Branch-and-price approaches for real-time vehicle routing with picking, loading, and soft time windows

Martin Wölk and Stephan Meisel

Münster School of Business and Economics,
University of Münster, 48149 Münster, Germany
martin.woelck@uni-muenster.de, stephan.meisel@uni-muenster.de

Technical Report (14th of December 2020)
(first version published on 10th of June 2020)

Abstract

We propose and evaluate branch-and-price approaches for vehicle routing problems with picking, loading, and soft time windows. This general type of vehicle routing problem is of particular relevance in the same-day delivery context, where fast routing algorithms are required due to the commitment to real-time delivery in the presence of high customer order frequencies. To boost the performance of branch-and-price we introduce the new method of tree compatible labeling with non-dominance-trees. This method represents cost functions by a fixed number of breakpoints and uses a specialized tree-based data structure to store Pareto-optimal labels. We prove the theoretical soundness of the new method and evaluate its performance numerically with respect to pricing, column generation, and branch-and-price. Our numerical results show that the method yields substantial performance gains. In particular, we show that with the new method, branch-and-price is able to reliably generate within a few minutes close to optimal solutions for problem instances with fifty customers. Our approach is the first branch-and-price approach for vehicle routing with picking, loading, and soft time windows. As such it represents an exact routing algorithm that is able to reliably satisfy the runtime requirements of real-time delivery services.

Keywords — branch-and-price, vehicle routing, same-day delivery

1 Introduction

Many businesses in the retail industry and in other industries offer same-day delivery services for distributing products to their customers. With such a service, customers place orders typically online and are then served within the same calendar day. In many cases, businesses, such as supermarkets and grocery shops, are even committing themselves to providing service within a few hours after a customer order has been placed. The resulting type of same-day delivery service mostly occurs in urban areas, and is often referred to as rush service, instant delivery service, or real-time delivery service. The fact that consumers want faster and more frequent deliveries (Jacobs et al. 2019) has led to a significantly increasing number of real-time delivery service providers. Examples range from long-established companies such as Amazon to younger players such as the rapidly expanding German real-time delivery company flaschenpost. A key challenge for all real-time delivery service providers is to minimize the costs of delivery operations while satisfying the real-time service commitment. This challenge raises the need for high-quality routing algorithms that are able to cope with the runtime requirements of real-time delivery services.

In this work, we propose and compare branch-and-price (BaP) approaches for solving the vehicle routing problems that typically occur with real-time delivery services. Our problem formulation is new and takes into account the facts (1) that customer orders become known only very shortly before the routing algorithm is executed, and (2) that violations of the delivery time service commitment cannot be fully avoided, given that businesses have both limited service vehicle fleet sizes and uncertainty about
customer orders. As a consequence of (1) our problem features both picking times (sometimes also called release dates) and loading times of customer orders. As a consequence of (2) our problem features soft time windows. We note that real-time delivery typically implies that the time window of an order starts when the order is placed, and that therefore only the upper bound of the time window can be violated.

To our knowledge this is the first work to propose a BaP approach for vehicle routing problems with picking, loading, and soft time windows (VRPSTW-PL). Our model considers that both picking times and loading times depend on the individual customer orders, and that loading requires the vehicles to be present at designated loading areas within the depot. Each order has an individual picking duration, denoting, e.g., the amount of time still needed to transfer the order from a warehouse to the loading area. Once all of a route's orders are picked, they can be loaded into a vehicle, where the order loading durations are proportional to the order volume. Moreover, our model considers ready times of vehicles (that may be on their way back to the depot when the optimization run starts) and ready times of loading areas (that may be occupied when the optimization run starts). Ready times are particularly relevant for real-time delivery services with high order frequency, where the combination of continuously incoming orders and commitment to fast delivery requires that a sequence of VRPSTW-PL instances must be solved at short time intervals throughout the business day, and where thus feasible solutions may only be guaranteed if vehicles and loading areas that become available in the near future are also considered in the current optimization run.

In this work, we propose BaP approaches for the VRPSTW-PL. In particular, we introduce the new method of tree compatible labeling with non-dominance trees (TCL-ND) to accelerate the dynamic programming algorithm in the pricing step of BaP. The presence of soft time windows, loading times, and picking times requires that cost functions (instead of scalar costs) must be compared in the dynamic programming algorithm. Tree compatible labeling (TCL) represents these cost functions with a fixed number of points, and enables the use of non-dominance trees (ND-trees) as data structures to store Pareto-optimal labels. ND-trees were originally proposed by Jaszkiewicz and Lust (2018) for multi-objective optimization with evolutionary algorithms. By using TCL-ND the label-space in the pricing step of BaP is partitioned into (ideally non-overlapping) hyperrectangles that are stored in a tree structure. We prove the theoretical soundness of TCL, and we show numerically that it leads to significant performance improvements when combined with ND-trees in the TCL-ND method. All numerical experiments are carried out for the VRPSTW-PL as well as for the special case of vehicle routing problems with soft time windows (VRPSTWs). Our main contributions are:

- We introduce the VRPSTW-PL and illustrate its relevance for real-time delivery services.
- We propose the first BaP approaches for the VRPSTW-PL.
- We evaluate these BaP approaches numerically with different dynamic programming algorithms, pricing heuristics, relaxations, and stabilization techniques.
- We introduce the new method TCL-ND and prove its theoretical soundness.
- We show that BaP with TCL-ND outperforms the other approaches and is able to reliably satisfy the runtime requirements of real-time vehicle routing applications.

The remainder of this paper is structured as follows. Section 2 summarizes the literature about BaP approaches for vehicle routing problems (VRPs) related to ours. In Section 3 we introduce the VRPSTW-PL in a real-time delivery context. In Section 4 we present a formal model for the problem. In Section 5 we introduce TCL-ND, and propose dynamic programming algorithms for solving the VRPSTW-PL pricing problem. In Section 6 we provide the additional algorithmic elements we consider in our BaP approaches. In Section 7 we describe the setup of our numerical experiments including the considered problem instances. In Section 8 we present our numerical results. Section 9 concludes the paper.

2 Related Work

This section summarizes works on BaP approaches for VRPs related to ours. For a general review of branch-and-price-and-cut approaches for VRPs, we refer to Costa et al. (2019). For an extensive review of VRP variants and solution methods other than BaP, we refer to Lahyani et al. (2015). To our knowledge picking times for VRPs have first been introduced by Arda et al. (2014) and Cattaruzza et al. (2016) under the name release dates. We prefer, however, the term picking times, as orders are picked from a warehouse in the most relevant applications for our problem class. In Section 7.1 we provide an overview
of works on BaP for VRPs with picking times and loading times. All of these works consider VRPs with hard time windows. We note that, to our knowledge, no prior works exist on BaP for VRPs that combine picking times and loading times with soft time windows. This combination is present in the VRPSTW-PL and leads to a critical cost tradeoff, as adding a customer to the end of a path implies that the vehicle has to wait at the depot (due to the durations of additional picking and loading operations), which may then cause expensive time window violations at other customers of the path. In Section 2.2 we provide an overview of BaP approaches for VRPs with soft time windows (without picking and loading).

2.1 Branch-and-Price for VRPs with Picking and Loading

[Azi et al. (2010)] propose a BaP approach for a multi-trip VRP with hard time windows and loading. The pricing problem is an elementary shortest path problem with resource constraints (ESPPRC), that the authors solve with bounded bidirectional dynamic programming. The dynamic programming algorithm operates on a graph, in which nodes represent feasible routes. The requirement that all feasible routes must be enumerated to generate the graph is mitigated by a maximum route duration constraint that drastically reduces the number of feasible routes. The fact that the problem has hard time windows allows the authors to use a dynamic programming algorithm with scalar costs instead of with cost functions (which are required for the VRPSTW-PL).

[Hernandez et al. (2014)] propose a BaP approach for a VRP that is quite similar to the problem considered by [Azi et al. (2010)]. The problem also includes a maximum route duration constraint allowing for a-priori enumeration of all routes that satisfy both capacity constraints and hard time window constraints. The pricing problem schedules routes at different starting times, and the generated columns are tuples of route and starting time. The master problem assigns zero or more of these tuples to the limited set of homogeneous vehicles. [Hernandez et al. (2016)] extend the work of [Hernandez et al. (2014)]. The authors present two alternative problem models, one where columns represent a sequence of routes, and one where columns represent tuples of a single route and its starting time. The first model is solved by a BaP algorithm in which the pricing algorithm creates a sequence of routes. The second model discretizes the time horizon into a set of non-overlapping intervals. This model includes mutual exclusion constraints in the master problem to ensure that a bound on the number of vehicles that may start loading in an interval is not violated. As a consequence, the pricing problem is adapted to include the newly introduced dual costs. Moreover, the number of pricing problems to be solved is reduced by aggregation of labels. The authors highlight that loading times lead to a significant runtime increase when solving the pricing problems with forward dynamic programming, and therefore consider mono-directional backward dynamic programming instead. Due to the presence of hard time windows, both [Hernandez et al. (2014)] and [Hernandez et al. (2016)] are able to work with scalar costs of routes in their dynamic programming algorithms.

[Paradiso et al. (2020)] propose an exact optimization algorithm for multi-trip VRPs. The authors develop models with an exponential number of both columns and rows, by introducing the concept of structures and structure feasibility constraints. A structure is a feasible sequence of customers that can be visited by a vehicle as part of its multi-trip operation. A trip is a structure with a departure time. The authors develop a structure enumeration algorithm that is integrated into BaP and show how this algorithm can be applied to several multi-trip VRPs with hard time windows, including a VRP with picking times and a VRP with loading times. As their dynamic programming algorithm exploits the fact that the time windows are hard, no tradeoff between loading, picking and time window violation costs exists. The authors are therefore able to work with scalar costs (which is not possible for the VRPSTW-PL).

[Ceselli et al. (2009)] do not provide a full BaP approach, but propose a column generation algorithm for a VRP problem with a heterogeneous fleet, multiple depots, hard time windows, and loading times. The authors' definition of loading times differs from the related work shown above. Loading times are approximated by a constant for each vehicle type. These constant loading times and the hard time windows allow the authors to model the pricing problem as an ESPPRC that can be solved with the bounded bidirectional dynamic programming algorithm of [Righini and Salani (2006)]. Note that the assumption of constant loading times per vehicle differs from the loading time properties of our problem, where loading times are functions of the volumes of the orders to be loaded into a vehicle.

2.2 Branch-and-Price for VRPs with Soft Time Windows

[Qureshi et al. (2009)] propose a BaP approach for a VRPSTW, where the lower bounds of time windows are hard, and the upper bounds of time windows are soft. Linear penalties occur if a soft bound is
Table 1: BaP approaches for vehicle routing with loading, picking, and soft time windows.

<table>
<thead>
<tr>
<th></th>
<th>loading</th>
<th>picking</th>
<th>TCL</th>
<th>pricing algorithm</th>
<th>pricing</th>
<th>pricing</th>
<th>stabilized</th>
<th>column gen.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FW-</td>
<td>BW-</td>
<td>Bi-</td>
<td>DP</td>
<td>DP</td>
</tr>
<tr>
<td>Azi et al. (2010)</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hernandez et al. (2010)</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hernandez et al. (2016)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Caselli et al. (2009)</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qureshi et al. (2009)</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AbbaJallah and Jang (2014)</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>this work</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Symbol ‘x’ indicates that a feature is evaluated in the respective work. Columns ‘FW-DP’, ‘BW-DP’, ‘Bi-DP’ indicate if forward dynamic programming, backward dynamic programming, or bidirectional dynamic is considered for solving the pricing problem. Column ‘TCL’ indicates if tree compatible labeling is considered.

violated, whereas, in case of early arrival at a customer location, vehicles must wait at the location until the customer’s time window begins. Waiting at a customer is not permitted in case of arrival within the time window. These problem features allow the authors to work with scalar costs instead of with cost functions when solving the pricing problem with forward dynamic programming. The authors’ approach cannot be transferred to the VRPSTW-PL, as in this case extending a label leads to additional loading times and picking times, which delays the arrival at all customers along the complete path, and may thus increase the penalties for violation of soft upper bounds of time windows. Similarly, AbbaJallah and Jang (2014) consider a VRPSTW with a special type of time windows that may be violated by a predefined amount of time at predefined, fixed penalty costs. If a vehicle arrives at a customer location later than the lower bound of the customer’s outer time window, service may either start immediately or be postponed until the lower bound of the customer’s inner time window is reached. In the former case a fixed penalty is incurred, whereas in the latter case the vehicle has to wait without incurring penalty costs. The pricing problem within the proposed BaP approach is a slightly modified ESPPRC, that the authors solve by forward dynamic programming. Due to the special type of time windows considered, the authors’ dynamic programming algorithm is able to generate a relatively low number of labels, and can therefore work with scalar costs.

Tab et al. (2014) propose a BaP approach for a VRPSTW with gamma distributed travel times. Vehicles may arrive early at a customer location, but must start service immediately at arrival while incurring penalty costs for being early. The authors show that the ESPPRC dominance criteria can be adjusted by modifying labels such that they arrive at the same point. These fixed arrival times allow for dynamic programming with scalar costs. The proposed BaP approach makes use of column pools and additionally applies decremental state space relaxation (DSSR), which was independently introduced by Boland et al. (2006) and by Righini and Salani (2008). The BaP approach that Liberaore et al. (2011) propose for a VRPSTW (without picking and loading) is closest to our work due to the authors’ use of cost functions for solving the pricing problem. Vehicles that arrive early at a customer location may wait until the customer’s time window opens, or otherwise incur penalty costs for being early. As waiting delays the arrival time at succeeding customers, a waiting operation may cause additional costs or may reduce the overall costs of a route. As a result, the cost term considered by the pricing algorithm is a function of time. (Cost functions were initially studied by Koch et al. (1998) for a shortest path problem with hard time windows.) Liberaore et al. (2011) show that the cost function for a single path is convex and piecewise linear, and that bidirectional dynamic programming as well as DSSR can be adjusted to the cost function. Bettinelli et al. (2014) transfer the BaP approach of Liberaore et al. (2011) to a multi-depot pickup and delivery problem with a heterogeneous fleet of vehicles and with soft time windows. The authors solve their pricing problem by extending the dynamic programming algorithm of Liberaore et al. (2011) and by applying pricing heuristics that are based on a restricted edge set. Neither loading times nor picking times are considered.
2.3 Summary

Table 1 presents an overview of the works discussed in Sections 2.1 and 2.2 and contrasts these works with our work. The table highlights that our work is the first to provide a BaP approach for a vehicle routing problem with picking, loading, and soft time windows, and that our work is the first to propose BaP approaches with tree compatible labeling.

