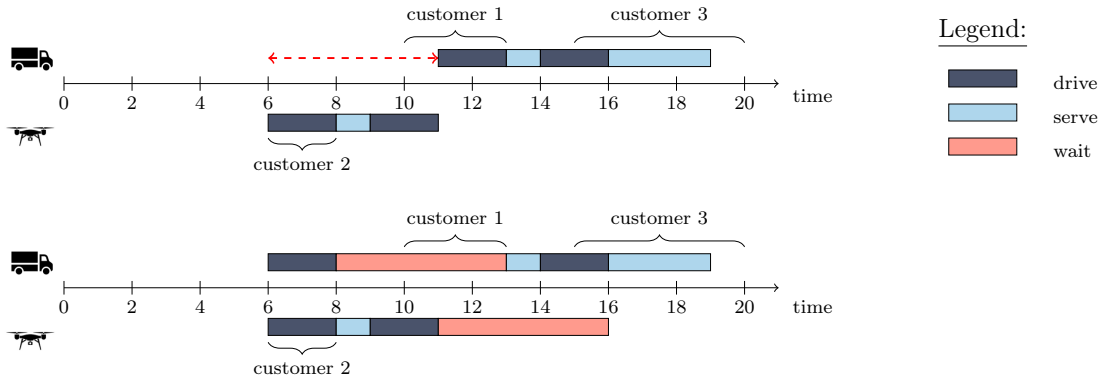


Figure 5.1: Incorrect synchronization of truck and drone resulting in an incorrect duration.

On each vertex j along a path, it is possible to shift the latest possible departure time (for truck or drone) in the direction to e_0 for a specific amount of time without increasing the duration of the path. More precisely, it is possible to shift the departure of a vehicle by the difference between the as-late-as-possible schedule and the as-early-as-possible schedule. Each additional shift in time results in additional waiting times that increases the path's duration. To determine the additional duration, we define the variable $\alpha^{\mathrm{dr}} = \max\{R_i^{dur,ar} + t_{i^{\mathrm{dr}}k} - l_k, R_i^{strt}\}$ as the latest possible departure time for the drone at node j just before synchronizing with the truck. Whenever the drone leaves earlier than the truck, i.e., $\mathscr{R}_i^{strt} < \alpha^{\mathrm{dr}}$, the additional duration is calculated as follows:

$$\Delta = \left(|\mathscr{R}_i^{strt} - \alpha^{\mathrm{dr}}| - (-\mathscr{R}_i^{strt} - \mathscr{R}_i^{tme,dp} + \mathscr{R}_i^{dur,dp})\right)^+,$$
$$\Delta^{ar} = \left(|\mathscr{R}_i^{strt} - \alpha^{\mathrm{dr}}| - (-\mathscr{R}_i^{strt} - \mathscr{R}_i^{tme,ar} + \mathscr{R}_i^{dur,ar})\right)^+,$$

where Δ (Δ^{ar}) represents the additional duration on departure (arrival). In case the truck leaves earlier than the drone, i.e., $\mathscr{R}_i^{strt} > \alpha^{\mathrm{dr}}$, the additional duration is determined by

$$\Delta = \left(|\mathscr{R}_i^{strt} - \alpha^{\mathrm{dr}}| - (-\alpha^{\mathrm{dr}} - \max\{R_i^{tme,ar} + t^{\mathrm{dr}}_{i^{\mathrm{tr}}k}, l_k\} + \max\{R_i^{dur,ar} + t^{\mathrm{dr}}_{i^{\mathrm{tr}}k}, R_i^{strt} + e_k\})\right)^+.$$

Note that in the second case only the duration on departure needs to be considered, since the duration on arrival at drone customer k is not relevant.

After determining the additional duration, we can express the REFs as follows:

$$R_j^{dur,ar} = \begin{cases} \max\left\{\max\{R_i^{dur,ar} + t_{i^{\mathrm{dr}}k}, R_i^{strt} + e_k\} + \sigma_k + t_{k,j^{\mathrm{dr}}} + \Delta, \mathscr{R}_j^{dur,ar}\right\}, & \text{if } \mathscr{R}_i^{strt} > \alpha^{\mathrm{dr}} \\ \max\left\{\max\{R_i^{dur,ar} + t_{i^{\mathrm{dr}}k}, R_i^{strt} + e_k\} + \sigma_k + t_{k,j^{\mathrm{dr}}}, \mathscr{R}_j^{dur,ar} + \Delta^{ar}\right\}, & \text{if } \mathscr{R}_i^{strt} < \alpha^{\mathrm{dr}} \end{cases} \tag{5.6a'}$$

$$R_j^{dur,dp} = \begin{cases} \max\left\{\max\{R_i^{dur,ar} + t_{i^{\mathrm{dr}}k}, R_i^{strt} + e_k\} + \sigma_k + t_{kj^{\mathrm{dr}}} + \Delta, \mathscr{R}_i^{dur,dp}\right\}, & \text{if } \mathscr{R}_i^{strt} > \alpha^{\mathrm{dr}} \\ \max\left\{\max\{R_i^{dur,ar} + t_{i^{\mathrm{dr}}k}, R_i^{strt} + e_k\} + \sigma_k + t_{kj^{\mathrm{dr}}}, \mathscr{R}_i^{dur,dp} + \Delta\right\}, & \text{if } \mathscr{R}_i^{strt} < \alpha^{\mathrm{dr}} \end{cases} \tag{5.6b'}$$

Example 12 *(continued from Example 11) The new timeline in Figure 5.2 depicts the situation when synchronizing truck and drone according to REFs (5.6a') and (5.6b').*

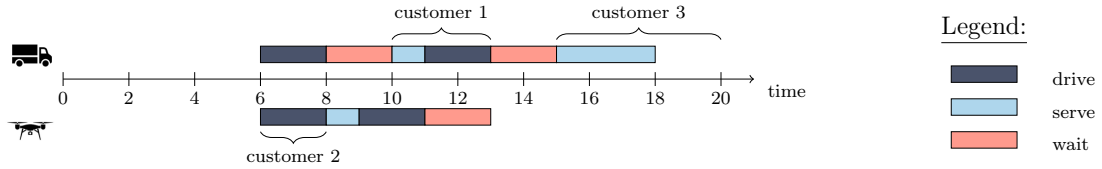


Figure 5.2: Synchronization of truck and drone resulting in a correct duration.

It shows that customer 1's service now starts immediately when the time window opens for service. This reduces the truck's waiting time by three time units (compared to the second timeline in Figure 5.1). In addition, the earlier arrival of the truck at customer 3 also reduces the time that the drone waits for the truck. The new duration of the partial route r when leaving vertex 3 is $R_3^{dur,dp} = 12$.

5.4.3 Dominance

A *dominance procedure* can be used to eliminate provable redundant labels during the labeling process. Following the ideas of Roberti and Ruthmair (2021), we present three dominance rules that depend on the truck-and-drone positions. Whenever truck and drone are at the same position, a label $\mathcal{L}_1$ dominates a label $\mathcal{L}_2$ at the same vertex $i = (i^{\mathrm{tr}}, i^{\mathrm{dr}}) \in \mathcal{N}$ with $i^{\mathrm{tr}} = i^{\mathrm{dr}}$ if the following dominance rule holds true:

Dominance Rule 3 *If $i^{tr} = i^{dr}$, $R_1^{rdc} \leq R_2^{rdc}$, $R_1^{load} \leq R_2^{load}$, $R_1^{tme,ar} \leq R_2^{tme,ar}$, $R_1^{dur,ar} \leq R_2^{dur,ar}$, $R_1^{dur,dp} \leq R_2^{dur,dp}$, $R_1^{strt} \leq R_2^{strt}$, and $R_1^{cust,n} \leq R_2^{cust,n}$ for all $n \in N$, then $\mathcal{L}_1$ dominates $\mathcal{L}_2$.*

then Dominance Rule 3 follows the *standard dominance rules*, i.e., a simple pairwise $\leq$-comparison, since all REFs considered are non-decreasing and bounded from above (Irnich, 2008). Since truck and drone are at the same position, all attributes in $\mathscr{R}_1$ and $\mathscr{R}_2$ are zero according to REFs (5.2) and (5.4). Consequently, the standard dominance rules are always fulfilled for this set of attributes.

Whenever truck and drone are separated, i.e., $i^{\mathrm{tr}} \neq i^{\mathrm{dr}}$, stronger dominance criteria than a pairwise $\leq$-comparison can be applied for all attributes that have an equivalent attribute in $\mathscr{R}_i$. A label $\mathcal{L}_1$ dominates a label $\mathcal{L}_2$ at the same vertex $i = (i^{\mathrm{tr}}, i^{\mathrm{dr}}) \in \mathcal{N}$ with $i^{\mathrm{tr}} \neq i^{\mathrm{dr}}$ if one of the following dominance rules hold true:

Dominance Rule 4 *If* $i^{tr} \neq i^{dr}, R_1^{rdc} \leq R_2^{rdc}, R_1^{load} \leq R_2^{load}, R_1^{tme,ar} \leq R_2^{tme,ar}$, $R_1^{tme,ar} + \mathscr{R}_1^{tme,ar} \leq R_2^{tme,ar} + \mathscr{R}_2^{tme,ar}, R_1^{dur,ar} \leq R_2^{dur,ar}, R_1^{dur,ar} + \mathscr{R}_1^{dur,ar} \leq R_2^{dur,ar} + \mathscr{R}_2^{dur,ar}, R_1^{dur,dp} \leq R_2^{dur,dp}, R_1^{dur,dp} + \mathscr{R}_1^{dur,dp} \leq R_2^{dur,dp} + \mathscr{R}_2^{dur,dp}, R_1^{strt} \leq R_2^{strt}, R_1^{strt} + \mathscr{R}_1^{strt} \leq R_2^{strt} + \mathscr{R}_2^{strt}$ *and* $R_1^{cust,n} \leq R_2^{cust,n}$ *for all* $n \in N$, *then* $\mathcal{L}_1$ *dominates* $\mathcal{L}_2$.