The six rightmost columns of the table characterize the column generation parts of the works from the literature with respect to the four general features “pricing algorithm”, “pricing heuristic”, “pricing relaxation”, and “stabilized column generation”. We observe that (1) both forward dynamic programming and bidirectional programming have been proposed several times as basic algorithms for solving the pricing problem, (2) most of the works use a heuristic algorithm to generate solutions of the pricing problem, (3) most of the works do not relax the elementary path constraint during the pricing algorithm, and (4) a few authors use a technique to stabilize column generation. Against the background of this heterogeneity of the approaches from the literature, we provide in this work an evaluation of a number of different BaP approaches for the VRPSTW-PL. In particular, we compare the performances of forward dynamic programming and bidirectional dynamic programming, we propose and evaluate an approach for solving the pricing problem heuristically, we consider relaxing the pricing problem, and we evaluate the benefit of stabilizing column generation.

Before presenting the algorithmic components of our BaP approaches in Sections 5 and 6, we describe the VRPSTW-PL in Section 3 and formulate the problem mathematically in Section 4.

3 Real-Time Vehicle Routing with Picking, Loading, and Soft Time Windows

We illustrate the VRPSTW-PL without loss of generality in the particularly relevant context of a real-time delivery service provider that continuously receives new customer orders. In order to be able to comply with the real-time service commitment, the runtime $\Delta$ that such a service provider is willing to allocate at most for solving a VRPSTW-PL instance must consume only a small fraction of the total processing time of a customer order. Hence, real-time service providers may set $\Delta$ to merely a few minutes, and solve a new problem instance every $\Delta$ minutes throughout the business day. Note that our focus is on solving a single VRPSTW-PL instance, and that we do not address the related, but separate issue of modeling the service provider’s daily operations as a dynamic decision problem (e.g., Meisel 2011).

Figure 1a presents an illustrative example of the real-time service provider’s problem at the point in time at which the optimization run starts. The example features a total of eight customer locations (indicated by dots) that must be served. The set of customers consists of customers whose orders have already been fully picked (gray dots), and of customers whose orders are still in the process of being picked (white dots). For each customer $i$ we define the upper bound of the delivery time window of $i$ as the time $t_i$ remaining (from the start of the optimization run on) without a violation of the service commitment of $i$.

The state of the picking processes is illustrated at the bottom of the depot in the center of Figure 1a. Gray rectangles illustrate the fully picked orders of the five customers represented by gray dots, and the larger white rectangles illustrate the three customer orders with ongoing picking process. We assume that the remaining picking time $t_{pi}^i$ (remaining duration of the picking process from the start of the optimization run on) of customer order $i$ is known with sufficient accuracy, and that an order may be loaded into a vehicle only if it has been fully picked. We also assume that the loading time $t_{li}^i$ (duration of the loading process) of a fully picked order $i$ is known with sufficient accuracy (and may in practice depend on attributes such as order volume). The total duration of the loading process of a vehicle equals the sum of the loading times of the orders assigned to the vehicle. Our example depot has three loading areas (I, II, III), one of which (I) is currently occupied by the loading process of service vehicle ‘a’. For a loading area that is occupied at the start of the optimization run, we define the loading area ready time as the time remaining from the start of the optimization run on until the area is free and available for use again. Loading area ready times are denoted as $t^I, t^{II}, t^{III}$ in the example of Figure 1.

A homogeneous set of service vehicles with given capacity is available for the current optimization run. In the example of Figure 1a vehicles ‘b’, ‘c’, ‘d’, ‘e’ are available, whereas vehicle ‘a’ has an ongoing loading process and is therefore not available for the current optimization run. Each of the available vehicles has a vehicle ready time indicating how much time remains (from the start of the optimization run on) until the vehicle is present at the depot. A vehicle’s ready time may be zero (e.g., vehicle ‘b’), or larger than zero (representing the case where a vehicle currently is on its way back to the depot). Note
Figure 1: Example of a VRPSTW-PL instance and its solution. We assume that \( \Delta = 2 \), i.e., that time advances by 2 time units during the optimization run.

![Diagram](image)

(a) At the start of the optimization run.

(b) At the end of the optimization run.

Table 2: Non-empty \( \Delta \)-slots of the VRPSTW-PL instance of Figure 1a. (\( \Delta = 2 \), vehicle loading time approximated by 10 time units)

<table>
<thead>
<tr>
<th>( \Delta )-slot</th>
<th>vehicle</th>
<th>area</th>
<th>vehicle ready</th>
<th>area ready</th>
<th>vehicle-area ready</th>
<th>( \Delta )-slot ready</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>II</td>
<td>0</td>
<td>0</td>
<td>max{0,0} = 0</td>
<td>max{0,1} = 1</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>III</td>
<td>1</td>
<td>0</td>
<td>max{1,0} = 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>d</td>
<td>I</td>
<td>4</td>
<td>5</td>
<td>max{4,5} = 5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>e</td>
<td>II</td>
<td>8 + 10 = 10</td>
<td>max{8,10} = 10</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

that the latter case may be of particular relevance for ensuring feasible solutions in real-time delivery services with high order frequency and limited fleet size. We assume that ready times of vehicles can be estimated with sufficient accuracy. In Figure 1a the vehicle ready times are indicated next to the available vehicles.

Figure 1b illustrates the solution of our example VRPSTW-PL instance at the moment where the runtime \( \Delta = 2 \) of the optimization has elapsed. All orders have been assigned to a route, and two of the three routes contain both orders with finished picking process and orders with ongoing picking process. The picking times of all customers, as well as the ready times of vehicles and loading areas have been reduced by \( \Delta = 2 \) time units. For each route, the maximum of the reduced picking times and the vehicle’s ready time determines the ready time of the route, which is the time that elapses from the end of the optimization run on until the vehicle’s loading process can start. In Figure 1b the ready times of routes are indicated next to the ready times of the available vehicles.

Recall that \( \Delta \) is short, and that ready times of vehicles, loading areas, and routes may be larger than \( \Delta \). As a consequence, a real-time service provider may choose to implement only the routes with ready times that are less than or equal to \( \Delta \) and to reconsider the orders of all other routes together with new orders as parts of a subsequent VRPSTW-PL instance. Following this logic, we think of time as a sequence of intervals of length \( \Delta \), and refer to each such interval as a \( \Delta \)-slot. Before an optimization run starts, we can now assign vehicles and loading areas according to their ready times to \( \Delta \)-slots. Based on this assignment, we define the capacity of a \( \Delta \)-slot as the number of vehicles in the \( \Delta \)-slot that are available at a loading area in the same slot.

Table 2 shows the three \( \Delta \)-slots with nonzero capacity of the VRPSTW-PL instance presented in Figure 1a. Slot 0 contains vehicles ‘b’ and ‘c’ (capacity 2), slot 1 contains vehicle ‘d’ (capacity 1), and slot 2 contains vehicle ‘e’ (capacity 1). The table shows that \( \Delta \)-slot 0 contains all vehicles that are ready and available at a loading area within interval \([0,2]\), that \( \Delta \)-slot 1 contains all vehicles that are ready and available at a loading area within interval \((4,6]\), and that \( \Delta \)-slot 2 contains all vehicles that are ready and
available at a loading area within interval \((8, 10]\). We can now use the groupings of vehicles and loading areas in the \(\Delta\)-slots to represent loading areas implicitly in terms of \(\Delta\)-slot ready times. The rightmost column of Table 2 indicates for each \(\Delta\)-slot the ready time (by which all vehicles in the slot are ready and available at a loading area). The example also includes the case where a loading area (II) reappears in a later \(\Delta\)-slot, i.e., where a loading area is used by two vehicles subsequently. In this special case, we must approximate the vehicle loading time (before the optimization run starts) for the first time that area II is used in order to calculate the second ready time of area II. In our example we approximate the vehicle loading time by 10. In practice such an approximation could be derived from historical data. The following Section 4 shows that considering ready times of \(\Delta\)-slots in the VRPSTW-PL formulation facilitates the pricing step within our BaP approaches.

4 Problem Formulation

In Section 4.1 we formulate a compact VRPSTW-PL model, and in Section 4.2 we provide both the master problem and the pricing problem resulting from dual decomposition.

4.1 Compact Model

The VRPSTW-PL is defined on a directed graph \(G = (\mathcal{N}, \mathcal{E})\) with nodes \(\mathcal{N} = \{0, \ldots, N - 1\}\) and with edges \(\mathcal{E} \subseteq \{(i, j) : i, j \in \mathcal{N}\}\). Node 0 represents the depot, and the set of customers is given as \(\mathcal{C} = \mathcal{N} \setminus \{0\}\). We denote the homogeneous set of vehicles as \(\mathcal{V}\), and the vehicle capacity as \(Q\). \(\mathcal{L} = \{0, \ldots, L - 1\}\) represents the set of non-empty \(\Delta\)-slots, where each \(\Delta\)-slot \(l \in \mathcal{L}\) has a ready time \(t_l^\Delta\) and a capacity of \(|\mathcal{Y}_l|\) vehicles, with \(\mathcal{Y} = \bigcup_{l \in \mathcal{L}} \mathcal{Y}_l\), and with \(\forall l, l' \in \mathcal{L} : l \neq l' \implies \mathcal{Y}_l \cap \mathcal{Y}_{l'} = \emptyset\).

We assume that the driving times \(t^d = (t_{ij}^d)_{(i, j) \in \mathcal{E}} \in \mathbb{N}[\mathcal{E}]\) along the edges of \(G\) are deterministic and fulfill the triangle inequality. Moreover, each \(i \in \mathcal{N}\) has a non-negative demand \(q_i\), a non-negative service time \(t_i^s\), a non-negative loading time \(t_l^o\), and a non-negative picking time \(t_l^p\), where we assume that \(q_0 = t_0^s = t_0^o = t_0^p = 0\). Recall that the picking time \(t_l^p\) denotes the time remaining (from the start of the optimization run on) until the order of \(i\) is fully picked, and that the loading process of the order may start only after the order is fully picked. The time \(t_i\) for customer \(i\) is modeled as a soft upper bound of the time window of \(i\), where \(t_0 = \infty\), and where \(t_i < 0\) is allowed to represent the case that the time window of \(i\) has been violated already before the optimization run starts.

Let \(x_{v(ij)} = 1\) if vehicle \(v \in \mathcal{Y}_l\) of \(\Delta\)-slot \(l\) drives along edge \((i, j)\), and let \(x_{v(i)} = 0\) otherwise. Further, let \(y_{v(i)} = 1\) if \(v \in Y_l\) visits node \(i\), and let \(y_{v(i)} = 0\) otherwise. We refer to the number of time steps by which the upper bound of the time window of \(i\) is violated as the tardiness at \(i\), and we represent the tardiness of vehicle \(v \in \mathcal{Y}_l\) at \(i\) by variable \(z_{v(i)}\). We define that \(z_{v(i)} = 0\) also represents the case where \(v\) does not visit \(i\), and that \(\forall l \in \mathcal{L}, v \in \mathcal{Y}_l : z_{v(i)} = 0\). Parameter \(\kappa_i \in \mathbb{R}_+\) denotes the penalty per time unit of tardiness at customers \(i\). Further, we introduce parameters \(b_l^p, b_l^s \in \mathbb{N}_+\), with \(0 \leq b_l^p \leq |\mathcal{Y}_l| \leq b_l^s\), to be able to bound the capacities of \(\Delta\)-slots during BaP (cf. Section 6.6).

Problem 1 describes the VRPSTW-PL as a minimization over the coupling constraints (1)-(2) and the non-coupling constraints \(\mathcal{R}_l\) given by Eqs. (3)-(10). We separate the set of coupling constraints from the set of non-coupling constraints in our problem formulation to make the structure of the pricing problems more apparent.

**Problem 1 (VRPSTW-PL).**

\[
\begin{align*}
\min & \quad \sum_{l \in \mathcal{L}} \sum_{v=1}^{b_l^p} \left( \sum_{(i,j) \in \mathcal{E}} (t_{ij}^d + t_l^o) x_{v(ij)} + \sum_{i \in \mathcal{E}} \kappa_i z_{v(i)} + l \right) \\
\text{subject to} & \quad \sum_{l \in \mathcal{L}} \sum_{v=1}^{b_l^p} y_{v(i)} \geq 1 & \quad \forall i \in \mathcal{C}, \\
& \quad \sum_{v=1}^{b_l^p} x_{v(ij)} \geq b_l^s & \quad \forall l \in \mathcal{L}, \\
& \quad x_{v(i)}, y_{v(i)}, z_{v(i)} \in \mathcal{R}_l & \quad \forall l \in \mathcal{L}, v \in \mathcal{Y}_l.
\end{align*}
\]

Problem 1 has the objective to minimize the overall sum of driving times and tardiness penalties. We add \(l\) to the objective to prioritize \(\Delta\)-slots with smaller ready times. Constraint (1) ensures that...
each customer is visited at least once, and Constraint (3) ensures that at least \( b_i^- \) vehicles are used in \( \Delta \)-slot \( l \). We start with \( b_i^- = 0 \) for each \( l \), and maintain the option to increase the parameter values when branching (cf. Section 6.4). The set of non-coupling constraints for every vehicle is then given by

\[
\mathcal{J}_i = \left\{ (x_i, y_i, z_i) : \begin{array}{l}
\sum_{i \in \mathcal{V}} q_i y_i \leq Q, \\
\sum_{(k,i) \in \mathcal{E}^{-}} x_{k,i} = \sum_{(i,j) \in \mathcal{E}^{+}} x_{i,j} = y_i \\
i \in \mathcal{V}, \\
t_l^a x_{k,i} \leq a_i^a, \\
t_l^b y_i \leq a_i^b, \\
a_i^a + \sum_{i' \in \mathcal{V}} t_{i,i'} y_{i'} \leq \sum_{(0,j) \in \mathcal{E}} a_{10j} \\
i \in \mathcal{V}, \\
\sum_{(k,i) \in \mathcal{E}^{-}} (a_{k,i}^v + x_{k,i}(t_k^v + t_{k,i}^d)) \leq \sum_{(i,j) \in \mathcal{E}^{+}} a_{i,j}^\nu \\
i \in \mathcal{V}, \\
v_i + z_i \geq a_{i,j}^\nu - (1 - y_i)M \\
M y_i \geq a_{i,j}^\nu \\
x_i \in \{0,1\}^{n_{\mathcal{V}}}, y_i \in \{0,1\}^{n_{\mathcal{V}}}, z_i \in \mathbb{N}_{+}^{n_{\mathcal{E}}} , a_i^a \in \mathbb{N}_{+}^{n_{\mathcal{E}}} \}.
\]

Constraints (3) and (4) are the vehicle capacity and flow constraints. Constraints (5) and (6) ensure that auxiliary variable \( a_i^a \) is greater than or equal to the maximum of the ready time of \( \Delta \)-slot \( l \) and the maximum picking time of the customer orders assigned to the vehicle. We let the auxiliary variable \( a_{i,j}^v \) denote the arrival time of the vehicle at \( i \), if \((i,j)\) is traversed, and we let \( a_{i,j}^v = 0 \) otherwise. Constraints (7) and (8) eliminate subtours by associating arrival times to edges. Finally, Constraints (9) and (10) map arrival times to tardiness. In the following Section 4.2 we decompose Problem 1 to be able to solve our VRPSTW-PL with BaP.

### 4.2 Dual Decomposition

In this section, we formulate the master problem and the pricing problem that results when the master problem is solved with column generation. Problem 2 shows the master problem over an exponential number of feasible (w.r.t. \( \mathcal{J}_i \)) routes \( r \in \Omega_l \). To simplify the notation, we limit \( \Omega_l \) to non-empty routes, i.e., to routes that visit at least one customer. Each route \( r \) is associated with a binary variable \( x_r \), which is equal to 1 if \( r \) is assigned to a vehicle and 0 otherwise. We denote the cost of \( r \) with \( c_r = \sum_{(i,j) \in \mathcal{E}} t_{i,j}^d x_{i,j} + \sum_{i \in \mathcal{V}} \kappa_i z_{ri} + l \), where \( x_{ri} = 1 \) if \( r \) includes \((i,j)\), and where \( z_{ri} \) is the tardiness of \( r \) at \( i \). The binary coefficient \( y_{ri} \) is equal to 1 if \( r \) contains \( i \) and 0 otherwise.

**Problem 2 (Master Problem).**

\[
\begin{align*}
\min & \quad \sum_{l \in \mathcal{L}} \sum_{r \in \Omega_l} c_r x_r \\
\text{subject to} & \quad \sum_{l \in \mathcal{L}} \sum_{r \in \Omega_l} y_{ri} x_r \geq 1 & \forall i \in \mathcal{E}, \\
& \quad \sum_{r \in \Omega_l} x_r \geq b_l^- & \forall l \in \mathcal{L}, \\
& \quad \sum_{r \in \Omega_l} x_r \leq b_l^+ & \forall l \in \mathcal{L}, \\
& \quad x_r \in \{0,1\} & \forall l \in \mathcal{L}, r \in \Omega_l.
\end{align*}
\]

The linear programming relaxation of the master problem is solved by column generation due to the exponential number of variables. Let \( \lambda_r, \lambda^-, \sigma \) be the non-negative dual vectors of constraints (11)-(13), respectively. Problem 3 shows the pricing problem for \( \Delta \)-slot \( l \).
Problem 3 (Pricing Problem).

\[ \gamma_l = \min \sum_{i,j \in \mathcal{E}} (t_{ij}^0 + t_{ij}^1)x_{ij} + \sum_{i \in \mathcal{G}} \kappa_i z_i + l - \sum_{i \in \mathcal{G}} \lambda_i^+ y_i - \sum_{(0,j) \in \mathcal{E}} \lambda_j^- x_{0j} \]

subject to \((x_l, y_l, z_l) \in \mathcal{X}_l\)

A non-empty solution \((x_r, y_r, z_r)\) of Problem 3 represents a column, that may enter the basis of the restricted master problem if its reduced costs

\[ c_r x_r - \sum_{i \in \mathcal{G}} \lambda_i^+ y_i - \lambda^+_l - \sigma_l \]

are negative. Note that one can compute a lower bound on the optimal solution of Problem 3 at every iteration of the column generation algorithm by

\[ \sum_{l \in \mathcal{L}} \sum_{r \in \mathcal{L}_l} c_r x_r + \sum_{l \in \mathcal{L}} b^+_l (\gamma_l - \sigma_l), \]

where \(\Omega'_l\) is the restricted column set at the respective iteration. Our pricing problem is an ESPPPRC. To solve this problem, we propose in Section 5 dynamic programming algorithms with the new method TCL-ND.

5 Pricing with Tree Compatible Labeling and ND-Trees

In Section 5.1 we provide a forward dynamic programming algorithm for solving Problem 3 and prove that the algorithm solves the problem to optimality. In Section 5.2 we introduce TCL-ND to accelerate the algorithm, and prove the correctness of the method.

5.1 Forward Dynamic Programming

In Section 5.1.1 we define the labels used by our forward dynamic programming algorithm. In Section 5.1.2 we detail the cost function that is required as an element of the labels. In Section 5.1.3 we provide sufficient criteria for checking the dominance relation between labels. In Section 5.1.4 we outline the forward dynamic programming algorithm.

5.1.1 Label Definition

Our forward dynamic programming approach is based on the algorithm that Pei et al. (2004) propose for the ESPPPRC of a VRP with hard time windows. In order to be able to solve the ESPPPRC of the VRPSTW-PL, we define the notions of path and label as follows.

A path \(p = (i_1, i_2, \ldots, i_{|p|})\) consists of a sequence of nodes \(i_n\), with \(i_1 = 0\) (the depot), and with \(i_n \in \mathcal{G}\) for \(n > 1\). We quantify the resource consumption of a path \(p\) by the following measures: functions \(R^t(p) = \sum_{n=1, \ldots, |p|-1} t^d_{i_n i_{n+1}}\), \(R^s(p) = \sum_{n=1, \ldots, |p|-1} t^d_{i_n i_{n+1}}\), and \(R^c(p) = \sum_{n=1, \ldots, |p|-1} g_{i_n}\) compute the total travel time, the total service time, and the total vehicle capacity used by \(p\). Let \(R^g(p) = \max \{0, \{t^g_{i_n} - t^d_{i_n} : n \in \{1, \ldots, |p|\}\}\}\) be the amount of time the vehicle has to wait at the depot due to ongoing picking processes, and let \(R^c(p) = \sum_{n=1, \ldots, |p|-1} t^c_{i_n}\) be the amount of time that the vehicle has to wait at the depot due to the loading process. The vector \(R^u(p) = (R^u_{i_n}(p))_{i_n < \mathcal{G}} \in \mathbb{N}_+^{\mathcal{G}}\) indicates how often each customer appears in \(p\), and the function \(R^N(p) = \sum_{i \in \mathcal{G}} R^u_{i}(p)\) aggregates \(R^u(p)\). \(R^N(p)\) is used to quickly check if a path visits less customers than another path (before \(R^u(p)\) is checked), and to thereby speed up the dominance checks (cf. Section 5.1.3). Finally, \(R^u(p) = t^d_{i_n} + R^s(p) + R^c(p)\) defines the earliest possible arrival time at the path’s terminal node \(i_{|p|}\) for the case that the waiting times due to picking and loading are ignored. In the following Section 5.1.2 we additionally define the costs associated with a path \(p\) in terms of a cost function \(R^f(p, \cdot)\). We refer to the combination of a path, its resources, and its cost function as a label \(L_i = (p, R^N(p), R^u_{i}(p), R^d(p), R^c(p), R^f(p, \cdot))\) for node \(i\), where \(i\) represents the terminal node \(i_{|p|}\) of \(p\). Note that any given label can only be associated with one single path, and that any node \(i\) may be associated with a set of labels \(\{L_{i}^1, L_{i}^2, \ldots\}\). If the terminal node of \(p\) is the depot \((i = 0)\), we refer to path \(p\) as a route, and if a route contains each customer at most once, we refer to it as an elementary route.
Figure 2: The paths $p^1$ (bold arrows) and $p^2$ (regular arrows) both start at the depot 0 and end at customer 3. Numbers next to arrows indicate travel times. We assume that $t_i^e = 0$ for all $i$, $t_i^b = t_i^e = \lambda_i^e = \lambda_i^b = 0$ for $i \in \{0, 1, 2, 3\}$, $\bar{t}_1 = \bar{t}_2 = 12$, $\bar{t}_0 = \bar{t}_3 = \bar{t}_4 = \infty$, and that $t_4^e + t_4^b > 0$.

Figure 3: Two alternative ways of formulating cost functions for the paths $p^1$ and $p^2$ from the example in Figure 2.

5.1.2 Cost Function

We define the cost function (that is an element of the labels) on two arguments $(p, \delta)$, where $p$ is the path of the label, and where we refer to $\delta$ as the delay of $p$. The delay of a path is the total amount of time that a vehicle has to wait at the depot until all of the path’s orders are picked and loaded into the vehicle. Extending a label, i.e., considering to add a customer to the end of the label’s path, always increases the delay $\delta$ of the path (assuming that loading times are greater than zero), and thereby always delays the arrival at every customer in the path. Note that delaying the arrival at a customer may lead to additional costs in terms of tardiness penalties, and that therefore the VRPSTW-PL features a cost tradeoff between adding a customer to a path and causing additional tardiness penalties. This cost tradeoff differs from the cost tradeoffs that occur in VRPSTWs, where waiting merely is an option to decrease penalties for arriving too early at a customer (before the customer’s time-window starts). This key difference between the two problem types also implies that the VRPSTW-PL requires special criteria for checking the dominance relation between two labels (cf. Section 5.1.3). For the VRPSTW-PL, the cost function is given as

$$R_c(p, \delta) = R^d(p) + R^e(p) + R^\epsilon(p, \delta) + l - \sum_{n=1, \ldots, |p|} \lambda_n^e - \lambda_n^b,$$

(14)

where $R^\epsilon(p, \delta)$ is the penalty cost caused by time window violations. With $p[1:n] := (i_1, \ldots, i_n)$, these costs can be written as

$$R^\epsilon(p, \delta) = \sum_{n=2}^{[|p|]} \max \{0, R^\epsilon(p[1:n]) + \delta - \bar{t}_n\} \kappa_n.$$

We illustrate the difference between $R^\epsilon(p, \delta)$ and the cost function formulation used in the VRPSTW literature by the example given in Figures 2 and 3. Figure 2 shows two example paths $p^1$, $p^2$, each of
which starts at the depot 0 and terminates at customer 3. For the sake of simplicity we assume that only customers 1 and 2 have a delivery time window, that \( t_{1}^{N} = 0 \), and that only for customer 4 the sum of picking time and loading time is nonzero, i.e., we assume that in the example \( \delta \) may become greater than zero only by adding customer 4 to a path. Figure 5 shows the costs of \( p_{1}^{1} \) and \( p_{2}^{1} \) expressed as a function of the arrival time at the terminal customer 3, which is the standard cost function formulation in the VRPSTW literature (cf., e.g., Liberatore et al. 2011). The figure shows that if the vehicle arrives at customer 3 via \( p_{1}^{1} \) at \( t = 17 \) the costs are 11, and if the vehicle arrives at customer 3 via \( p_{2}^{1} \) at \( t = 17 \) the costs are 15, i.e., at \( t = 17 \) \( p_{1}^{1} \) is cheaper than \( p_{2}^{1} \). Now assume that we consider extending both \( p_{1}^{1} \) and \( p_{2}^{1} \) to customer 4, and that \( t_{4}^{1} + t_{1}^{1} = 7 \). In order to visit customer 4 (and collect its negative dual price), the vehicle must delay its arrival at customers 1 to 3 by waiting \( \delta = 7 \) time units at the depot. Due to the delay \( \delta \) the vehicle arrives at the terminal customer 3 at \( t = 18 \) via \( p_{1}^{1} \) and at \( t = 22 \) via \( p_{2}^{1} \), i.e., given a delay, we need to compare the cost functions at different points, and it is not apparent if any of the paths is cheaper than the other for all possible values of \( \delta \). In contrast, Figure 5 shows the costs of the two paths as a function of \( \delta \). Here the two cost functions do not intersect, and it is therefore apparent that \( p_{1}^{1} \) is cheaper than \( p_{2}^{1} \) for all values of \( \delta \). (We refer to Figure 9 in the Appendix for an example in which the cost functions of two paths intersect.)