Dominance Rule 5 *If* $i^{tr} \neq i^{dr}, R_1^{rdc} \leq R_2^{rdc}, R_1^{load} \leq R_2^{load}, R_1^{cust,n} \leq R_2^{cust,n}$ *for all* $n \in N$,

$$
\begin{aligned}
R_1^{tme,ar} &\geq R_2^{tme,ar},\\
R_1^{tme,ar} + \mathscr{R}_1^{tme,ar} &\leq R_2^{tme,ar} + \mathscr{R}_2^{tme,ar},\\
R_1^{tme,ar} + \max_{k \in N^{dr}, w \in N \cup \{0'\}: k \neq w} \{l_k + \sigma_k + t_{kw}^{dr}\} &\leq R_2^{tme,ar} + \mathscr{R}_2^{tme,ar},\\
R_1^{dur,ar} &\geq R_2^{dur,ar},\\
R_1^{dur,ar} + \mathscr{R}_1^{dur,ar} &\leq R_2^{dur,ar} + \mathscr{R}_2^{dur,ar},\\
R_1^{dur,ar} + \max\{\mathscr{R}_1^{dur,ar} + t_{i^{dr}k^*}^{dr}, \mathscr{R}_1^{strt} + e_{k^*}\} + \sigma_{k^*} + t_{k^*w^*}^{dr} &\leq R_2^{dur,ar} + \mathscr{R}_2^{dur,ar},\\
R_1^{dur,dp} &\geq R_2^{dur,dp},\\
R_1^{dur,dp} + \mathscr{R}_1^{dur,dp} &\leq R_2^{dur,dp} + \mathscr{R}_2^{dur,dp}\\
R_1^{dur,dp} + \max\{\mathscr{R}_1^{dur,dp} + t_{i^{dr}k^*}^{dr}, \mathscr{R}_1^{strt} + e_{k^*}\} + \sigma_{k^*} + t_{k^*w^*}^{dr} &\leq R_2^{dur,dp} + \mathscr{R}_2^{dur,ar},\\
R_1^{strt} &\geq R_2^{strt},\\
R_1^{strt} + \mathscr{R}_1^{strt} &\leq R_2^{strt} + \mathscr{R}_2^{strt}, \text{ and}\\
R_1^{strt} + \max\{R_1^{dur,ar} + t_{k^*w^*}^{dr} - l_{k^*}, R_1^{strt}\} &\leq R_2^{strt} + \mathscr{R}_2^{strt}
\end{aligned}
$$

with $(k^*, w^*) = \arg\max_{k \in N^{dr}, w \in N \cup \{0'\}: k \neq w} \{l_k + \sigma_k + t_{kw}^{dr}\}$, *then* $\mathcal{L}_1$ *dominates* $\mathcal{L}_2$.

For example, in Dominance Rule 4, the inequality $R_1^{tme,ar} + \mathscr{R}_1^{tme,ar} \leq R_2^{tme,ar} + \mathscr{R}_2^{tme,ar}$ is only valid if $R_1^{tme,ar} \leq R_2^{tme,ar}$ and $\mathscr{R}_1^{tme,ar} \leq \mathscr{R}_2^{tme,ar}$ are fulfilled at the same time. Dominance Rule 5 works in a similar way: When $R_1^{tme,ar} \geq R_2^{tme,ar}$ holds true, the inequality $R_1^{tme,ar} + \mathscr{R}_1^{tme,ar} \leq R_2^{tme,ar} + \mathscr{R}_2^{tme,ar}$ can only be true if $\mathscr{R}_1^{tme,ar} \leq \mathscr{R}_2^{tme,ar}$. Additionally, the maximum term guarantees that the resource consumption of the first path is never larger than that of the second path, when truck and drone meet each other at a later position $j = (j^{\text{tr}}, j^{\text{dr}})$ with $j^{\text{tr}} = j^{\text{dr}}$ in the path.

5.4.4 Acceleration Techniques

In this section, we sketch techniques to accelerate the proposed BPC algorithm.

Lower Bound on the Number of Trucks We compute a lower bound on the number of trucks needed to serve all customers based on their demand. Therefore, we solve a bin-packing problem with a bin size equal to the capacity of the truck Q and

item weights equal to the demand q_v of each customer $v \in N$. This bin-packing problem is modeled as an arc-flow formulation (Valério de Carvalho, 1999). Using a lower bound on the number of trucks can prevent the branch-and-bound tree from solving infeasible nodes, while branching on the number vehicles used.

ng-Route Relaxation and Dynamic ng-Neighborhood Extension Instead of solving an *elementary* SPPRC, it is common practice to relax its elementary requirement and only solve a SPPRC. On the one hand, this leads to an easier/faster solution of the pricing problem, but on the other hand, it comes at the cost of a weaker linear relaxation of the corresponding MP. A prominent method to control this trade-off is the ng-route relaxation, originally invented by Baldacci *et al.* (2011). It introduces a parametrized neighborhood $N_v \subset N$ for all $v \in N$ to effectively avoid non-elementary paths. A non-elementary cycle $(v, v_1, \ldots, v_m, v)$ over vertex v with $m \geq 1$ is only feasible if $v \notin N_{v_l}$ for some $l \in \{1, \ldots, m\}$. The quality of the lower bound and the efficiency of the pricing problem now depends on the size of N_v, i.e., larger neighborhoods lead to tighter bounds, while smaller neighborhoods lead to a faster labeling algorithm. To adapt the ng-route relaxation to the artificial network $\mathcal{N}$ we assign with each artificial vertex $(i^{\text{tr}}, i^{\text{dr}}) \in \mathcal{N}$ the neighborhood of the truck vertex $N_{i^{\text{tr}}}$. In addition, we strength the ng-route relaxation with *dynamic ng-neighborhood extensions* (DNE, Roberti and Mingozzi, 2014; Bode and Irnich, 2015) in the same fashion as proposed in Chapter 4. The REFs (5.2h), (5.3h), and (5.4f) are modified by $R_j^{cust,n} = 0$ if $n \notin N_j$.

Heuristic Pricing The pricing subproblem does not necessarily have to be solved to optimality, as long as negative reduced-cost columns can be generated. Therefore, we use a hierarchy of pricing heuristics (Gamache *et al.*, 1999) that rely on reduced networks to accelerate the labeling procedure. The first (second) heuristic only includes 6 (10) outgoing truck arcs ($\mathcal{A}^{alone} \cup \mathcal{A}^{tog}$) and at most 3 (5) drone arcs ($\mathcal{A}^{drone}$). Only if none of these heuristics has generated a negative reduced-cost column, the pricing problem is solved exactly. We choose the arcs according to the lowest reduced cost in each pricing iteration.

MIP-based heuristic To provide an early upper bound, we solve use MIP-based heuristic at the first and second level of the branch-and-bound tree. Therefore, we solve the RMP with all columns generated up to this point as an integer model. Whenever the BPC terminates without finding an upper bound within the given time limit, the final RMP is solved with a MIP-based heuristic again.

Unsuccessful Techniques Developing a bidirectional labeling algorithm on the asymmetric network is a non-trivial task. Nevertheless, the work of Blufstein *et al.* (2024) and our work in Chapter 4 have shown that a bidirectional labeling algorithm on this network is possible for TSP-D and VRP-D variants. Pre-tests have shown, that it is also possible to develop a bidirectional labeling for the VRP-DTW, but the number of

generated labels in the backward labeling increases rapidly compared to the forward labeling. To prevent the backward labeling from this *combinatorial explosion*, even stronger dominance rules or other techniques are necessary. Despite the general success of bidirectional labeling (Righini and Salani, 2006; Tilk *et al.*, 2017), we do not employ it here because it unfortunately does not pay off.

5.4.5 Valid Inequalities

In our BPC algorithm, we implement *capacity cuts* (CCs, Baldacci *et al.*, 2008) to strengthen the linear relaxation of the RMP. Let $C \subseteq N$ be any subset of customers and $K(C)$ a lower bound on the number of trucks needed to serve all customers in C according to their demand. The valid inequality reads

$$\sum_{r \in \Omega_C} \lambda_r \geq K(C), \tag{5.8}$$

where $\Omega_C = \{r \in \Omega : N(r) \cap C \neq \varnothing\}$ represents the subset of routes that contain at least one customer from subset C. Instead of solving a bin packing problem (as done in Section 5.4.4), we compute the lower bound $K(C) = \lceil \sum_{v \in C} q_v / Q \rceil$.

Since this variant of CCs is non-robust, they affect the structure of the pricing problem, i.e., we have to adjust the pricing problem to handle the dual prices of each inequality. The adjustments are as follows: Let $C \in \mathcal{C}$ denote all active CCs and $\gamma_C > 0$ their corresponding positive dual prices. Further, we define an additional binary resource $(R^{cc,C})_{C \in \mathcal{C}}$, initially set to zero. When propagating along an arc $a = [i, j, k] \in \mathcal{A}$, we update the resource according to the arc type used:

$$R_j^{cc,C} = \begin{cases} 1, & \text{if } a = [i, j, \perp] \in \mathcal{A}^{tog} \cup \mathcal{A}^{alone} \text{ and } j^{\text{tr}} \in C \\ 1, & \text{if } a = [i, j, k] \in \mathcal{A}^{drone} \text{ and } k \in C \\ R_i^{cc,C}, & \text{otherwise.} \end{cases} \tag{5.9}$$

The dual price γ_C has to be subtracted from the reduced cost, whenever a partial path serves a customer in C for the first time. Therefore, we adjust the REFs in (5.2a), (5.3a), and (5.4a) as follows:

$$R_j^{rdc} = \begin{cases} R_i^{rdc} - \pi_{j^{\text{tr}}} - \sum\limits_{C \in \mathcal{C}: R_i^{cc,C} = 0, j^{\text{tr}} \in C} \gamma_C & \text{if } a = [i, j, \perp] \in \mathcal{A}^{tog} \quad (5.2\text{a'}) \\ R_i^{rdc} - \pi_{j^{\text{tr}}} - \sum\limits_{C \in \mathcal{C}: R_i^{cc,C} = 0, j^{\text{tr}} \in C} \gamma_C & \text{if } a = [i, j, \perp] \in \mathcal{A}^{alone} \quad (5.3\text{a'}) \\ R_i^{rdc} - \pi_k - \sum\limits_{C \in \mathcal{C}: R_i^{cc,C} = 0, k \in C} \gamma_C & \text{if } a = [i, j, k] \in \mathcal{A}^{drone} \quad (5.4\text{a'}) \end{cases}$$

The dominance Rules 3 to 5 are modified according to Baldacci *et al.* (2008). To identify violated inequalities (5.8), we use the MIP-based separation procedure by Martinelli *et al.* (2013).

5.4.6 Branching

To obtain an integer solution of formulation (5.1), we use a four-stage branching scheme. Let λ_r^* be a fractional solution of the RMP defined over $\bar{\Omega}$. At the first stage, we branch on the number of trucks in use, whenever $K^* = \sum_{r\in\Omega} \lambda_r^*$ is fractional. Therefore, we set the bounds in (5.1c) to $L = \lceil K^* \rceil$ and $K = \lfloor K^* \rfloor$, respectively.
Second, we branch on the number of times each customer is served by a drone (Roberti and Ruthmair, 2021). Let y_{kr} be the number of times that customer k is served by drone in route r. Whenever $y_k^* = \sum_{r\in\Omega} y_{kr}\lambda_r^*$ is fractional for a customer $k \in N^{dr}$, we create two branches $y_{k^*} = 0$ (i.e., customer k^* has to be served by the truck) and $y_{k^*} = 1$ (i.e., customer k^* has to be served by a drone) for customer k^*. We choose k^* according to a value $y_{k^*}^* - \lfloor y_{k^*}^* \rfloor$ closest to 0.5.
At the third stage, we consider the truck flow between two customers $v, w \in N$ with $v \neq w$. Let f_{ar}^{tr} be the flow value on a route r along a truck arc $a \in \mathcal{A}_{vw}^{tr} = \{[i,j,\perp] \in \mathcal{A}^{tog} : i^{\text{tr}} = i^{\text{dr}} = v, j^{\text{tr}} = j^{\text{dr}} = w\} \cup \{[i,j,\perp] \in \mathcal{A}^{alone} : i^{\text{tr}} = v, j^{\text{tr}} = w, i^{\text{dr}} = j^{\text{dr}} \in V\}$. Whenever, $f_{ar}^{tr*} = \sum_{r\in\Omega}\sum_{a\in\mathcal{A}_{vw}^{tr}} f_{ar}^{tr}\lambda_r^*$ is fractional for a customer pair v and w, we choose v^*, w^* such that the value $f_{v^*w^*}^{tr*} - \lfloor f_{v^*w^*}^{tr*} \rfloor$ is closest to 0.5. We create the two branches $\sum_{r\in\Omega}\sum_{a\in\mathcal{A}_{v^*w^*}^{tr}} f_{ar}^{tr}\lambda_r = 0$, i.e., arc $(v^*, w^*) \in A$ must not be traversed by a truck and $\sum_{r\in\Omega}\sum_{a\in\mathcal{A}_{v^*w^*}^{tr}} f_{ar}^{tr}\lambda_r = 1$, i.e., arc $(v^*, w^*) \in A$ must be traversed by a truck.

Finally, we branch on the flow of a drone subpath $\langle v, k, w\rangle$, similar to the branching strategy on the third stage. Therefore, let f_{ar}^{dr} be the flow value on a route r along a drone subpath $a \in \mathcal{A}_{\langle v,k,w\rangle}^{dr} = \{[(w,v),(w,w),k] \in \mathcal{A}^{drone}\}$. Again, whenever $f_{ar}^{\text{dr}*} = \sum_{r\in\Omega}\sum_{a\in\mathcal{A}_{\langle v,k,w\rangle}^{\text{dr}}} f_{ar}^{\text{dr}}\lambda_r^*$ is fractional for a drone subpath $\langle v, k, w\rangle$ with $v \neq k \neq w, v \neq w$ we choose the subpath $\langle v^*, k^*, w^*\rangle$ according to a value $f_{\langle v^*,k^*,w^*\rangle}^{\text{dr}} - \lfloor f_{\langle v^*,k^*,w^*\rangle}^{\text{dr}} \rfloor$ closest to 0.5. The two branches are created as follows: $\sum_{r\in\Omega}\sum_{a\in\mathcal{A}_{\langle v^*,k^*,w^*\rangle}^{dr}} f_{ar}^{\text{dr}}\lambda_r = 0$, i.e., the subpath $\langle v^*, k^*, w^*\rangle$ must not be traversed by a drone and $\sum_{r\in\Omega}\sum_{a\in\mathcal{A}_{\langle v^*,k^*,w^*\rangle}^{dr}} f_{ar}^{\text{dr}}\lambda_r = 1$, i.e., the subpath $\langle v^*, k^*, w^*\rangle$ must be traversed by a drone. Unique truck and drone paths are implied by the branching decisions on stages three and four. Since both types of decisions fully determine a solution, the presented branching scheme is complete.

We remark that all branching decisions on the last three stages can be implemented directly on the artificial network by removing a subset of arcs and/or vertices. Thus, it is not necessary to add the constraints to the RMP and handle their dual prices in the pricing problem. We use a *best-first* strategy to explore the branch-and-bound tree with the primal goal to improve the dual bound as fast as possible.

5.5 Computational Results

Our BPC algorithm was coded in `C++` and compiled into a 64-bit single-thread executable with GCC 11.2.0 in release mode. CPLEX 20.1.0 with default parameters (except for allowing only a single thread) is used to (re)optimize the RMPs, solve the primal MIP-based heuristic, solve the bin-packing problem, and separate CCs. The computation

time is limited to 7200 seconds per instance plus an additional time of 60 seconds to solve the MIP-based heuristic after time out. The computations were carried out on the high-performance computing cluster `MOGON II` of Johannes Gutenberg University Mainz. All used nodes are equipped with Intel Xeon E5-2630v4 (Broadwell) processors running at 2.2 GHz. Thus, its performance (on a single node) is slightly worse compared to a recent standard desktop processor.

5.5.1 Instances

Since there is no commonly used instance set publicly available, we generate a set of VRP-DTW instances to test our BPC algorithm. We create a set of 20 instances with $15, 20, 25$, and 30 customers, respectively, together with a single depot. Each customer location is placed at a randomly distributed location on a 50×50 grid. The depot is always centrally located at $(25, 25)$ and houses a homogeneous fleet of trucks, each with a capacity of $Q = 100$. The fleet size K is equal to the number of customers considered in the instance, so the fleet is assumed to be unconstrained. A drone can carry goods of at most $Q^{\mathrm{dr}} = 25$. Each customer $v \in N$ has a demand q_v randomly drawn from $q_v \in \{10, 15, 20, 25, 30\}$. The service time is set to 10 for all customers, i.e., $\sigma_v = 10$ for all $v \in N$. We assume that there is no service at the depot, i.e., $\sigma_0 = \sigma_{0'} = 0$. We assume a planning horizon of 540 units, so we set the time window at the depots accordingly: $[e_0, l_0] = [e_{0'}, l_{0'}] = [0, 540]$. For each customer, we randomly draw a time window from the following options: $[0, 240], [240, 480]$, or a randomly generated time window with a width of 180 within the interval of the planning horizon. As is common practice in the literature on truck-and-drone routing, we assume that drone travel times are lower than truck travel times. Given two customers v and w with their corresponding positions on the grid (x_v, y_v) and (x_w, y_w), we compute the travel time for trucks as Manhattan distance rounded down to one decimal, i.e., $t_{vw} = \lfloor 10\,(|x_v - x_w| + |y_v - y_w|)\rfloor$. The travel time for a drone is computed as Euclidean distance divided by a factor β representing the speed of the drone (also rounded down to one decimal), i.e, $t^{\mathrm{dr}}_{vw} = \lfloor 10\sqrt{(x_v - x_w)^2 + (y_v - y_w)^2}/\beta\rfloor$. We initially set $\beta = 3$, i.e., a drone is three times faster than a truck. We add a small offset ϵ to all travel times such that the triangle inequality holds.