Note that the maximum possible delay \( \delta^{\max} \) is bounded due to the vehicles’ capacity restrictions. We can compute an upper bound on the maximum waiting time for picking, and an upper bound for the maximum waiting time for loading. While the former is computed by solving the knapsack problem

\[
T^{o} = \max \left\{ \sum_{i \in C} t_{i}^{o} x_{i} \left| \sum_{i \in C} q_{i} x_{i} \leq Q, x_{i} \in \{0,1\}, \forall i \in C \right. \right\}
\]

We thus have \( \delta^{\max} = \max \{ \max_{i \in C} t_{i}^{o} - t_{i}^{N}, 0 \} + T^{o} \). Bounding \( \delta \) helps us in approximating the cost function with a small number of breakpoints, as shown in Section 5.2. The cost function \( R^{o}(p, \delta) \) is piecewise linear and convex as it is the maximum over a set of linear functions. For a given path \( p \) the function has at most \( \min\{\delta^{\max} + 1, |p|\} \) breakpoints, i.e., points at which its gradient changes. The gradient changes whenever a customer’s time window is violated for the first time.

5.1.3 Dominance

The forward dynamic programming approach we provide in Section 5.1.4 guarantees an optimal solution to Problem 3 if it is equipped with valid criteria for checking the dominance relation between labels. The dominance checking step of the algorithm uses these criteria to identify the labels (and thus the paths) that can be excluded from further consideration in the process of constructing an optimal solution to Problem 3. We will in the following associate with every path and (piecewise linear) cost function a set of breakpoints at which the cost function’s gradient changes. Given two labels \( L_{1}^{1}, L_{2}^{1} \) with breakpoint sets \( B^{1}, B^{2} \), we may exclude \( L_{2}^{1} \) from further consideration, if each possible extension of \( p_{2}^{1} \) to a feasible elementary route also extends \( p_{1}^{1} \) to a feasible elementary route, and if the costs of each route resulting from extension of \( p_{1}^{1} \) do not exceed the costs of the route resulting from applying the same extension to \( p_{2}^{1} \). In this case, we say that \( L_{1}^{1} \) dominates \( L_{2}^{1} \) and that \( p_{1}^{1} \) dominates \( p_{2}^{1} \). The following Theorem 1 provides sufficient criteria for checking the dominance relation between two labels. Note that \( R^{N} \) is excluded from the sufficient dominance conditions as it is only used to accelerate the dominance checks in practice. Further note that our dominance criteria differ from the related work on VRPSTWs (e.g., Liberatore et al. 2011) as we have to include the earliest possible arrival time \( R^{o}(p) \) in the label.

**Theorem 1.** A label \( L_{1}^{1} = (p_{1}^{1}, R^{N}(p_{1}^{1}), R^{o}(p_{1}^{1}), R^{i}(p_{1}^{1}), R^{e}(p_{1}^{1}, \delta)) \) dominates another label \( L_{2}^{1} = (p_{2}^{1}, R^{N}(p_{2}^{1}), R^{o}(p_{2}^{1}), R^{i}(p_{2}^{1}), R^{e}(p_{2}^{1}, \delta)) \), if

\[
R^{o}(p_{1}^{1}) \leq R^{o}(p_{2}^{1}), \quad (15)
\]

\[
R^{i}(p_{1}^{1}) \leq R^{i}(p_{2}^{1}), \quad (16)
\]

\[
R^{e}(p_{1}^{1}) \leq R^{e}(p_{2}^{1}), \quad (17)
\]

\[
\forall \delta_{b} \in (B^{1} \cup B^{2}) : R^{e}(p_{1}^{1}, \delta_{b}) \leq R^{e}(p_{2}^{1}, \delta_{b}). \quad (18)
\]

Eqs. (15)-(17) state that \( L_{1}^{1} \) may only dominate \( L_{2}^{1} \) if it does not consume more of the vehicle’s capacity than \( L_{2}^{1} \), and if it visits a subset of customers of \( L_{2}^{1} \), and if its arrival time at \( i \) is not greater than the corresponding arrival time of \( L_{2}^{1} \). Eq. (18) states that \( L_{1}^{1} \) may dominate \( L_{2}^{1} \) if at each of the breakpoints in \( B^{1} \) and in \( B^{2} \), the costs of \( L_{1}^{1} \) are not greater than the costs of \( L_{2}^{1} \).
Proof. Proof of Theorem 1. We define a backward path \( \overrightarrow{p} = (i_1, i_2, \ldots, i_m) \) as a sequence of nodes with \( i_{m+1} = 0, i_1 \in \mathcal{N} \), and with \( i_k \in \mathcal{C} \) for each remaining \( k \). Both the cost function given by Eq. (14) and all (forward) path resource consumption functions, except \( R^a(p) \), apply also to backward paths. For any given \( \overrightarrow{p} \) we let \( R^a(\overrightarrow{p}) = 0 \). A forward path \( p = (0, i_1, \ldots, i_m) \) and a backward path \( \overrightarrow{p} = (i_m, \ldots, 0) \) can be joined to an elementary route \( \overrightarrow{p} \cup \overrightarrow{p} := (0, i_1, \ldots, i_m, 0) \), if \( R^f(p) + R^a(\overrightarrow{p}) \leq Q \), if \( \mathcal{C}(p) \cap \mathcal{C}(\overrightarrow{p}) = \{i\} \), and if \( \forall k \in \mathcal{N} \setminus \{i\} : R^f_k(p) + R^a_k(\overrightarrow{p}) \leq 1 \). We denote the set of backward paths that can be joined with a path \( p \) to form a feasible elementary route as \( \mathcal{F}(p) \). With this, it follows from our definition of dominance, that \( p^1 \) dominates \( p^2 \), if \( p^1 \) can be joined with each \( \overrightarrow{p} \in \mathcal{F}(p^2) \) such that the cost of the resulting route does not exceed the cost of the route resulting from joining \( p^2 \) with \( \overrightarrow{p} \).

It is evident that the criteria given by Eqs. (13) and (16) ensure that each backward path \( \overrightarrow{p} \in \mathcal{F}(p^2) \) can be joined with \( p^1 \). We prove by contradiction that the criteria given by Eqs. (17) and (18) ensure that the costs of the route resulting from joining \( p^1 \) with a backward path \( \overrightarrow{p} \in \mathcal{F}(p^2) \) do not exceed the costs of the route resulting from joining \( p^2 \) and \( \overrightarrow{p} \). Assume that both Eqs. (17) and (18) hold, and that a backward path \( \overrightarrow{p} \in \mathcal{F}(p^2) \) exists, for which

\[
R^c(p^1 \cup \overrightarrow{p}) > R^c(p^2 \cup \overrightarrow{p}), \tag{19}
\]

where the cost \( R^c(p \cup \overrightarrow{p}) \) of the elementary route \( p \cup \overrightarrow{p} \) is given as

\[
R^c(p \cup \overrightarrow{p}) := R^c(p, \max\{R^f(p), R^a(\overrightarrow{p})\}) + R^c(\overrightarrow{p}) + R^a(\overrightarrow{p}) - t^c_1,
\]

where \( \delta(p, \overrightarrow{p}) \) is the delay at the depot (caused by waiting due to picking and loading) of \( p \cup \overrightarrow{p} \), and where \( \overrightarrow{\delta}(p, \overrightarrow{p}) \) is the arrival time of path \( p \cup \overrightarrow{p} \) at \( i \). As Eq. (17) holds, and as \( p^1 \) visits a subset of the customers visited by \( p^2 \), we may conclude that

\[
\delta(p^1, \overrightarrow{p}) \leq \delta(p^2, \overrightarrow{p}), \tag{20}
\]

\[
\overrightarrow{\delta}(p^1, \overrightarrow{p}) \leq \overrightarrow{\delta}(p^2, \overrightarrow{p}). \tag{21}
\]

Eq. (20) states that the delay of \( p^1 \cup \overrightarrow{p} \) is less or equal to the delay of \( p^2 \cup \overrightarrow{p} \), and Eq. (21) states that the arrival time of \( p^1 \cup \overrightarrow{p} \) is less than or equal to the arrival time of \( p^2 \cup \overrightarrow{p} \). Since \( R^c(p, \overrightarrow{p}) \) is monotone with non-negative slopes, it follows from Eqs. (20) and (21) that

\[
R^c(p^2 \cup \overrightarrow{p}) \geq R^c(p^1 \cup \overrightarrow{p}), \tag{22}
\]

(it is always preferable to arrive as early as possible), and from Eqs. (19) and (22) that

\[
R^c(p^1, \delta(p^1, \overrightarrow{p})) + R^c(\overrightarrow{p}, \overrightarrow{\delta}(p^1, \overrightarrow{p})) > R^c(p^2, \delta(p^1, \overrightarrow{p})) + R^c(\overrightarrow{p}, \overrightarrow{\delta}(p^1, \overrightarrow{p}))
\]

\[
\implies R^c(p^1, \delta(p^1, \overrightarrow{p})) > R^c(p^2, \delta(p^1, \overrightarrow{p}))
\]

which contradicts Eq. (18) at \( \delta_0 = \delta(p^1, \overrightarrow{p}) \).

We note that the dominance check can be implemented by using statically sized arrays, i.e., by using arrays whose size is known at compile-time. It is then possible to poll the dominance checks into one Boolean function suitable for automatic detection of SIMD operations. A potential downside of the statically sized arrays is an increase in compiled code and in the chance of instruction cache misses. However, we observe in practice that the statically sized arrays are faster than, e.g., a dominance check in which we check \( R^a(p^1) \leq R^a(p^2) \) (by checking if \( p^1 \) visits a subset of \( p^2 \)'s customers using hash sets), and in which we perform pairwise comparison for the other label elements.

We further note that as the earliest possible arrival time \( R^a(p) \) is a resource of the label (and must be considered in the dominance check), it is in case of the VRPSTW-PL computationally disadvantageous to apply the label aggregation technique proposed by Liberatore et al. (2011) and by Bettinelli et al. (2014). With this technique labels that use the same resources are merged into one single label. It is, however, rather unlikely that any two paths have the same earliest possible arrival time, as real-world VRPSTW-PL instances typically (1) feature asymmetric travel times, (2) require seconds as time units, and (3) rely on real-world traffic information. As a consequence, the label aggregation technique is very unlikely to merge labels, but rather leads to a significant runtime increase of the labeling algorithm.
5.1.4 Labeling Algorithm

Our forward dynamic programming algorithm is based on the approach proposed by Feillet et al. (2004) for the ESPPRC of a VRP with hard time windows. The algorithm starts by allocating data structures \( \eta_i \) for storing Pareto-optimal labels for each \( i \in \mathcal{N} \). We point out that the choice of the data structure is crucial with respect to the runtime of the algorithm (and therefore of BaP). Labels are additionally stored in a pairing heap \( \zeta \), which is heap-ordered by the labels’ costs at \( \delta = 0 \). (We also evaluated sorting the heap by arrival time, but observed larger runtimes with this ordering.) Next, an initial label \( L_0 \) containing only zeros is created and added to the heap. The algorithm then iterates until either the heap is empty, or any other termination criterion is satisfied. Within each iteration the algorithm executes the two steps of (1) fetching and removing the label \( L_i \) with lowest costs from the heap (best-first-search with complexity \( O(\log |\zeta|) \)), and (2) trying to extend \( L_i \).

Extending \( L_i \) is done by iterating over all neighboring nodes \( j \) of \( i \). For each neighbor \( j \) we check (with complexity \( \Theta(1) \)) if \( p \) can be extended by edge \((i,j)\) without violation of resource constraints. If the extension is feasible, a new label \( L_j \) is created with complexity \( \Theta(|p|) \). New labels are only added to \( \eta_j \) and \( \zeta \) if they are not dominated by any label in \( \eta_j \). Dominated labels are removed from \( \eta_j \) and \( \zeta \). The complexity of the dominance check and update of \( \eta_j \) depends on the data structure selected for \( \eta_j \) and will be discussed in Section 5.2.2. Removing dominated labels from the heap \( \zeta \) is done in \( O(\log |\zeta|) \), and adding a label to the heap is done in \( \Theta(1) \).

Note that most of the runtime of BaP approaches is typically spent for solving the pricing problem, and that therefore the computational complexity of the labeling algorithm is crucial. In Section 5.2.2 we introduce the TCL-ND method to realize the dominance checking step in the labeling algorithm at low computational complexity.

5.2 Tree Compatible Labels and ND-Trees

In Section 5.2.1 we introduce tree compatible labels as labels with a fixed dimensionality, where cost functions are represented by a fixed set of breakpoints. In Section 5.2.2 we adapt the dominance criteria of Section 5.1.3 to tree compatible labels and prove that the resulting variant of our labeling algorithm solves Problem 3 to optimality. In Section 5.2.3 we propose dominance checking with ND-trees, where tree compatible labels are leveraged to increase the computational efficiency of the labeling algorithm.