5.5.2 Evaluation of Algorithmic Components

We now present algorithmic results for our BPC algorithm. At first, we show the results on a pure branch-and-price (BaP) algorithm, i.e., without considering CCs. In a second step, we successively add CCs and strength the *ng*-route relaxation with DNE, to show their contribution to the solution process. Table 5.2 shows the results for the BaP algorithm grouped by the number of customers considered. For the *ng*-route relaxation, we choose the neighborhood N_v that contains vertex v itself and the five closest customers that can be reached within their time windows. The columns of the table have the following meaning:

#Opt: the number of instances (out of 20) solved to proven optimality within the given time limit;
Gap$_r$: the average gap between the optimal solution (*opt*) and the lower bound at the root node (LB_r) in percent, i.e., $100 \cdot (opt - LB_r)/LB_r$;
Gap: the average remaining gap between the upper bound (UB) and lower bound (LB) after time out in percent, i.e., $100 \cdot (UB - LB)/LB$;
#B&B: the average number of solved branch-and-bound nodes; and
Time: the average solution time (in seconds).

Note that Gap$_r$ only considers instances for which an optimal solution *opt* is known. In contrast, Gap considers all 80 instances, since the algorithm computed an UB and LB for each instance.

$\|N\|$	#Opt	Gap$_r$	Gap	#B&B	Time
15	20	1.97	0.00	27.5	81.0
20	19	1.40	<0.01	101.0	1,869.6
25	12	1.16	0.36	164.6	4,810.7
30	1	0.32	1.14	61.3	6,954.9
Total	52	1.54	0.36	88.6	3,429.1

Table 5.2: Computational results for the branch-and-price algorithm.

The BaP algorithm can solve 51 out of 60 instances with up to 25 customers to proven optimality. For instances with 30 customers, only one instance can be solved to proven optimality. Since the average solution times and the number of solved branch-and-bound nodes are high, we implement CCs to strengthen the linear relaxation at the root node (see Section 5.4.5). Additionally, we reduce the size of all neighborhoods N_v to 3 customers (including the vertex v itself and its two closest neighbors) and extend the neighborhoods dynamically (see Section 5.4.4). The dynamic extension is stopped when all routes are elementary or 20 task cycles have been eliminated.

Table 5.3 presents the results of both BPC algorithms, including CCs and DNE, again grouped by the number of customers considered. The columns have the same meaning as in Table 5.2. The additional column Gap$_{r+c}$ represents the average percentage gap at the root node after adding CCs and at most 20 iterations of DNE. Analogous to Gap$_r$, it is only calculated for instances solved to proven optimality.

Table 5.3 shows that the addition of CCs reduces the number of solved branch-and-bound nodes by around 75% compared to the BaP algorithm. Also, the average gap at the root node (after adding cuts) can be reduced by approximately 50%. Unfortunately, CCs increase the computation times for small instances with 15 and 20 customers, but they reduce the computation times as soon as more than 20 customers are considered in an instance. Nevertheless, only one additional instance with 30 customers can be solved to proven optimality in comparison to the BaP algorithm. The addition of DNE can finally decrease the computation times by approximately 15% over all instances.

	BPC with CCs						BPC with CCs and DNE					
$\|N\|$	#Opt	Gap_r	Gap_{r+c}	Gap	#B&B	Time	#Opt	Gap_r	Gap_{r+c}	Gap	#B&B	Time
15	20	1.97	0.89	0.00	6.8	114.3	20	2.85	0.89	0.00	7.8	84.8
20	19	1.40	0.69	0.03	21.5	1,969.0	20	1.75	0.69	0.00	25.2	1,630.0
25	12	1.30	0.55	0.38	43.1	4,621.8	18	1.66	0.81	0.21	88.5	3,386.8
30	2	0.59	0.13	1.10	14.0	6,776.4	4	1.13	0.63	1.18	58.5	6,618.6
Total	53	1.56	0.71	0.36	21.3	3,370.4	62	2.04	0.79	0.34	45.0	2,930.0

Table 5.3: Computational results for the branch-price-and-cut algorithm with CCs and DNE.

Additionally, nine more instances can be solved so that all small-sized instances with 15 and 20 customers are now solved to proven optimality. For the other instance sizes with 25 and 30 customers, there remains a relatively small gap of 0.2% and 1.2%, respectively.

In several VRPTW variants, the non-robust *subset-row inequalities* (SRIs, Jepsen *et al.*, 2008) turned out to be a successful method to further strengthen the linear relaxation. Pre-tests have shown that adding SRIs to our VRP-DTW-specific BPC algorithm makes the labeling algorithm significantly more difficult and leads to a considerable increase in computation time. Therefore, we do not consider SRIs.

5.5.3 Managerial Insights

We now present three analyses on (i) the impact of combined truck-and-drone routing compared to classical truck routing, (ii) the impact of different drones that only differ in their flying range, and (iii) the comparison of an as-late-as-possible schedule with an as-early-as-possible schedule.

Impact of combined truck-and-drone routing We analyze the impact of combined truck-and-drone routing in comparison to classical truck routing. By removing all truck alone and drone arcs from the artificial network, the BPC algorithm solves a VRPTW with the objective to minimize the total route duration. Of course, this problem is computationally much easier to solve than the VRP-DTW so that all 80 instances were solved to proven optimality within approximately 160 seconds of computation time on average over all customer sizes. Since we are only interested in an upper bound (or optimal solution) to the objective function, we do not present any further computational results for the VRPTW solutions.

Figure 5.3 shows the average route duration for classical truck routing (VRPTW) compared to truck-and-drone routing with different types of drones, meaning that each type of drone has a different speed relative to the truck. If trucks and drones have the same speed ($\beta = 1$), the average route duration can be reduced by approximately 30% compared to classical truck routing consistent over all instance sizes. Whenever the drone is faster than the truck ($\beta = 3$), the total duration can be further decreased by

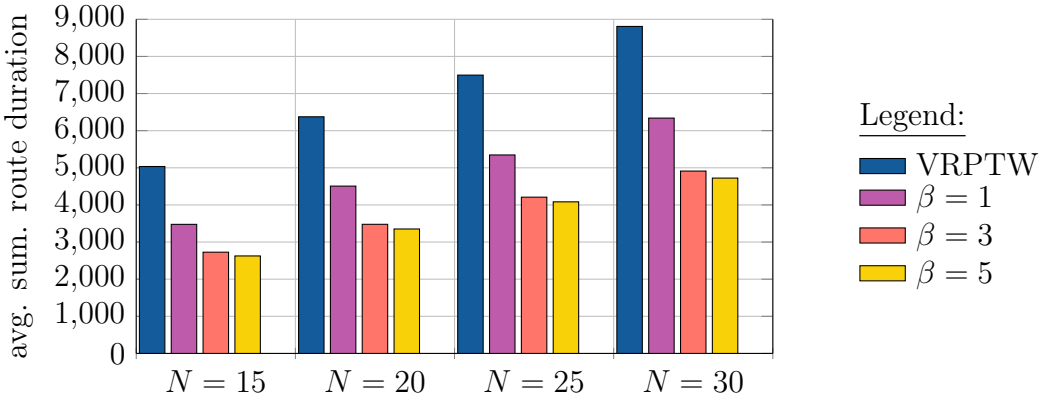


Figure 5.3: Comparison between VRPTW and VRP-DTW with different drone speeds.

approximately 15 percentage points. Increasing the drone speed to $\beta = 5$ or above does result in considerably additional time savings, consistent with the findings in Chapter 4.

Analysis of different drone flying ranges In the following, we assume a drone speed of $\beta = 3$ and analyze different flying ranges δ for the drone. To generate such flying ranges, we follow the method used in Chapter 4: We introduce a new parameter $\gamma \in [0, 100]$ and limit the instance-specific flying range so that $t^{\text{dr}}_{i^{\text{dr}}k} + t^{\text{dr}}_{k,j^{\text{dr}}} \leq \delta$ is fulfilled for the largest γ with $\gamma \cdot |\mathcal{A}^{drone}|/100$ drone arcs. Thus, all other $100 - \gamma$ percent of the drone arcs can be removed from the network.

Figure 5.4 visualizes the effects of using different types of drones with different flying ranges.

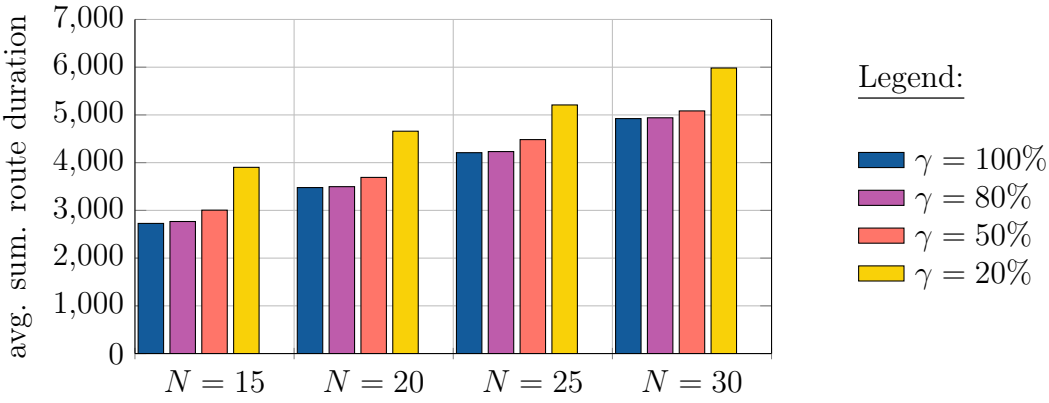


Figure 5.4: Comparison of different drone flying ranges.

Obviously, the more customers can be reached by a drone, the lower is the total duration over all routes. However, this figure shows that it is not necessary to use a

drone with an unlimited flying range of $\gamma = 100\%$. The time savings compared to a drone that can only reach 80% of all customers is negligible. Using drones with a flying range that can reach half of all customers results in an average increase of the total delivery time by approximately 6% compared to an unlimited drone. Only if the drone's flying range is further limited to reach a maximum of only 20% of all customers, the total delivery time increases by approximately 30% on average.

Minimizing route duration vs. minimizing completion time With only slight adjustments, our BPC algorithm can also solve a VRP-DTW instance with the objective to minimize the total completion time. This allows us to analyze the impact of both objective functions (minimizing the total route duration and minimizing the total completion time) on the total time needed to serve all customers. Table 5.4 presents the average completion time (*Compl. time*), the average duration (*Duration*) and the average percentage savings in time (*Savings (%)*) that can be realized with an as-late-as-possible schedule compared to an as-early-as-possible schedule. All values are grouped by the number of customers considered in the instance. It shows that the average time

$\|N\|$	Compl. time	Duration	Savings (%)
15	7,819	2,727	65.12
20	9,194	3,478	62.17
25	11,694	4,209	64.00
30	14,161	4,923	65.24
Avg.	10,717	3,834	64.13

Table 5.4: Comparison of route duration and completion time.

needed to serve all customers can be reduced by approximately 64% on average over all instances if both vehicles can leave the depot at any time instead of starting immediately at e_0. At first sight, these values seem high, but waiting times are often long, especially for routes that serve customers with late time windows.

5.6 Conclusions and Outlook

In the paper at hand, we studied the VRP-DTW that generalizes the VRP-D by the existence of time windows. Further we presented the first exact BPC algorithm to solve the VRP-DTW with the objective to minimize the total duration over all routes. The presentation mainly focused on the modeling and the effective solution of the column-generation pricing problem, the SPPRC. Further, non-robust capacity cuts and DNE were implemented to strengthen the linear relaxation of the set-partitioning formulation. A computational study showed that the addition of capacity cuts alone is not enough to decrease the computation times compared to a branch-and-price algorithm. With with

addition of DNE, the solution times were considerably reduced and nine more instances were solved to proven optimality. Managerial insights showed that combined truck-and-drone routing can reduce the average delivery times by up to 45% (depending on the speed of a drone) compared to classical truck routing. We also showed that it is not necessary to use drones with an "unlimited" flying range. The average delivery time increases by only 6% when using a drone that can reach at most 50% of all customers. Finally, we showed that an as-late-as-possible schedule can considerably reduce the delivery times compared to an as-early-as-possible schedule.

Our version of the VRP-DTW is based on the assumption that a drone always waits on the ground instead of hovering, which implies that there is no energy consumption that affects its flying range while waiting. Of course, this assumption can be changed by assuming that a drone always hovers while waiting. In this case, the proposed BPC algorithm is no longer applicable. When propagating a label along a path through the artificial network, the latest departure time for both vehicles may change, resulting in (additional) waiting times at a customer node. The labeling algorithm in its current form cannot ensure that existing drone flights along a truck-and-drone path remain feasible in such a situation. New techniques, similar to those that ensure *ride time constraints* in a *Dial-A-Ride Problem*, need to be developed to keep track of the feasibility of a truck-and-drone path.

Bibliography

Amazon Prime Air (2023). Amazon is launching ultra-fast drone deliveries in Italy, the UK, and a third location in the U.S. `https://www.aboutamazon.com/news/operations/amazon-prime-air-drone-delivery-updates`.

Bakir, I. and Tiniç, G. Ö. (2020). Optimizing drone-assisted last-mile deliveries: The vehicle routing problem with flexible drones. Optimization online. `https://optimization-online.org/?p=16382`.

Baldacci, R., Christofides, N., and Mingozzi, A. (2008). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, **115**(2), 351–385.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Blufstein, M., Lera-Romero, G., and Soulignac, F. J. (2024). Decremental state-space relaxations for the basic traveling salesman problem with a drone. *INFORMS Journal on Computing*, **36**(4), 1064–1083.

Bode, C. and Irnich, S. (2015). In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. *Transportation Science*, **49**(2), 369–383.

Carlsson, J. G. and Song, S. (2018). Coordinated logistics with a truck and a drone. *Management Science*, **64**(9), 4052–4069.

Chen, C., Demir, E., and Huang, Y. (2021a). An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and delivery robots. *European Journal of Operational Research*, **294**(3), 1164–1180.

Chen, C., Demir, E., Huang, Y., and Qiu, R. (2021b). The adoption of self-driving delivery robots in last mile logistics. *Transportation Research Part E: Logistics and Transportation Review*, **146**, 102214.

Chung, S. H., Sah, B., and Lee, J. (2020). Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions. *Computers & Operations Research*, **123**, 105004.

Coindreau, M.-A., Gallay, O., and Zufferey, N. (2021). Parcel delivery cost minimization with time window constraints using trucks and drones. *Networks*, **78**(4), 400–420.

Das, D. N., Sewani, R., Wang, J., and Tiwari, M. K. (2021). Synchronized truck and drone routing in package delivery logistics. *IEEE Transactions on Intelligent Transportation Systems*, **22**(9), 5772–5782.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Desaulniers, G., Madsen, O., and Ropke, S. (2014). The vehicle routing problem with time windows. In P. Toth and D. Vigo, editors, *Vehicle Routing*, volume 18 of *MOS-SIAM Series on Optimization*, chapter 5, pages 119–159. Society for Industrial and Applied Mathematics, Philadelphia, PA, Philadelphia, PA.

Federal Aviation Administraion (2023). The FAA authorizes UPS flight forward and uAvionix to operate drones beyond visual line of sight. `https://www.faa.gov/newsroom/faa-authorizes-ups-uavionix`.

Gamache, M., Soumis, F., Marquis, G., and Desrosiers, J. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations Research*, **47**(2), 247–263.

Goodchild, A. and Toy, J. (2018). Delivery by drone: An evaluation of unmanned aerial vehicle technology in reducing CO_2 emissions in the delivery service industry. *Transportation Research Part D: Transport and Environment*, **61**, 58–67.

Irnich, S. (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**(2), 497–511.

Joerss, M., Schröder, J., Neuhaus, F., Klink, C., and Mann, F. (2016). Parcel delivery: The future of last mile. McKinsey&Company, Brochure. `https://bdkep.de/files/bdkep-dateien/pdf/2016_the_future_of_last_mile.pdf`

Kuo, R., Lu, S.-H., Lai, P.-Y., and Mara, S. T. W. (2022). Vehicle routing problem with drones considering time windows. *Expert Systems with Applications*, **191**, 116264.

Li, H. and Wang, F. (2022). Branch-price-and-cut for the truck–drone routing problem with time windows. *Naval Research Logistics (NRL)*, **70**(2), 184–204.

Li, H., Wang, H., Chen, J., and Bai, M. (2020). Two-echelon vehicle routing problem with time windows and mobile satellites. *Transportation Research Part B: Methodological*, **138**, 179–201.

Macrina, G., Di Puglia Pugliese, L., Guerriero, F., and Laporte, G. (2020). Drone-aided routing: A literature review. *Transportation Research Part C: Emerging Technologies*, **120**, 102762.

Madani, B. and Ndiaye, M. (2022). Hybrid truck-drone delivery systems: A systematic literature review. *IEEE Access*, **10**, 92854–92878.

Martinelli, R., Poggi, M., and Subramanian, A. (2013). Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, **40**(8), 2145–2160.

Moshref-Javadi, M. and Winkenbach, M. (2021). Applications and research avenues for drone-based models in logistics: A classification and review. *Expert Systems with Applications*, **177**, 114854.

Otto, A., Agatz, N., Campbell, J., Golden, B., and Pesch, E. (2018). Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks*, **72**(4), 411–458.

Poikonen, S., Wang, X., and Golden, B. (2017). The vehicle routing problem with drones: Extended models and connections. *Networks*, **70**(1), 34–43.

Pugliese, L. D. P. and Guerriero, F. (2017). Last-mile deliveries by using drones and classical vehicles. In *Springer Proceedings in Mathematics & Statistics*, pages 557–565. Springer International Publishing.

Pugliese, L. D. P., Macrina, G., and Guerriero, F. (2020). Trucks and drones cooperation in the last-mile delivery process. *Networks*, **78**(4), 371–399.

Pugliese, L. D. P., Guerriero, F., and Scutellá, M. G. (2021). The last-mile delivery process with trucks and drones under uncertain energy consumption. *Journal of Optimization Theory and Applications*, **191**(1), 31–67.

Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, **3**(3), 255–273.

Roberti, R. and Mingozzi, A. (2014). Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*, **48**(3), 413–424.

Roberti, R. and Ruthmair, M. (2021). Exact methods for the traveling salesman problem with drone. *Transportation Science*, **55**(2), 315–335.

Savelsbergh, M. W. P. (1992). The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, **4**(2), 146–154.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, **35**(2), 254–265.

Tilk, C. and Irnich, S. (2017). Dynamic programming for the minimum tour duration problem. *Transportation Science*, **51**(2), 549–565.

Tilk, C., Rothenbächer, A.-K., Gschwind, T., and Irnich, S. (2017). Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research*, **261**(2), 530–539.

Valério de Carvalho, J. M. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, **86**, 629–659.