5.2.1 Tree Compatible Labels

A label, as defined in Section 5.1.4 contains the cost function \( R^c(p, \delta) \) at the label’s path \( p \) (cf. Eq. (14)). Storing the cost function requires storing the function’s breakpoints in terms of both their positions \( b_0 \in \mathcal{B} \) and values \( R^c(p, \delta_0) \). Recall that the number and positions of the breakpoints depend on \( p \), and that the cost function definition of Section 5.1.2 implies that no label can contain a cost function with more than \( \delta_{\text{max}} + 1 \) breakpoints. The number of breakpoints required for representing the cost function of a label is of particular importance for dominance checking, where the cost functions of two labels must be compared by computing their values at the union of the two involved sets of breakpoint positions (cf. Eq. (18)). In this section, we keep this union constant by introducing new labels that represent the cost function by a fixed, path-independent set of breakpoint positions. These labels allow for both the use of cost function approximations and the use of ND-trees (cf. Section 5.2.3) for the data structures \( \eta_i \) in the labeling algorithm of Section 5.1.4. Due to the latter, we refer to the new labels as tree compatible labels. We further refer to the use of such labels in the labeling algorithm as tree compatible labeling (TCL). We consider the following two TCL variants:

- Exact TCL: We represent the cost function by the breakpoint positions that are necessary to guarantee that the forward dynamic programming approach of Section 5.1 still solves Problem 3 to optimality. This TCL approach replaces the union in Eq. (18) by the set

\[
\mathcal{B}^{ex} = \left\{ 0, \tau_1, 2\tau_1, \ldots, \left\lfloor \frac{\delta_{\text{max}}}{\tau_1} \right\rfloor \tau_1, \left\lfloor \frac{\delta_{\text{max}}}{\tau_1} \right\rfloor \tau_1, \delta_{\text{max}} \right\}, \tag{23}
\]

where \( \tau_1 = \gcd \left\{ \max \left\{ t^o_i - t^A_i, 0 \right\}, \forall i \in \mathcal{C} \right\}, \left\{ t^o_i, \forall i \in \mathcal{C} \right\} \right\} \) is the smallest possible delay that can be caused by adding any customer to any path, i.e., \( \mathcal{B}^{ex} \) contains all points at which the gradient of any label’s cost function may change.
- **Optimistic TCL**: As small \( \gamma \) tends to increase the computational burden of dominance checks with exact TCL, we additionally consider a TCL approach that approximates \( R^c(p, \delta) \) by a smaller number \( |B^{op}| < |B^{ex}| \) of keypoints, replacing the union in Eq. (18) with

\[
B^{op} = \left\{ 0, \left[ \frac{\delta_{\text{max}}}{|B^{op}| - 2}, 2 \frac{\delta_{\text{max}}}{|B^{op}| - 2} \right], \ldots, \left( (|B^{op}| - 3) \frac{\delta_{\text{max}}}{|B^{op}| - 2} \right), \delta_{\text{max}} \right\},
\]

i.e., \(|B^{op}| - 2\) keypoints are placed with roughly equal distance between 0 and \( \delta_{\text{max}} \). We refer to this TCL version as optimistic TCL to reflect the fact that with the cost function approximation the labeling algorithm may not be able to find existing columns with negative reduced costs.

Note that the piecewise linear cost function approximation of optimistic TCL,

\[
\bar{R}^c(p, \delta) = \max_b \left\{ \bar{R}^c_b(p, \delta) := R^c(p, \delta_b) + (\delta - \delta_b) \frac{R^c(p, \delta_{b+1}) - R^c(p, \delta_b)}{\delta_{b+1} - \delta_b} \mid \delta_b \in B^{op} \setminus \delta_{\text{max}} \right\},
\]

is an upper bound for \( R^c(p, \delta) \), as illustrated in Figure 4 that we discuss in the following Section 5.2.2.

With the tree compatible labels

\[
\bar{L}_i = (p, R^N(p), R^u(p), R^d(p), R^c(p, \delta)) \in \mathbb{N}^5 \times \mathbb{R}^{|B^{op}|},
\]

the dominance checking step in the forward dynamic programming approach of Section 5.1 is prone to approximation errors. In the presence of such errors, a dominance check may lead to the conclusion that \( \bar{L}_1^{\text{op}} \) dominates \( \bar{L}_2^{\text{op}} \), although a \( \delta \in B^{ex} \setminus B^{op} \) exists for which \( R^c(p^1, \delta) > R^c(p^2, \delta) \). In Section 5.2.2 we provide two alternative approaches to mitigating the impact of approximation errors, i.e., to using optimistic TCL without relaxing the optimality guarantee of BaP.

### 5.2.2 Optimality with Optimistic TCL

In order to use optimistic TCL without relaxing the optimality guarantee of BaP, we may (1) replace Eq. (18) by a dominance criterion that guarantees an optimal solution to Problem 3 or, alternatively, we may (2) substitute \( R^{ex} \) for \( B^{op} \). In the latter case, it is also possible to compute a lower bound on the optimal solution of the pricing problem when \( B^{op} \) is used. We formulate both approaches in the following, and prove their correctness in the appendix.

To derive dominance criteria that guarantee an optimal solution to Problem 3 with optimistic TCL, we introduce a lower bound \( \underline{R}^c(p, \delta) \) for \( R^c(p, \delta) \). We define for \( \delta \in [0, \delta_{\text{max}}] \), and for \( b \in \{1, 2, \ldots, |B^{op}| - 2, |B^{op}| - 1 \} \), the linear functions

\[
\underline{R}^c(p, \delta, b) := R^c(p, \delta_b) + (\delta - \delta_b) \nabla R^c(p, \delta_b^+),
\]

\[
\underline{R}^c(p, \delta, b) := R^c(p, \delta_{b+1}) + (\delta - \delta_{b+1}) \nabla R^c(p, \delta_{b+1}^-),
\]

where \( \nabla R^c(p, \delta^+) \) and \( \nabla R^c(p, \delta^-) \) are the RHS and LHS gradients of the cost function at \( \delta \). With these linear functions we introduce the piecewise linear lower bound \( \underline{R}^c(p, \delta) \leq R^c(p, \delta) \) as

\[
\underline{R}^c(p, \delta, b) := \max \{ \underline{R}^c(p, \delta, b), \underline{R}^c(p, \delta, b) \} \text{ for } \delta \in [\delta_b, \delta_{b+1}].
\]

Figure 4a illustrates both \( \bar{R}^c(p, \delta) \) and the cost function approximation \( \bar{R}^c(p, \delta) \) between two neighboring break point locations \( \delta_b, \delta_{b+1} \in B^{op} \) for two example labels \( \bar{L}_1^{\text{op}}, \bar{L}_2^{\text{op}} \) with paths \( p^1, p^2 \). As the keypoints intersect, we conclude that none of the labels dominates the other in \([\delta_b, \delta_{b+1}] \). However, if dominance is checked by Eq. (18) and \( B^{op} \), the approximation error in \( \bar{R}^c(p, \delta) \) leads to the conclusion that \( \bar{L}_1^{\text{op}} \) dominates \( \bar{L}_2^{\text{op}} \) within \([\delta_b, \delta_{b+1}] \). We denote the upper bound on the approximation error at a point \( \delta \in [0, \delta_{\text{max}}] \) as

\[
\epsilon(p^1, p^2, \delta) = R^c(p^1, \delta) - R^c(p^2, \delta),
\]

and use this bound in the following Theorem 2 to provide sufficient criteria for checking the dominance between two tree compatible labels.
Figure 4: Cost function $R^c(p, \delta)$, approximate cost function $\hat{R}^c(p, \delta)$, and lower bounds for $R^c(p, \delta)$.

(a) Cost functions $R^c(p^1, \delta)$, $R^c(p^2, \delta)$ of labels $L^1_{i}, L^2_{i}$ over $[\delta_b, \delta_{b+1}]$. The cost function approximation $\hat{R}^c(p, \delta)$ and the lower bound $R^c(p^2, \delta)$ lead to the maximum upper bound $\epsilon(p^1, p^2, \delta^*_b)$ of the approximation error.

(b) Cost function $R^c(p, \delta)$, of label $L_i$ over $[\delta_b, \delta_{b+1}]$. The cost function approximation $\hat{R}^c(p, \delta)$ and the lower bound $R^c(p, \delta)$ lead to the upper bound $\epsilon(p, \delta)$ of the approximation error for $L_i$ at point $\delta$.

**Theorem 2.** Label $L_i^1 = (p^1, R^c(p^1), R^c(p^1), R^c(p^1), R^c(p^1), R^c(p^1), R^c(p^1), R^c(p^1), R^c(p^1), R^c(p^1), R^c(p^1))$ dominates another label $L_i^2 = (p^2, R^c(p^2), R^c(p^2), R^c(p^2), R^c(p^2), R^c(p^2), R^c(p^2), R^c(p^2), R^c(p^2), R^c(p^2), R^c(p^2))$, if Eqs. [15]-[17] hold, and if the maximum approximation error is non-positive, i.e.

$$\forall \delta_b \in \left\{0, \delta^*_0, \delta^*_1, \ldots, \delta^*_{|B^op|-1}, \delta^*_{|B^op|-1}, \delta_{|B^op|}\right\} : \epsilon(p^1, p^2, \delta_b) \leq 0,$$

where

$$\delta^*_b = \frac{R^c(p^2, \delta_{b+1}) - R^c(p^2, \delta_b) + \delta_b \nabla R^c(p^2, \delta^*_b) - \delta_{b+1} \nabla R^c(p^2, \delta^*_{b+1})}{\nabla R^c(p^1, \delta_b) - \nabla R^c(p^2, \delta^*_{b+1})}$$

is the point at which the distance between the lower and the upper bound is greatest within $[\delta_b, \delta_{b+1}]$.

**Proof.** Proof of Theorem 2 We formulate the proof of Theorem 2 in Section 11 of the appendix.

The dominance criteria imposed by Theorem 2 ensure that optimistic TCL solves Problem 3 to optimality. Compared with the original dominance criteria of Section 5.1.3, the criteria for optimistic TCL tend to lower the rate of discarded labels. Therefore we alternatively consider using optimistic TCL with the original dominance criteria, given by Eqs. [15]-[18], and to substitute $B^{ex}$ for $B^{op}$ whenever no column with negative reduced costs is found. To derive a lower bound for the objective function value of the optimal solution to Problem 3 we introduce a new lower bound $\bar{R}^c(p, \delta) \leq R^c(p, \delta)$ for the cost function $R^c(p, \delta)$. We let

$$\bar{R}^c(p, \delta) = \max_b \left\{\widehat{R}^c(p, \delta) := \begin{cases} -\infty, & \text{if } \delta < \delta_b \\ R^c(p, \delta_b), & \text{otherwise} \end{cases} \right\}; \delta_b \in B^{op} \setminus \delta^{max},$$

and introduce (as illustrated in Figure 4b) the upper bound $\epsilon(p, \delta)$ of the approximation error at $\delta \in \{0, \ldots, \delta^{max}\}$ for label $L_i$ with path $p$ as

$$\epsilon(p, \delta) := \bar{R}^c(p, \delta) - \widehat{R}^c(p, \delta).$$

With these definitions we are now able to formulate a lower bound on the objective function value of the optimal solution of Problem 3.
Figure 5: Example of a two-dimensional ND-tree representation of a set \( \eta \) with five labels (black dots). Each region indicates the dominance relationship between a new label from the region and \( \eta \). Question marks indicate that the dominance relation is undecided in a region.

(a) Root node, representing the full set \( \eta_i \).
(b) Leaf node containing two of the labels in \( \eta_i \).
(c) Leaf node containing three of the labels in \( \eta_i \).

Figure 6: A new label (black star) is added to the ND-tree given by Figure 5.

(a) The new label does not dominate the root node’s ideal point and is not dominated by its nadir point.
(b) The new label dominates a subtree’s ideal point, and thus all of the subtree’s labels.
(c) The subtree has been replaced by the new label.

Theorem 3. For TCL with \( B^\text{ex} \) and \( B^\text{op} \), let \( \eta_0^\text{ex} \) and \( \eta_0^\text{op} \) be the sets of non-dominated labels generated with the dominance criteria provided by Eqs. (15)-(18). Then, the objective function value
\[
\min \{ R_c(p) \mid L_0 \in \eta_0^\text{ex} \},
\]
where
\[
\epsilon(p) = \sum_{i=2}^{[p]-1} \epsilon(p[1:i], \delta(p[1:i], p[i+1:p[i]])) .
\]

Proof. Proof of Theorem 3. We formulate the proof of Theorem 3 in Section 11 of the appendix.

5.2.3 Dominance Checking with ND-Trees

In this section we propose to integrate a specific tree data structure with TCL in order to increase the computational efficiency of the dominance checks in the labeling algorithm. In particular, we propose for the TCL algorithm to represent the sets \( \eta_i \) of Pareto-optimal labels at node \( i \in \mathcal{N} \) (cf. Section 5.1.4) by the ND-tree data structure introduced by Jaszkiewicz and Lust (2018).

This data structure represents a set \( \eta_i \) as a (balanced) tree of hyperrectangles. Recall that tree compatible labels are characterized by their fixed dimensionality, as the cost function in the labels is represented by a fixed (and possibly small) set of breakpoint positions \( B^\text{ex} \) or \( B^\text{op} \). As a consequence, each \( \eta_i \) forms a hyperrectangle for which we can compute an ideal label \( \bar{L}_i^{\text{ideal}}(\eta_i) \) and a nadir label...
The elements of the ideal label contain (for each dimension that is relevant for dominance checking) the lowest values of all \( L_i \in \eta_i \), and the elements of the nadir label contain for each relevant dimension the largest values of all \( L_i \in \eta_i \). Representing \( \eta_i \) as a hyperrectangle with an ideal label and a nadir label implies that a new label \( L'_i \notin \eta_i \) dominates all labels in \( \eta_i \), if \( L'_i \) dominates \( L_i^{\text{ideal}}(\eta_i) \), and that \( L'_i \) is dominated by all labels in \( \eta_i \), if \( L'_i \) is dominated by \( L_i^{\text{nadir}}(\eta_i) \). A simplified (two-dimensional) example of the hyperrectangle representation of a set \( \eta_i \) is shown in Figure 5a. The figure shows the region where a new label \( L'_i \notin \eta_i \) dominates all labels in \( \eta_i \), the region where \( L'_i \) is dominated by all labels in \( \eta_i \), as well as the regions (indicated by non-dominated) where \( L'_i \) is not dominated by any label in \( \eta_i \) and does not dominate any label in \( \eta_i \). If a new label \( L'_i \) is located in a region with a question mark, \( L'_i \) may be dominated by some labels of \( \eta_i \) and may dominate some labels of \( \eta_i \).