Wang, X., Poikonen, S., and Golden, B. (2017). The vehicle routing problem with drones: several worst-case results. *Optimization Letters*, **11**(4), 679–697.

Yin, Y., Li, D., Wang, D., Ignatius, J., Cheng, T., and Wang, S. (2023). A branch-and-price-and-cut algorithm for the truck-based drone delivery routing problem with time windows. *European Journal of Operational Research*, **309**(3), 1125–1144.

Zhen, L., Gao, J., Tan, Z., Wang, S., and Baldacci, R. (2023). Branch-price-and-cut for trucks and drones cooperative delivery. *IISE Transactions*, **55**(3), 271–287.

Zhou, H., Qin, H., Cheng, C., and Rousseau, L.-M. (2023). An exact algorithm for the two-echelon vehicle routing problem with drones. *Transportation Research Part B: Methodological*, **168**, 124–150.

Chapter 6

Conclusions

The aim of this thesis is to contribute to the development of exact and heuristic solution methods for a classical routing problem and three routing problems with synchronization constraints. As a classical routing problem, we considered the GTSP, which is a generalization of the well-known TSP. We developed a straightforward basic ILS and a refined ILS to solve the problem heuristically. BPC algorithms became the method of choice to solve all routing problems with synchronization constraints, namely the LMDPSL, the VRP-D, and the VRP-DTW.

In Chapter 2, we developed three novel GTSP neighborhoods that allow the simultaneous permutation of the visiting sequence of the clusters and the vertex selection of each cluster. Together with known GTSP neighborhoods from the literature, they are used within a VND in the local search part of the ILS. The significant contribution of our new neighborhoods to the success of the overall algorithm became especially clear in our refined ILS. For example, the adapted Gutin neighborhood is used with highest priority in the nested VND II, and the Balas-Simonetti neighborhood with $k = 8$ can further improve so-called elite solutions. The results showed that the refined ILS is particularly powerful on the `GTSP_LIB`. Compared to the literature, the results are not clear-cut and there is no unique winner algorithm that outperforms all the others on each benchmark library.

In Chapter 3, we introduced the LMDPSL that integrates public transport services into last-mile delivery networks. We developed exact and heuristic BPC algorithms to solve instances with up to 150 customers to proven optimality or with a relatively small optimality gap. The good performance of the overall algorithms allowed us not only to solve the operational planning problem but also to address several questions arising from a more tactical perspective. The studies showed that scenarios with a high number of lines and stops almost always have the lowest routing costs and require a smaller number of city freighters, regardless of the number of customers. Scenarios with scheduled lines operating in a circle around the city center always lead to the highest routing costs and number of used city freighters because a small number of stops (or a missing stop at a central station) results in longer distances to replenish the city freighters. We further showed that the distribution of customer locations has a significant impact on the performance of the BPC algorithm, but there is almost no impact on the different studies comparing scenarios or objectives.

In Chapter 4, we proposed an implicit bidirectional labeling algorithm to solve the

SPPRC pricing problem within the proposed BPC algorithm. We made use of an artificial network originally introduced by Roberti and Ruthmair (2021). At first sight it seemed rather unpromising to develop a bidirectional labeling algorithm over this network because it is a directed multi-graph. By exploiting the symmetries of the original problem, we showed that an implicit bidirectional labeling algorithm can be developed. Since forward and backward labeling create exactly the same partial paths, it suffices to propagate labels only in forward direction and merge them in a merge procedure. Two different merge procedures are necessary, depending on the positions of truck and drone. In the end, the BPC algorithm solves instance with up to 49 (29) customers considering a cost-minimization (duration-minimization) objective within one hour of computation time.

In Chapter 5, we studied the VRP-DTW that generalizes the VRP-D by the existence of time windows. We presented the first exact BPC algorithm to solve the VRP-DTW with the objective to minimize the total duration over all routes. Our presentation mainly focused on the effective solution of the SPPRC pricing problem and the proper synchronization of trucks and drones to obtain a route with minimal duration. A computational study showed that the addition of non-robust capacity cuts along with dynamic *ng*-neighborhood extension mainly contributes to the success of the overall BPC algorithm. In total, instances with up to 30 customers can be solved to proven optimality within two hours of computation time. On average, the gap over all considered instances is less than 0.5%. Managerial insights showed that synchronized truck-and-drone routing can reduce the delivery time by up to 45% on average compared to classical truck routing. In addition, we showed that an as-late-as-possible schedule can considerably reduce the delivery times compared to an as-early-as-possible schedule.

Bibliography

Achterberg, T. (2007). *Constraint Integer Programming*. Ph.D. thesis, Technische Universität Berlin, Fakultät II – Mathematik und Naturwissenschaften, Berlin, Germany.

Amazon Prime Air (2023). Amazon is launching ultra-fast drone deliveries in Italy, the UK, and a third location in the U.S. `https://www.aboutamazon.com/news/operations/amazon-prime-air-drone-delivery-updates`, Last accessed 2024-04-30.

Anderluh, A., Hemmelmayr, V. C., and Nolz, P. C. (2019). Sustainable logistics with cargo bikes—methods and applications. In J. Faulin, S. E. Grasman, A. A. Juan, and P. Hirsch, editors, *Sustainable Transportation and Smart Logistics*, chapter 8, pages 207–232. Elsevier.

Augerat, P., Belenguer, J. M., Benavent, E., Corberán, A., Naddef, D., and Rinaldi, G. (1995). Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical report, Institut National Polytechnique, 38 - Grenoble (France).

Bakir, I. and Tiniç, G. Ö. (2020). Optimizing drone-assisted last-mile deliveries: The vehicle routing problem with flexible drones. Optimization online. `https://optimization-online.org/?p=16382`.

Balas, E. (1999). New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research*, **86**, 529–558.

Balas, E. and Simonetti, N. (2001). Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS Journal on Computing*, **13**(1), 56–75.

Baldacci, R., Christofides, N., and Mingozzi, A. (2008). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, **115**(2), 351–385.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Baldacci, R., Mingozzi, A., and Roberti, R. (2012). New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*, **24**(3), 356–371.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, **46**(3), 316–329.

Behiri, W., Belmokhtar-Berraf, S., and Chu, C. (2018). Urban freight transport using passenger rail network: Scientific issues and quantitative analysis. *Transportation Research Part E: Logistics and Transportation Review*, **115**, 227–245.

Ben-Arieh, D., Gutin, G., Penn, M., Yeo, A., and Zverovitch, A. (2003). Transformations of generalized ATSP into ATSP. *Operations Research Letters*, **31**(5), 357–365.

Bentley, J. J. (1992). Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, **4**(4), 387–411.

BIEK (2021). KEP-Studie 2021: Analyse des Marktes in Deutschland: Möglichmacher in bewegten Zeiten. Bundesverband Paket und Expresslogistik e. V. (BIEK) `https://www.biek.de/files/biek/downloads/papiere/BIEK_KEP-Studie_2021.pdf`, Berlin, Germany (in German).

Blufstein, M., Lera-Romero, G., and Soulignac, F. J. (2024). Decremental state-space relaxations for the basic traveling salesman problem with a drone. *INFORMS Journal on Computing*, **36**(4), 1064–1083.

Bode, C. and Irnich, S. (2012). Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research*, **60**(5), 1167–1182.

Bode, C. and Irnich, S. (2015). In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. *Transportation Science*, **49**(2), 369–383.

Bodin, L. D. (1975). A taxonomic structure for vehicle routing and scheduling problems. *Computers & Urban Society*, **1**(1), 11–29.

Bontoux, B. (2008). *Techniques hybrides de recherche exacte et approchée : application à des problèmes de transport*. Ph.D. thesis, Université d'Avignon. Français.

Bontoux, B., Artigues, C., and Feillet, D. (2010). A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem. *Computers & Operations Research*, **37**(11), 1844–1852.

Boysen, N., Schwerdfeger, S., and Weidinger, F. (2018). Scheduling last-mile deliveries with truck-based autonomous robots. *European Journal of Operational Research*, **271**(3), 1085–1099.

Cacchiani, V., Muritiba, A., Negreiros, M., and Toth, P. (2011). A multistart heuristic for the equality generalized traveling salesman problem. *Networks*, **57**, 231–239.

Carlsson, J. G. and Song, S. (2018). Coordinated logistics with a truck and a drone. *Management Science*, **64**(9), 4052–4069.

Chen, C., Demir, E., and Huang, Y. (2021a). An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and delivery robots. *European Journal of Operational Research*, **294**(3), 1164–1180.

Chen, C., Demir, E., Huang, Y., and Qiu, R. (2021b). The adoption of self-driving delivery robots in last mile logistics. *Transportation Research Part E: Logistics and Transportation Review*, **146**, 102214.

Chung, S. H., Sah, B., and Lee, J. (2020). Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions. *Computers & Operations Research*, **123**, 105004.

Coindreau, M.-A., Gallay, O., and Zufferey, N. (2021). Parcel delivery cost minimization with time window constraints using trucks and drones. *Networks*, **78**(4), 400–420.

Costa, L., Contardo, C., and Desaulniers, G. (2019). Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, **53**, 946–985.

Cuda, R., Guastaroba, G., and Speranza, M. G. (2015). A survey on two-echelon routing problems. *Computers & Operations Research*, **55**, 185–199.

Daganzo, C. (2005). *Logistics Systems Analysis*. Springer, Berlin, Germany, 4th edition.

Dakin, R. J. (1965). A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, **8**(3), 250–255.