We represent each \( \eta_i \) by an ND-tree structure in order to take advantage of ideal points and nadir points for checking dominance between a new label \( L'_i \) and the labels in \( \eta_i \), and in order to reduce the computational overhead for the case \( L_i^{\text{ideal}}(\eta_i) < L'_i < L_i^{\text{nadir}}(\eta_i) \). The latter can be achieved due to the fact that an ND-tree represents \( \eta_i \) not necessarily as one single hyperrectangle, but as a set of (ideally non-overlapping) sub-hyperrectangles, each of which corresponds to a tree node. Figure 5a shows a simplified (2-dimensional) example of an ND-tree with a root node and two leaf nodes. The root node is shown in Figure 5a and represents the hyperrectangle of all non-dominated labels generated for node \( i \) so far. The leaf nodes are shown in Figures 5b and 5c and represent sub-hyperrectangles (with their own ideal point and nadir point) of the root node’s hyperrectangle. Note that for larger sets \( \eta_i \), ND-trees may have more layers and contain internal nodes in addition to the root node and leaf nodes.

The ND-tree structure allows for the following dominance checking approach for tree compatible labels: Dominance checking for a new label \( L'_i \) starts at the root node of the ND-tree. If \( L'_i \) is dominated by the nadir label of the root node, the new label can be discarded. Likewise, if \( L'_i \) dominates the root node’s ideal label, all labels in \( \eta_i \) can be discarded, and \( L'_i \) becomes the only label in the root node. If none of these two cases applies, the dominance check for \( L'_i \) continues in the root node’s child nodes. Figure 5b shows an example dominance check for the ND-tree given by Figure 5a. In Figure 5b we observe that the new label (visualized by a star) does not dominate the root node’s ideal label and is also not dominated by the root node’s nadir label. In Figure 5b we observe that the new label dominates the ideal point of the hyperrectangle of the leaf node shown in Figure 5c. As a consequence, all labels in the leaf’s hyperrectangle are removed and the new label is added to the ND-tree (as shown in Figure 5c).

In case a newly generated label cannot be discarded and does not replace a subtree, it is added to the closest leaf node (where distance is measured to the center of a node). If a label is added, all the properties of the node’s ideal and nadir points are propagated up to the root node. Moreover, if the number of labels in a leaf node exceeds the predefined node capacity, the node becomes an internal node, its label set is split, and the labels are assigned to a predefined number of child nodes. For a more detailed description of the ND-tree structure we refer to [Jaschke and Lust, 2018]. The computational efficiency of dominance checking with ND-trees depends on how balanced the tree is that the node-splitting procedure generates.

If \( \eta_i \) is represented by a balanced ND-tree, dominance checks may be done with an average case complexity of \( \Theta(\log \log |\eta_i|) \), where \( |\eta_i| \) is the predefined maximum number of children per internal node. In the worst case, the ND-tree may degenerate to a list with dominance check complexity \( \Theta(|\eta_i|) \).

We refer to TCL with ND-trees as TCL-ND, and distinguish (depending on the applied cost function representation) between exact TCL-ND and optimistic TCL-ND. In Section 6 we compare the performance of TCL-ND with the performance of TCL with pure list structures, as well as with the performance of the state-of-the-art labeling approaches from the literature, where different labels may have different dimensions.

### 6 Branch-and-Price

In this section we present algorithmic techniques that we include in our BaP approaches for solving the VRPSTW-PL. The benefit of using these techniques is evaluated in Section 6 with the VRPSTW-PL instances that we introduce in Section 5. In Section 6.1 we detail our approach to ensuring feasibility for the initial restricted master problem. In Section 6.2 we provide an approach to considering a relaxed version of Problem 3 in the pricing step. In Section 6.3 we provide approaches to accelerating the pricing step. In Section 6.4 we present a technique for stabilizing column generation. In Section 6.5 we introduce a technique for fast generation of elementary upper bounds. In Section 6.6 we describe the branching techniques we apply.
6.1 Initialization

We generate a feasible primal solution for the restricted master problem by applying the following four greedy construction heuristics: (1) a nearest-neighbor heuristic that constructs routes by satisfying the vehicles’ capacity constraints, and that assigns routes to the available Δ-slot with the smallest ready time, (2) an extended version of the previous heuristic that additionally satisfies the customers’ time windows when constructing routes (this heuristic may fail if too few vehicles are available), (3) a nearest neighbor heuristic that satisfies the vehicles’ capacity constraints and that constructs for each Δ-slot a total of |C| routes by enforcing that each customer \( i \in C \) appears as the first customer in one of the routes, and (4) an extended version of the previous heuristic that additionally satisfies the customers’ time windows when constructing routes. We apply 2-opt to each of the generated routes before adding the routes as columns to Problem 2. Finally, we add a dummy column with high costs for each Δ-slot to avoid that branching causes infeasibility.

6.2 Relaxing the Pricing Problem

Many authors propose BaP approaches where the complexity of the pricing problem is reduced by (partially) relaxing the elementary restriction imposed by the ESPPRC. In the VRP literature, the most common relaxations are 2-cycle elimination (e.g., Desrochers et al. 1992), k-cycle elimination (e.g., Irnich 1999), and Villeneuve 2006, Fukasawa et al. 2006), as well as the \( ng \)-relaxation. The \( ng \)-relaxation was initially proposed by Baldacci et al. (2011), and may be considered as the current state-of-the-art relaxation for ESPPRCs. Hence, we adapt the \( ng \)-relaxation to our problem and evaluate its performance in Section 8.

The \( ng \)-relaxation drops the elementary path restriction \( (R^p(j) \neq 0) \), but restricts the set of nodes to which a label may be extended. Given a label \( L_{i[p]} \) with path \( p = (i_1, i_2, \ldots, i_p) \), the relaxation forbids extending \( L_{i[p]} \) to any element of

\[
\left\{ i_r \in C(L_{i[p]}) \mid i_r \in \bigcap_{k=r}^{p} N_{i_r} \right\} \cup \{i_p\},
\]

where \( C(L_{i[p]}) \) is the set of customers in \( p \), where \( N_{i_r} \) is the set of closest neighbors of \( i_r \), and where the closest neighbor \( j \in C\{i\} \) minimizes \( t_{ij}^R \). If the \( ng \)-relaxation is applied, we have to modify our labels in three ways. We substitute \( R^\text{ng} = (R_k^\text{ng} \in \{0, 1\}^{|C|} \) for \( R^p \), and we add the picking time \( R^R(p) \) as well as the loading time \( R^S(p) \) to the label. Label extension to a customer \( j \) updates \( R^\text{ng} \) by

\[
R_k^\text{ng} = \begin{cases} 0, & \text{if } (k \notin N_j \lor R_k^\text{ng} = 0) \land k \neq j \\ 1, & \text{otherwise} \end{cases}
\]

Sufficient dominance checking with the relaxation requires both replacing Eq. (16) by \( R_k^\text{ng}(p^1) = R_k^\text{ng}(p^2) \), and considering the two criteria \( R^R(p^1) \leq R^R(p^2) \), \( R^S(p^1) \leq R^S(p^2) \) in addition to the criteria given by Eqs. (15)-18. The changes are necessary to ensure that Eqs. (20) are satisfied, i.e., to ensure that picking times and loading times as well as the arrival time at the terminal node are not greater for \( p^1 \) than for \( p^2 \).

6.3 Accelerating the Pricing Step

We consider two approaches to accelerating the pricing step. Our first approach speeds up the dynamic programming algorithm of Section 6.1.4 by taking into account unreachable nodes (Fell et al. 2004) and unprofitable nodes (Libatone et al. 2011, Fell et al. 2004) note that the dynamic programming approach remains valid, if given a label \( L_i \) with path \( p \), we let \( R^R_j(p) = 1 \) for all nodes \( j \in C \) that are unreachable due to capacity constraints. Moreover, Libatone et al. (2011) show that extending \( L_i \) to a node \( j \in C \) is not profitable if the corresponding arrival time is greater than \( t_j + \frac{\lambda_j}{\mu_j} \), and that we may therefore let \( R^R_j(p) = 1 \) for all such nodes.

Our second approach accelerates the pricing step by solving the pricing problem heuristically, which is a well-known approach in column generation algorithms. In the VRP literature the most common techniques for heuristic pricing are solving the pricing problem with heuristic heuristics (e.g., Archetti et al. 2011, Ciancio et al. 2018), as well as solving the pricing problem with modified dynamic programming algorithms. In the latter case the most common types of heuristics are reducing the dynamic programming graph to a smaller subgraph (e.g., Dabia et al. 2017, Pecin et al. 2017b), and relaxing the dominance
relations used in the dynamic programming algorithm (e.g., Chabrier 2006, Ceselli et al. 2009). In Section 6.5 we evaluate the following two heuristics:

- **Dominance heuristic**: We relax the dominance relations by ignoring the \( R^e \) vector during the dominance tests. This leads to a smaller number of Pareto optimal labels at every node and thereby decreases the runtime of the pricing algorithm. This heuristic has also been applied, e.g., by Liberatore et al. (2011).

- **Subgraph heuristic**: We compute for each edge \((i, j)\) the normalized dual score \( \frac{\lambda_i^+ + \lambda_j^+}{t_{ij}} \). The score aims at approximating the profitability of \((i, j)\) with the dual prices \( \lambda_i^+ \), \( \lambda_j^+ \) and the travel time \( t_{ij} \). Based on this score we generate a subgraph by removing 50% of the least profitable edges from the original graph.

The two heuristics can be applied with the dynamic programming algorithm of Section 6.1.4 and with the pricing problem relaxation of Section 6.2. Taking into account the fact that the heuristics may fail to identify a column with negative costs (even if one exists), we follow a three step procedure when solving the pricing problem heuristically: First, we solve the pricing problem by applying both heuristics at once. If no column with negatively costs is found, we solve the pricing problem with the dominance heuristic only. If again no column with negative costs is found, we fall back to solving the pricing problem without the heuristics.

### 6.4 Stabilizing Column Generation

A number of authors propose BaP approaches with stabilization techniques to mitigate oscillations that large jumps in the dual variables may cause during column generation. The most common stabilization techniques in the VRP literature are box stabilization (e.g., Ciancio et al. 2018), dual variable smoothing (e.g., Peccin et al. 2017b, Pessoa et al. 2018) as well as interior point stabilization. The latter was initially proposed by Rousseau et al. (2007) and used for VRPs by Feillet et al. (2007) as well as by Gentile et al. (2016). The idea of interior point stabilization is to select a dual iterate from the center of the optimal dual space in every column generation iteration, and to approximate this center by averaging over a predefined number of randomly sampled extreme points. We include interior point stabilization in our empirical evaluation and approximate the center of the optimal dual space by 50 extreme points. For a detailed description of the method we refer to Rousseau et al. (2007).

### 6.5 Elementary Upper Bound

We use a three-stage heuristic to generate elementary upper bounds during column generation, and we thereby decrease the gap between the best elementary upper bound and the column generation’s lower bound. Routes generated by the upper bound heuristic are represented as a set \( R = \bigcup_{l \in \mathcal{L}} R_l \), where \( R_l \) are the routes of \( \Delta \)-slot \( l \). Each of the three stages extracts a number of candidate routes for entering \( R \). A candidate route is added to \( R \) if it cannot be merged into a route already present in \( R \) such that the cost of the resulting route is not larger than the costs of the two separate routes. Given two routes \( r^1 \), \( r^2 \) with their \( \Delta \)-slots \( l_1 \), \( l_2 \), the merge-operation attaches all customers of \( r^2 \) to the end of \( r^1 \), runs 2-opt on the resulting route, and then checks if

\[
R^l_1(\text{merge}(r^1, r^2, l_1)) \leq R^l_1(r^1) + R^l_2(r^2).
\]

The heuristic iterates over all \( \Delta \)-slots in increasing order of \( t^\Delta_1 \) and adds as many routes as possible to \( R \) in the first stage. Columns are traversed in decreasing order of the corresponding master variable’s value. The second stage ignores the order of the \( \Delta \)-slots and iterates over all columns in decreasing order of the corresponding master problem’s variable. If none of a column’s customers are visited by the routes in \( R \), it is added to the \( R \) with the lowest \( t^\Delta_1 \) and with at least one unused vehicle. The third stage ignores the order of the \( \Delta \)-slots and iterates over all columns in decreasing order of the corresponding master problem’s variable. It removes all customers that are already visited in \( R \) from each column, and adds the resulting route to \( R \) if it contains at least one customer. The route is added to the \( R \) with the lowest \( t^\Delta_1 \) and with at least one unused vehicle. The three-stage approach does not guarantee a feasible solution, but proved to be very reliable in our computational experiments.
6.6 Branching

We apply a three-stage branching approach: (1) branching on the number of routes in a \( \Delta \)-slot, (2) branching on which customer is visited in a \( \Delta \)-slot, and (3) branching on which edges are assigned to a \( \Delta \)-slot. We branch on the number of routes if there is a \( l \in \mathcal{L} \) for which \( \sum_{r \in \Omega} x_{r l} \) is fractional. If more than one fractional value exists, we branch on the \( \Delta \)-slot \( l \) with lowest \( t_{ij}^l \). Branching on the number of routes is enforced by adjusting \( b_i^\Delta \) and \( b_r^\Delta \) in the restricted master problem.

If branching on routes is not possible, we consider the second stage of our branching approach. We check if a customer is visited a fractional number of times by \( \sum_{r \in \Omega} \left(0.5 - |x_r - 0.5| \right) y_{ri} \) for all \( l \in \mathcal{L} \) and \( i \in \mathcal{C} \), and we branch on the customer with the largest fractional value. In case of ties, we prefer the \( \Delta \)-slot with smallest \( t_{ij}^l \). We opt to enforce branching in the pricing problem, as this approach allows for both stronger lower bounds (Vanderbeck and Wolsey 2010) and pricing with reduced node sets.

If branching on customers is not possible, we consider the third stage of our branching approach. We check if an edge is used a fractional number of times by \( \sum_{r \in \Omega} (0.5 - |x_r - 0.5|) x_{ij} \) for all \( l \in \mathcal{L} \) and \( (i, j) \in \mathcal{E} \), and branch on the variable with maximum value. Ties are resolved randomly. Branching on edges is conducted (as branching on customers) by removing the respective edge (node in case of customers) from the pricing problem in one sub-tree or ensuring that the edge has to be used in the other.

Finally, we reduce the number of columns that are infeasible after branching by applying a repair heuristic to each column. This heuristic iterates over the column’s path in forward direction and checks at every node if a branching constraint is violated. If a violation is detected, the node is removed from the path and the heuristic restarts at the beginning of the path. The path is removed, if it does eventually contain no customers.

7 Problem Instances and Experimental Setup

In Section 7.1 we present the problem instances that we consider in Section 8 for evaluating our BaP approaches. In Section 7.2 we provide an overview of our experimental setup.

7.1 Problem Instances

All experiments are conducted with two sets of problem instances. We use a new set of VRPSTW-PL instances that have been created by us and endorsed by the German real-time delivery service provider flaschenpost. We additionally use a set of VRPSTW instances (representing the special case of the VRPSTW-PL where all picking times and loading times are zero) that are derived from the well-known Solomon instances (Solomon 1987). We include the latter set in our study to increase comparability with works from the literature.

- **VRPSTW-PL:** We propose a set of 15 VRPSTW-PL instances. Without loss of generality we assume that time units represent seconds and let \( \Delta = 300 \) (5 minutes) to model the setting of a real-time delivery service provider (where vehicle ready times may be greater than zero due to vehicles that are about to return to the depot). Each instance has 50 customers and asymmetric travel times that do not violate the triangle inequality. The problem set contains 5 instances with relatively small time windows (72 minutes on average), 5 instances with medium-sized time windows (107 minutes on average), and 5 instances with rather large time windows (115 minutes on average). Across all instances, the number of available vehicles ranges from 21 to 37, and the number of \( \Delta \)-slots ranges from 3 to 17. Instances with smaller time windows tend to have more vehicles and more \( \Delta \)-slots than instances with larger time windows. All VRPSTW-PL instances are available in our online appendix at https://www.wi.uni-muenster.de/department/qm-logistik/vrpstw-pl-instances.

- **VRPSTW:** We adapt the 29 Solomon instances with short scheduling horizon (R1-type, C1-type, RC1-type) by considering only the first 50 customers, and by replacing the lower bounds of time windows with zero. Euclidean distances are always rounded up to an integer value. Due to the absence of picking and loading, we assume that all vehicle ready times are zero and assign only one \( \Delta \)-slot to each instance. Moreover, we bound the number of vehicles by 25.

We provide an illustrative overview of the considered instance types in Figure 2. Figure 2A compares our six instance types in terms of the empirical cumulative distribution function (ecdf) of the normalized driving times \( \frac{d_{ij}}{\max_{e \in E} d_{ij}} \). The figure shows that the shapes of the distributions for the three VRPSTW-PL instance types are of similar type (reflecting that all VRPSTW-PL instances are derived from a
Figure 7: Comparison of the VRPSTW-PL instances with small, medium, large time windows (PL-s, PL-m, PL-l), and of the VRPSTW instances with randomly allocated, clustered, mixed customer locations (R1, C1, RC1).

(a) Empirical cdf of the normalized driving time. (b) Empirical cdf of the relative leeway.

real-time delivery service case in an urban area). Solomon’s R1 instances tend to have less short driving times than the VRPSTW-PL instances, but the shape of the R1-distribution still is quite similar to the shapes of the VRPSTW-PL cdfs. The shapes of the distributions for the C1 and RC1 instances highlight the fact that clustered customer locations lead to both more very short driving times and more long driving times. Figure 7(b) compares our six instance types in terms of the empirical cdf of the relative leeway \((\bar{t}_i - t^0_i) / \bar{t}_i\), which can be interpreted as an inverse measure of the tightness of the time window of customer \(i\). The ordering of the cdfs for the three VRPSTW-PL instance types reflects the fact that instances with smaller time windows feature more customers with small leeway. In Section 7.2 we conduct all of our experiments for each of the six problem instance types presented in this section. Moreover, we vary the number of customer locations by conducting each experiment for the first 10 customers, for the first 30 customers, and for all 50 customers of each problem instance.

7.2 Experimental Setup

The main goals of our experiments are (1) to evaluate the performance of exact TCL-ND (using \(B^e\)) in the pricing step, (2) to evaluate the performance gains of optimistic TCL-ND (using \(B^{op}\)) in the column generation step, and (3) to identify a BaP approach that is able to cope with the runtime requirements of real-time vehicle routing applications. To achieve these goals, we conduct the following three sets of experiments:

- **Pricing with exact TCL-ND**: In Section 8.1 we evaluate the performance of exact TCL-ND. In particular, we compare the performances of different exact pricing algorithms (with and without TCL-ND) on a set of pricing problem instances. We generate this set of pricing instances by running column generation for all problem instances of Section 7.1 and then randomly sampling from the occurring pricing problems 50 distinct pricing problem instances for each VRPSTW-PL instance and for each VRPSTW instance. In order to mitigate the random influence of the state of the computer’s operating system on runtimes, we solve each pricing problem instance 10 times and consider the average runtime as the actual runtime. We terminate the pricing algorithms if 10 columns with negative reduced costs have been found or if the runtime limit of 120 seconds is reached.

- **Column generation with TCL-ND**: In Section 8.2 we evaluate the performance gains of using optimistic TCL-ND. In particular, we evaluate for the column generation step the benefit of (1) optimistic TCL-ND, and of (2) combining optimistic TCL-ND with the pricing heuristics of Section 6.3. The corresponding experiments comprise a total of 56 distinct column generation solvers. To mitigate the randomness inherent to column generation, we run each solver for each problem instance 10 times (using Gurobi with a different random seed each time, and using common random numbers for the interior point stabilization) and calculate the corresponding average runtimes. Each run has a runtime limit of one hour. If a column generation solver uses the pricing heuristics of
In all experiments we let \( \kappa_i = 2 \) for all \( i \in \mathcal{C} \), i.e., we assume that the cost of violating a customer’s time window by one unit of time is twice as high as the cost of driving for one additional unit of time. In the experiments with TCL-ND, ND-tree nodes may store up to 10 labels, and generate 5 offspring ND-nodes during node splitting. In all experiments, we accelerate pricing by considering unreachable nodes and unprofitable nodes (cf. Section 6.3), and we always run the 2-opt heuristic to improve the routes found by the pricing algorithm. All the approaches considered in our experiments use the ng-relaxation (cf. Section 6.2) with a neighborhood size of 5. Our algorithms are written in Julia 1.5 [Bezanson et al. 2017], and our linear programs are modeled with JuMP 0.18.6 [Dunning et al. 2017] and solved with Gurobi 8.1.1. All experiments run on a Slurm cluster with two Intel Xeon Platinum 8168 2.7GHz CPUs (3.7GHz Turbo, 24C, 10.4GT/s 3UPI, 33MB Cache).

8 Empirical Results

In this section, we present empirical results that show that TCL-ND leads to significant performance gains for both the VRPSTW-PL and the VRPSTW, and that with this new method BaP is able to cope with the runtime requirements of real-time vehicle routing. In Section 8.1 we evaluate the performance of exact TCL-ND in the pricing step. In Section 8.2 we evaluate the additional performance gains that can be realized in the column generation step by optimistic TCL-ND, and by the combination of optimistic TCL-ND with heuristic pricing. Finally, we use the results of Sections 8.1 and 8.2 to propose a BaP approach for real-time vehicle routing with picking, loading and soft time windows, and evaluate its performance in Section 8.3.

8.1 Pricing with exact TCL-ND

We evaluate the performance of exact TCL-ND by comparing the following dynamic programming approaches for solving Problem 3:

- **Exact TCL-ND**: We integrate both exact TCL-ND and the dominance criteria of Section 6.2 into the forward dynamic programming algorithm of Section 5.1.4.

- **Exact TCL with lists**: We use the same approach as before (with \( B^{\text{ex}} \)), but replace the ND-tree structure with a standard list data structure.

- **Dynamic list based labeling with forward dynamic programming**: We use the same approach as before, but without TCL. Instead of using TCL, we represent the (exact) cost function in terms of a dynamic list of the breakpoints where the gradient changes.

- **Dynamic list based labeling with bidirectional dynamic programming**: We use the same approach as before, but replace the forward dynamic programming algorithm with an adapted version of the bidirectional dynamic programming algorithm proposed by Righini and Salani [2008]. We include this approach as a benchmark, as bidirectional dynamic programming is most suitable for the case where (exact) cost functions are represented in terms of dynamic lists.

For notational convenience, we abbreviate exact TCL-ND as \( T^{\text{ex}} \) (‘T’ stands for tree), exact TCL with lists as \( L^{\text{ex}} \) (‘L’ stands for list), dynamic list based labeling with forward dynamic programming as \( L^{\text{df}} \), and dynamic list based labeling with bidirectional dynamic programming as \( L^{\text{db}} \). Table 3 compares \( T^{\text{ex}} \) with \( L^{\text{ex}} \) (top), \( T^{\text{ex}} \) with \( L^{\text{df}} \) (center), and \( T^{\text{ex}} \) with \( L^{\text{db}} \) (bottom) for all problem types defined in Section 7. The table shows for each problem type and each pair of approaches how many percent of the respective problem instances can be solved (under the conditions defined in Section 7.2) by \( T^{\text{ex}} \), and by the respective competing approach. Moreover, the table compares for each problem type the runtimes.
The comparison of \( T_{ex} \) and \( L_{ex} \) (top rows in Table \ref{tab:comparison}) shows that the performance of TCL increases significantly if the ND-tree structure is used. Within the runtime of 120 seconds per problem instance, \( T_{ex} \) is able to solve all instances with 10 and 30 customers as well as the vast majority of the instances with 50 customers. Note that merely in case of the VRPSTW-PL with small time windows, \( T_{ex} \) solves only a fraction (76\%) of the instances within the given runtime. In contrast, \( L_{ex} \) already struggles with the 30 customer VRPSTW-PL instances with small time windows (93\% of all instances solved), and fails with more than half of all 50 customer VRPSTW-PL instances. The fact that even \( L_{ex} \) is able to solve all VRPSTW instances reflects the fact that problems without picking and loading, i.e., problems without delays, are less challenging. Table \ref{tab:comparison} shows that \( T_{ex} \) is not only able to solve a much larger number of instances than \( L_{ex} \), but that the ND-tree structure also implies significant runtime improvements. In the VRPSTW case the runtimes of \( T_{ex} \) consume on average between 34\% and 65\% of the runtimes of \( L_{ex} \).

In the VRPSTW case the average relative runtimes of \( T_{ex} \) are even lower, as \( T_{ex} \) consumes between 13\% and 43\% of the runtimes of \( L_{ex} \). Taking into account that, if not all problem instances can be solved, the runtime shown in the table can be considered as upper bounds on the actual runtime, we conclude that the runtime advantage of \( T_{ex} \) seems to increase as the number of customers in a VRPSTW-PL instance increases.

Table \ref{tab:comparison} also shows that TCL-ND clearly outperforms the non-TCL approaches \( L_{df} \) and \( L_{db} \). First of all, we observe that both non-TCL approaches perform even worse than \( L_{ex} \) in terms of the number of solved instances. Note that in some cases the number of solved instances drops significantly if we apply \( L_{db} \). Secondly, we observe for the VRPSTW that \( L_{df} \) yields the same results as \( L_{ex} \) (due to the cost function structure without picking and loading), and that the relative runtimes of \( T_{ex} \) with respect to \( L_{db} \) tend to be significantly lower than the relative runtimes of \( T_{ex} \) with respect to \( L_{ex} \). Thirdly, we observe for the VRPSTW-PL that \( T_{ex} \) always has significantly lower runtimes than \( L_{df} \), and that the two cases where \( T_{ex} \) does not have significantly lower runtimes than \( L_{db} \) (PL-s and PL-m with 50 customers) are cases where \( L_{db} \) fails to solve about 75\% of the respective instances.

In summary, we conclude that the ND-tree structure leads to a significant performance boost for TCL, and that TCL-ND significantly outperforms dynamic list-based labeling with forward dynamic

### Table 3: Comparison of \( T_{ex} \) and \( L_{ex} \), \( T_{ex} \) and \( L_{df} \), \( T_{ex} \) and \( L_{db} \).

| & VRPSTW-PL & VRPSTW |
|---|---|---|
| \( T_{ex} \) & \( L_{ex} \) & \( L_{df} \) & \( L_{db} \) |
| | \( PL_{ex} \) & \( PL_{ex} \) & \( PL_{ex} \) & \( PL_{ex} \) |
| 0.1 | 0.87 | 0.65 | 0.38 |
| 0.2 | 0.53 | 0.32 | 0.2 |
| 0.3 | 0.28 | 0.18 | 0.1 |
| avg. | 0.37 | 0.22 | 0.15 |
| | \( PL_{1} \) & \( PL_{1} \) & \( PL_{1} \) & \( PL_{1} \) |
| 0.1 | 0.53 | 0.32 | 0.2 |
| 0.2 | 0.28 | 0.18 | 0.1 |
| 0.3 | 0.19 | 0.15 | 0.1 |
| avg. | 0.37 | 0.22 | 0.15 |
| | \( R1 \) & \( R1 \) & \( R1 \) & \( R1 \) |
| 0.1 | 0.53 | 0.32 | 0.2 |
| 0.2 | 0.28 | 0.18 | 0.1 |
| 0.3 | 0.19 | 0.15 | 0.1 |
| avg. | 0.37 | 0.22 | 0.15 |
| | \( RCl \) & \( RCl \) & \( RCl \) & \( RCl \) |
| 0.1 | 0.53 | 0.32 | 0.2 |
| 0.2 | 0.28 | 0.18 | 0.1 |
| 0.3 | 0.19 | 0.15 | 0.1 |
| avg. | 0.37 | 0.22 | 0.15 |
Table 4: Comparison of $T^{op}$ and $T^{ex}$, $T^{op-h}$ and $T^{op}$.