D'Andrea, R. (2014). Guest editorial Can drones deliver? *IEEE Transactions on Automation Science and Engineering*, **11**(3), 647–648.

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

Dantzig, G. B. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, **8**(1), 101–111.

Das, D. N., Sewani, R., Wang, J., and Tiwari, M. K. (2021). Synchronized truck and drone routing in package delivery logistics. *IEEE Transactions on Intelligent Transportation Systems*, **22**(9), 5772–5782.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Desaulniers, G., Madsen, O., and Ropke, S. (2014). The vehicle routing problem with time windows. In Toth and Vigo (2014), chapter 5, pages 119–159.

Drexl, M. (2012). Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. *Transportation Science*, **46**(3), 297–316.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, **42**(5), 977–978.

Dueck, G. (1993). New optimization heuristics. *Journal of Computational Physics*, **104**(1), 86–92.

Eiben, A. E. and Smith, J. E. (2015). What is an evolutionary algorithm? In *Introduction to Evolutionary Computing*, Natural Computing Series, pages 25–48. Springer, Berlin/Heidelberg.

El Krari, M., Ahiod, B., and El Benani, Y. B. (2020). A memetic algorithm based on breakout local search for the generalized traveling salesman problem. *Applied Artificial Intelligence*, **34**(7), 537–549.

El Krari, M., Ahiod, B., and El Benani, Y. B. (2021). A pre-processing reduction method for the generalized travelling salesman problem. *Operational Research*, **21**, 2543–2591.

Elbert, R. and Rentschler, J. (2021). Freight on urban public transportation: A systematic literature review. *Research in Transportation Business & Management*, **45**(Part A), 100679.

Fatnassi, E., Chaouachi, J., and Klibi, W. (2015). Planning and operating a shared goods and passengers on-demand rapid transit system for sustainable city-logistics. *Transportation Research Part B: Methodological*, **81**, 440–460.

Federal Aviation Administraion (2023). The FAA authorizes UPS flight forward and uAvionix to operate drones beyond visual line of sight. `https://www.faa.gov/newsroom/faa-authorizes-ups-uavionix`, Last accessed 2024-04-30.

Feillet, D., Dejax, P., Gendreau, M., and Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, **44**(3), 216–229.

Fischetti, M., Gonzáles, J., and Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, **45**(3), 378–394.

Fischetti, M., Salazar-Gonzalez, J.-J., and Toth, P. (2007). The generalized traveling salesman and orienteering problems. In Gutin and Punnen (2007), pages 609–662.

Furmanik, G. (2019). How much does retail space cost? *realla*. `https://blog.realla.co.uk/how-much-does-retail-space-cost`.

Gamache, M., Soumis, F., Marquis, G., and Desrosiers, J. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations Research*, **47**(2), 247–263.

Gauthier, J. B. and Irnich, S. (2024). Inter-depot moves and dynamic-radius search for multi-depot vehicle routing problems. *Discrete Applied Mathematics*, **346**, 131–153.

Ghilas, V., Demir, E., and Woensel, T. V. (2016a). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. *Computers & Operations Research*, **72**, 12–30.

Ghilas, V., Demir, E., and Woensel, T. V. (2016b). The pickup and delivery problem with time windows and scheduled lines. *INFOR: Information Systems and Operational Research*, **54**(2), 147–167.

Ghilas, V., Demir, E., and Woensel, T. V. (2016c). A scenario-based planning for the pickup and delivery problem with time windows, scheduled lines and stochastic demands. *Transportation Research Part B: Methodological*, **91**, 34–51.

Ghilas, V., Cordeau, J.-F., Demir, E., and Woensel, T. V. (2018). Branch-and-price for the pickup and delivery problem with time windows and scheduled lines. *Transportation Science*, **52**(5), 1191–1210.

Glover, F. (1996). Finding a best traveling salesman 4-opt move in the same time as a best 2-opt move. *Journal of Heuristics*, **2**, 169–179.

Goeke, D., Gschwind, T., and Schneider, M. (2019). Upper and lower bounds for the vehicle-routing problem with private fleet and common carrier. *Discrete Applied Mathematics*, **264**, 43–61.

Goodchild, A. and Toy, J. (2018). Delivery by drone: An evaluation of unmanned aerial vehicle technology in reducing CO_2 emissions in the delivery service industry. *Transportation Research Part D: Transport and Environment*, **61**, 58–67.

Gutin, G. and Karapetyan, D. (2009). Generalized traveling salesman problem reduction algorithms. arXiv 0804.0735v2.

Gutin, G. and Karapetyan, D. (2010). A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, **9**(1), 47–60.

Gutin, G. and Punnen, A. P., editors (2007). *The Traveling Salesman Problem and Its Variations*. Springer, Boston, MA.

Gutin, G., Yeo, A., and Zverovitch, A. (2007). Exponential neighborhoods and domination analysis for the TSP. In Gutin and Punnen (2007), pages 223–256.

Gutin, G., Karapetyan, D., and Krasnogor, N. (2008). Memetic algorithm for the generalized asymmetric traveling salesman problem. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, pages 199–210. Springer Berlin Heidelberg.

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, **130**(3), 449–467.

Hansen, P. and Mladenović, N. (2005). Variable neighborhood search. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 8, pages 211–238. Springer, Boston, MA.

Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, **126**, 106–130.

Helsgaun, K. (2015). Solving the equality generalized traveling salesman problem using the Lin–Kernighan–Helsgaun Algorithm. *Mathematical Programming Computation*, **7**(3), 269–287.

Henry-Labordere, A. (1969). The record balancing problem: A dynamic programming solution of a generalized travelling salesman problem. *RIRO B-2*, pages 43–49.

Heßler, K. and Irnich, S. (2023). Partial dominance in branch-price-and-cut for the basic multicompartment vehicle-routing problem. *INFORMS Journal on Computing*, **35**(1), 50–65.

Hoos, H. H. and Stützle, T. (2005). *Stochastic Local Search*. Morgan Kaufmann, Amsterdam.

Hu, B. and Raidl, G. R. (2008). Effective neighborhood structures for the generalized traveling salesman problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 36–47. Springer Berlin Heidelberg.

Irnich, S. (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

Irnich, S., Funke, B., and Grünert, T. (2006). Sequential search and its application to vehicle-routing problems. *Computers & Operations Research*, **33**(8), 2405–2429.

Irnich, S., Toth, P., and Vigo, D. (2014). The family of vehicle routing problems. In Toth and Vigo (2014), chapter 1, pages 1–33.

ITA (2023). Unmanned aircraft systems (UAS). International Trade Administration (ITA) `https://www.trade.gov/unmanned-aircraft-systems`, Last accessed 2023-05-03.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**(2), 497–511.

Joerss, M., Schröder, J., Neuhaus, F., Klink, C., and Mann, F. (2016). Parcel delivery: The future of last mile. McKinsey&Company, Brochure. `https://www.trade.gov/unmanned-aircraft-systems`.

Johnson, D. S. and McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 8, pages 215–310. Wiley, Chichester.

Karapetyan, D. and Gutin, G. (2011). Lin–Kernighan heuristic adaptations for the generalized traveling salesman problem. *European Journal of Operational Research*, **208**(3), 221–232.

Karapetyan, D. and Gutin, G. (2012). Efficient local search algorithms for known and new neighborhoods for the generalized traveling salesman problem. *European Journal of Operational Research*, **219**(2), 234–251.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1987). Optimization by simulated annealing. In *Readings in Computer Vision*, pages 606–615. Elsevier.

Kuo, R., Lu, S.-H., Lai, P.-Y., and Mara, S. T. W. (2022). Vehicle routing problem with drones considering time windows. *Expert Systems with Applications*, **191**, 116264.

Laporte, G. and Semet, F. (1999). Computational evaluation of a transformation procedure for the symmetric generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, **37**(2), 114–120.

Laporte, G., Asef-Vaziri, A., and Sriskandarajah, C. (1996). Some applications of the generalized travelling salesman problem. *Journal of the Operational Research Society*, **47**(12), 1461–1467.

Lera-Romero, G., Miranda Bront, J. J., and Soulignac, F. J. (2022). Dynamic programming for the time-dependent traveling salesman problem with time windows. *INFORMS Journal on Computing*, **34**(6), 3292–3308.

Li, H. and Wang, F. (2022). Branch-price-and-cut for the truck–drone routing problem with time windows. *Naval Research Logistics (NRL)*, **70**(2), 184–204.

Li, H., Wang, H., Chen, J., and Bai, M. (2020). Two-echelon vehicle routing problem with time windows and mobile satellites. *Transportation Research Part B: Methodological*, **138**, 179–201.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, **44**, 2245–2269.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, **21**(2), 498–516.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). Iterated local search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 320–353. Springer, New York, NY.

Lübbecke, M. E. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, **53**(6), 1007–1023.

Macrina, G., Di Puglia Pugliese, L., Guerriero, F., and Laporte, G. (2020). Drone-aided routing: A literature review. *Transportation Research Part C: Emerging Technologies*, **120**, 102762.

Madani, B. and Ndiaye, M. (2022). Hybrid truck-drone delivery systems: A systematic literature review. *IEEE Access*, **10**, 92854–92878.

Martinelli, R., Poggi, M., and Subramanian, A. (2013). Improved bounds for large scale capacitated arc routing problem. *Computers & Operations Research*, **40**(8), 2145–2160.

Masson, R., Trentini, A., Lehuédé, F., Malhéné, N., Péton, O., and Tlahig, H. (2015). Optimization of a city logistics transportation system with mixed passengers and goods. *EURO Journal on Transportation and Logistics*, **6**(1), 81–109.

Mestria, M., Ochi, L. S., and de Lima Martins, S. (2013). GRASP with path relinking for the symmetric Euclidean clustered traveling salesman problem. *Computers & Operations Research*, **40**(12), 3218–3229.

Moshref-Javadi, M. and Winkenbach, M. (2021). Applications and research avenues for drone-based models in logistics: A classification and review. *Expert Systems with Applications*, **177**, 114854.

Mourad, A., Puchinger, J., and Chu, C. (2019). A survey of models and algorithms for optimizing shared mobility. *Transportation Research Part B: Methodological*, **123**, 323–346.

Mourad, A., Puchinger, J., and Woensel, T. V. (2020). Integrating autonomous delivery service into a passenger transportation system. *International Journal of Production Research*, **59**(7), 2116–2139.

Noon, C. E. (1988). *The generalized traveling salesman problem.* Ph.D. thesis, University of Michigan.

Noon, C. E. and Bean, J. C. (1991). A Lagrangian based approach for the asymmetric generalized traveling salesman problem. *Operations Research*, **39**(4), 623–632.

Noon, C. E. and Bean, J. C. (1993). An efficient transformation of the generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, **31**(1), 39–44.

OECD (2020). E-commerce in the time of COVID-19. `https://www.oecd.org/coronavirus/policy-responses/e-commerce-in-the-time-of-covid-19-3a2b78e8/`.

Otto, A., Agatz, N., Campbell, J., Golden, B., and Pesch, E. (2018). Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks*, **72**(4), 411–458.

Ozturk, O. and Patrick, J. (2018). An optimization model for freight transport using urban rail transit. *European Journal of Operational Research*, **267**(3), 1110–1121.

Paradiso, R., Roberti, R., Laganá, D., and Dullaert, W. (2020). An exact solution framework for multitrip vehicle-routing problems with time windows. *Operations Research*, **68**(1), 180–198.

Pecin, D., Pessoa, A., Poggi, M., and Uchoa, E. (2017a). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, **9**, 61–100.

Pecin, D., Contardo, C., Desaulniers, G., and Uchoa, E. (2017b). New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal Computing*, **29**(3), 489–502.

Pessoa, A., Sadykov, R., Uchoa, E., and Vanderbeck, F. (2020). A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, **183**(1-2), 483–523.

Pintea, C. M., Pop, P. C., and Chira, C. (2017). The generalized traveling salesman problem solved with ant algorithms. *Complex Adaptive Systems Modeling*, **5**(1).

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, **34**(8), 2403–2435.

Poikonen, S., Wang, X., and Golden, B. (2017). The vehicle routing problem with drones: Extended models and connections. *Networks*, **70**(1), 34–43.

Pop, P. C. (2012). *Generalized Network Design Problems: Modeling and Optimization*. DE GRUYTER.

Pugliese, L. D. P. and Guerriero, F. (2017). Last-mile deliveries by using drones and classical vehicles. In *Springer Proceedings in Mathematics & Statistics*, pages 557–565. Springer International Publishing.

Pugliese, L. D. P., Macrina, G., and Guerriero, F. (2020). Trucks and drones cooperation in the last-mile delivery process. *Networks*, **78**(4), 371–399.

Pugliese, L. D. P., Guerriero, F., and Scutellá, M. G. (2021). The last-mile delivery process with trucks and drones under uncertain energy consumption. *Journal of Optimization Theory and Applications*, **191**(1), 31–67.

Reihaneh, M. and Karapetyan, D. (2012). An efficient hybrid ant colony system for the generalized traveling salesman problem. *Algorithmic Operations Research*, **7**, 22–29.

Ren, X., Wang, X., Wang, Z., and Wu, T. (2020). Parallel dna algorithms of generalized traveling salesman problem-based bioinspired computing model. *International Journal of Computational Intelligence Systems*, **14**(1), 228.

Renaud, J. and Boctor, F. F. (1998). An efficient composite heuristic for the symmetric generalized traveling salesman problem. *European Journal of Operational Research*, **108**(3), 571–584.

Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, **3**(3), 255–273.

Roberti, R. and Mingozzi, A. (2014). Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*, **48**(3), 413–424.

Roberti, R. and Ruthmair, M. (2021). Exact methods for the traveling salesman problem with drone. *Transportation Science*, **55**(2), 315–335.

Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, **40**(4), 455–472.

Saksena, J. P. (1970). Mathematical model of scheduling clients through welfare agencies. *Journal of the Canadian Operational Research Society*, **8**, 185–200.

Savelsbergh, M. and van Woensel, T. (2016). 50th Anniversary invited article—City logistics: Challenges and opportunities. *Transportation Science*, **50**(2), 579–590.

Savelsbergh, M. W. P. (1992). The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, **4**(2), 146–154.

Schiffer, M., Schneider, M., Walther, G., and Laporte, G. (2019). Vehicle routing and location routing with intermediate stops: A review. *Transportation Science*, **53**(2), 319–343.

Schneider, M., Stenger, A., and Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, **48**(4), 500–520.

Schocke, K.-O., Schäfer, P. K., Höhl, S., and Gilbert, A. (2020). *Last mile tram: Empirische Forschung zum Einsatz einer Güterstraßenbahn am Beispiel Frankfurt am Main*. Frankfurt University of Applied Sciences, Frankfurt am Main. (in German).

Schrijver, A. (2003). *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, Berlin Heidelberg.

Schröder, C., Gauthier, J. B., Gschwind, T., and Schneider, M. (2020). In-depth analysis of granular local search for capacitated vehicle routing. Working Paper DPO-2020-03, Deutsche Post Chair – Optimization of Distribution Networks, RWTH Aachen University, Aachen, Germany.

Silberholz, J. and Golden, B. (2007). The generalized traveling salesman problem: A new genetic algorithm approach. In *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, pages 165–181. Springer, New York, NY.

Smith, S. L. and Imeson, F. (2017). GLNS: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, **87**, 1–19.

Snyder, L. and Daskin, M. (2006). A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*, **174**(1), 38–53.

Soares, R., Marques, A., Amorim, P., and Parragh, S. N. (2024). Synchronisation in vehicle routing: Classification schema, modelling framework and literature review. *European Journal of Operational Research*, **313**(3), 817–840.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, **35**(2), 254–265.

Subramanian, A., Drummond, L. M. A., Bentes, C., Ochi, L. S., and Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, **37**(11), 1899–1911.

Tamke, F. and Buscher, U. (2021). A branch-and-cut algorithm for the vehicle routing problem with drones. *Transportation Research Part B: Methodological*, **144**, 174–203.

Tasgetiren, M. F., Suganthan, P. N., and Pan, Q. Q. (2007). A discrete particle swarm optimization algorithm for the generalized traveling salesman problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation - GECCO 2007*, pages 158–167.

Tilk, C. and Irnich, S. (2017). Dynamic programming for the minimum tour duration problem. *Transportation Science*, **51**(2), 549–565.

Tilk, C., Rothenbächer, A.-K., Gschwind, T., and Irnich, S. (2017). Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research*, **261**(2), 530–539.

Toth, P. and Vigo, D. (2002). An overview of vehicle routing problems. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 1, pages 1–26. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Toth, P. and Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, **15**(4), 333–346.

Toth, P. and Vigo, D., editors (2014). *Vehicle Routing*, volume 18 of *MOS-SIAM Series on Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, Philadelphia, PA, second edition edition.

United Nations (2021). COVID-19 and e-commerce: A global review. UNCTAD/DTL/STICT/2020/13 `https://unctad.org/system/files/official-document/dtlstict2020d13_en.pdf`.

Valério de Carvalho, J. M. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, **86**, 629–659.

Vanderbeck, F. (2005). Implementing mixed integer column generation. In Desaulniers *et al.* (2005), chapter 12, pages 331–358.

Wang, X., Poikonen, S., and Golden, B. (2017). The vehicle routing problem with drones: several worst-case results. *Optimization Letters*, **11**(4), 679–697.

Wang, Z. and Sheu, J.-B. (2019). Vehicle routing problem with drones. *Transportation Research Part B: Methodological*, **122**, 350–364.

Xia, Y., Zeng, W., Zhang, C., and Yang, H. (2023). A branch-and-price-and-cut algorithm for the vehicle routing problem with load-dependent drones. *Transportation Research Part B: Methodological*, **171**, 80–110.

Yang, J., Shi, X., Marchese, M., and Liang, Y. (2008). An ant colony optimization method for generalized TSP problem. *Progress in Natural Science*, **18**(11), 1417–1422.

Yin, Y., Li, D., Wang, D., Ignatius, J., Cheng, T., and Wang, S. (2023). A branch-and-price-and-cut algorithm for the truck-based drone delivery routing problem with time windows. *European Journal of Operational Research*, **309**(3), 1125–1144.

Zhen, L., Gao, J., Tan, Z., Wang, S., and Baldacci, R. (2023). Branch-price-and-cut for trucks and drones cooperative delivery. *IISE Transactions*, **55**(3), 271–287.

Zhou, H., Qin, H., Cheng, C., and Rousseau, L.-M. (2023). An exact algorithm for the two-echelon vehicle routing problem with drones. *Transportation Research Part B: Methodological*, **168**, 124–150.

Zhou, L., Baldacci, R., Vigo, D., and Wang, X. (2018). A multi-depot two-echelon vehicle routing problem with delivery options arising in the last mile distribution. *European Journal of Operational Research*, **265**(2), 765–778.