<table>
<thead>
<tr>
<th></th>
<th>VRPSTW-PL</th>
<th>PL-ns</th>
<th>PL-nm</th>
<th>PL-nl</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>avg.</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The runtime limit is 60 minutes per instance. Table entries a|b|c|d read as follows. a: fraction of problem instances that $T^{op}$ ($T^{op-h}$) solves, b: fraction of problem instances that $T^{ex}$ ($T^{op}$) solves, c and d: mean and median of the relative runtime (runtime of $T^{op}$ ($T^{op-h}$))/ (runtime of $T^{ex}$ ($T^{op}$)) for all instances that both approaches solve. Italic letters mark cases where runtimes of two approaches do not differ significantly according to the Wilcoxon signed-rank test with confidence level 0.01.

programming as well as with bidirectional dynamic programming.

8.2 Column Generation with TCL-ND

In this section we evaluate the additional performance gains that can be realized in the column generation step by replacing exact TCL-ND with optimistic TCL-ND. In particular, we evaluate for column generation the performance gains by optimistic TCL-ND, and by the combination of optimistic TCL-ND with the pricing heuristics proposed in Section 6.3. We consider the following (exact) column generation solvers:

- **Column generation with exact TCL-ND**: As our benchmark we use column generation with TCL-ND (the best pricing approach of Section 8.1). We run column generation without stabilization and terminate an iteration as soon as 10 columns with negative reduced costs are found.

- **Column generation with optimistic TCL-ND**: We use the same approach as before, but replace $B^{ex}$ with $B^{op} = \{0, \delta^{max}\}$. We terminate an iteration as soon as 10 columns with negative reduced costs are found. If no columns with negative costs are found, we fall back to $B^{ex}$ for the current iteration.

- **Column generation with optimistic TCL-ND and heuristic pricing**: We use the same approach as before, but additionally integrate the pricing heuristics of Section 6.3. We start a column generation iteration by solving the pricing problem with $B^{op}$ and with both the subgraph heuristic and the dominance heuristic. As soon as 100 columns with negative reduced costs are found we terminate the iteration. If no columns with negative costs are found, we solve the pricing problem with $B^{op}$ and with the dominance heuristic only. As soon as 10 columns with negative reduced costs are found we terminate the iteration. If again no columns with negative reduced costs are found, we fall back to $B^{ex}$.

For notational convenience, we abbreviate column generation with exact TCL-ND as $T^{ex}$, column generation with optimistic TCL-ND as $T^{op}$, column generation with optimistic TCL-ND and heuristic pricing as $T^{op-h}$. Table 4 compares $T^{op}$ with $T^{ex}$ (top), and $T^{op-h}$ with $T^{op}$ (bottom) for all VRPSTW-PL problem types defined in Section 7.1 and under the conditions defined in Section 7.2. We do not show results for the VRPST instances, as the absence of picking and loading implies equivalence of $T^{ex}$ and $T^{op}$. The comparison of $T^{op}$ and $T^{ex}$ (top rows in Table 4) shows that in most cases the performance
Table 5: Comparison of $T^{\text{op-h}}$ and $L^{\text{op-h}}$, $T^{\text{op-h}}$ with and without stabilization, $T^{\text{op-h}}$ with and without DSSR.

<table>
<thead>
<tr>
<th></th>
<th>VRPSTW-PL</th>
<th>VRPST W</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PL-ns</td>
<td>PL-nm</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.73</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>0.36</td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>0.34</td>
</tr>
<tr>
<td>avg.</td>
<td>0.78</td>
<td>0.22</td>
</tr>
</tbody>
</table>

The runtime limit is 60 minutes per problem instance. Table entries a|b|c|d read as in Table 4. Italic letters mark cases where runtimes of two approaches do not differ significantly according to the Wilcoxon signed-rank test with confidence level 0.01.

The runtime limit is 60 minutes per problem instance. Table entries a|b|c|d read as in Table 4. Italic letters mark cases where runtimes of two approaches do not differ significantly according to the Wilcoxon signed-rank test with confidence level 0.01.

The runtime limit is 60 minutes per problem instance. Table entries a|b|c|d read as in Table 4. Italic letters mark cases where runtimes of two approaches do not differ significantly according to the Wilcoxon signed-rank test with confidence level 0.01.

The runtime limit is 60 minutes per problem instance. Table entries a|b|c|d read as in Table 4. Italic letters mark cases where runtimes of two approaches do not differ significantly according to the Wilcoxon signed-rank test with confidence level 0.01.

The runtime limit is 60 minutes per problem instance. Table entries a|b|c|d read as in Table 4. Italic letters mark cases where runtimes of two approaches do not differ significantly according to the Wilcoxon signed-rank test with confidence level 0.01.

The runtime limit is 60 minutes per problem instance. Table entries a|b|c|d read as in Table 4. Italic letters mark cases where runtimes of two approaches do not differ significantly according to the Wilcoxon signed-rank test with confidence level 0.01.

The runtime limit is 60 minutes per problem instance. Table entries a|b|c|d read as in Table 4. Italic letters mark cases where runtimes of two approaches do not differ significantly according to the Wilcoxon signed-rank test with confidence level 0.01.

Table 5 provides a more detailed analysis of the performance of $T^{\text{op-h}}$. The top rows of the table show the performance impact of using the ND-tree structure for column generation with $T^{\text{op-h}}$ by comparing $T^{\text{op-h}}$ with its list-based TCL counterpart $L^{\text{op-h}}$. Note that $T^{\text{op-h}}$ significantly outperforms $L^{\text{op-h}}$ for all instance types, and that the advantage of $T^{\text{op-h}}$ is extremely large for the VRPSTW-PL instances with 50 customers. The center rows of Table 5 show that for our problem instances $T^{\text{op-h}}$ cannot be further improved by interior point stabilization (cf. Section 6.4). Nevertheless we observe that with stabilization all problem instances can still be solved, and that the negative impact of stabilization on the runtime of $T^{\text{op-h}}$ clearly decreases with increasing customer number. Finally, the bottom rows of Table 5 show that for our problem instances $T^{\text{op-h}}$ cannot be further improved by applying DSSR (cf. Section 6.2). Note that with DSSR not all VRPSTW-PL instances with small time windows can be solved, and that the negative impact of DSSR on the runtime increases with increasing customer numbers.

In summary, we conclude that both approximating the cost function and integrating heuristic pricing lead to a significant performance boost for TCL-ND. Moreover, we conclude that for our problem instances neither stabilization nor DSSR can improve the performance of column generation with optimistic TCL-ND and heuristic pricing.

8.3 Branch-and-Price with TCL-ND

In this section, we evaluate column generation with optimistic TCL-ND and heuristic pricing (denoted as $T^{\text{op-h}}$ in the previous section) within the BAP framework proposed in Section 6. Recall that real-time delivery service providers typically require runtimes of only a few minutes for their routing algorithm, and that this requirement is integrated in our VRPSTW-PL instances in terms of $\Delta$-slots with granularity $\Delta = 300$ seconds (5 minutes). As $\Delta$ represents the intended runtime limit, we focus primarily on runtimes of TCL-ND increases significantly if the cost function is approximated. Within the given runtime of 60 minutes per problem instance, $T^{\text{op-h}}$ is able to solve all problem instances, whereas $T^{\text{op}}$ solves only 80% of the instances with small time windows. The runtime comparisons show that $T^{\text{op}}$ and $T^{\text{op-h}}$ perform similarly for the 10 customer instances, and that an increasing number of customers leads to an increasing runtime advantage of $T^{\text{op-h}}$. In case of the VRPSTW-PL instances with small (medium) time windows, $T^{\text{op-h}}$ consumes on average only 43% (34%) of the runtime of $T^{\text{op}}$.

The comparison of $T^{\text{op-h}}$ and $T^{\text{op}}$ (bottom rows in Table 4) shows that both approaches solve all problem instances, and that for all instances with 50 customers $T^{\text{op-h}}$ leads to significantly better runtimes than $T^{\text{op}}$. Due to the fact that the runtime advantage of $T^{\text{op-h}}$ tends to vanish as the VRPSTW-PL instances get small, we cannot draw a general conclusion about which of the two approaches is better. However, given the relevance of larger problem instances for many real-world applications, we prefer $T^{\text{op-h}}$.
Figure 8: Gaps of BaP with column generation solvers $T_{op}^h$ and $L_{op}^h$. Each line represents the average gap of all runs with the VRPSTW-PL instances with medium time windows and 50 customers.

![Figure 8](image-url)

Table 6: Gaps of BaP with $T_{op}^h$ for the problem instances with 50 customers.

<table>
<thead>
<tr>
<th>runtime</th>
<th>VRPSTW-PL</th>
<th>VRPSTW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PLs</td>
<td>PL-m</td>
</tr>
<tr>
<td>1</td>
<td>0.01 [0.00, 0.00]</td>
<td>0.01 [0.01, 0.01]</td>
</tr>
<tr>
<td>2</td>
<td>0.01 [0.00, 0.00]</td>
<td>0.01 [0.01, 0.01]</td>
</tr>
<tr>
<td>3</td>
<td>0.00 [0.00, 0.00]</td>
<td>0.01 [0.00, 0.01]</td>
</tr>
<tr>
<td>4</td>
<td>0.00 [0.00, 0.00]</td>
<td>0.00 [0.00, 0.01]</td>
</tr>
<tr>
<td>5</td>
<td>0.00 [0.00, 0.00]</td>
<td>0.00 [0.00, 0.01]</td>
</tr>
<tr>
<td>15</td>
<td>0.00 [0.00, 0.00]</td>
<td>0.00 [0.00, 0.00]</td>
</tr>
</tbody>
</table>

Table entries a/b/c/d read as follows. a and b: mean and median relative posteriori gap, c and d: mean and median relative MIP gap. All gaps are rounded to two digits after the decimal point.

between 1 and 5 minutes in our analysis. We use the following two measures to evaluate the performance of our BaP approach:

- **Relative MIP gap**: We calculate the relative MIP gap during execution of BaP by dividing the difference between the quality of the current elementary upper bound and the quality of the current lower bound, by the quality of the current elementary upper bound.

- **Relative posteriori gap**: We calculate the relative posteriori gap by dividing the difference between the quality of the current elementary upper bound and the quality of the best lower bound that is known for the problem instance, by the quality of the current elementary upper bound. Note that the best known lower bound can only be identified after the optimization run terminates, and that the relative posteriori gap is a more accurate measure of how far away our current solution is from optimality. We include the best known lower bounds for our VRPSTW-PL instances in our online appendix.

Figure 8 illustrates the gaps of BaP with $T_{op}^h$ for the example of the VRPSTW-PL instances with medium time windows and 50 customers. The figure additionally shows the corresponding gaps of BaP with the list based counterpart $L_{op}^h$ as benchmarks. Each line represents the average gap over all runs with VRPSTW-PL instances with medium time window and 50 customers. The figure shows that $T_{op}^h$ is able to reduce both the relative MIP gap and the relative posteriori gap substantially faster than $L_{op}^h$. Figure 8a shows that within the intended runtime limit of $\Delta = 5$ minutes, $T_{op}^h$ is able to reduce the relative MIP gap to almost zero, whereas $L_{op}^h$ still has a MIP gap of 80% at $\Delta$. Figure 8b illustrates that at $\Delta$ the relative posteriori gap of $T_{op}^h$ is below 1%, whereas the corresponding gap of $L_{op}^h$ still is significantly larger than 1%. Both approaches feature a long-tail in which little progress is made, which indicates that additional improvements may be achieved by integrating additional branching rules.

Our experimental results show that $T_{op}^h$ is able to solve all our problem instances with 10 customers and almost all instances with 30 customers in less than one minute of runtime. For our problems with
50 customers, Table 5 provides information about the gaps of $T^{\text{opt-h}}$ after 1, 2, 3, 4, and 5 minutes of runtime, as well as after 15 minutes of runtime. The table shows for each problem type and runtime the mean and median of the relative posteriori gaps and relative MIP gaps of all runs with the instances of the problem type.

From the relative posteriori gaps in the first row of the table we conclude that BaP with $T^{\text{opt-h}}$ is (on average) able to find feasible elementary solutions within that are within 1% of the optimal solution for the VRPSTW-PL instances, and within 2% of the optimal solution for the R1 and R C1 instances. The C1 instances prove to be solvable very close to optimality within 1 minute of runtime. The relative posteriori gaps of the VRPSTW-PL instances indicate that instances with larger time windows are more difficult to solve to optimality than instances with smaller time windows. Note that the VRPSTW-PL instances with small (medium) time windows are solved close to optimality already after 3 (4) minutes, and that the median relative posteriori gap of the VRPSTW-PL instances with large time windows approaches zero after 4 minutes, whereas the mean relative posteriori gap still is at 1% after 5 minutes. With respect to the VRPSTW-PL instances we conclude that BaP with $T^{\text{opt-h}}$ is able to reliably generate close to optimal solutions within $\Delta = 5$ minutes of runtime. Table 5 also shows that the relative MIP gaps of the VRPSTW instances tend to be lower than the relative MIP gaps of the VRPSTW-PL instances, and that at after 5 minutes the R1 and RC1 instances have average posteriori gaps of 1% and 2%. Taking into account both the gaps of the C1 instances and the median of the relative posteriori gap of R1, we conclude that $T^{\text{opt-h}}$ is able to reliably generate close to optimal solutions for most VRPSTW instances.

In summary, we conclude that TCL-ND leads to a performance boost for BaP, and that BaP with optimistic TCL-ND and pricing heuristics is able to reliably comply with the runtime requirements of real-time vehicle routing for both VRPSTW-PL and VRPSTW instances.

9 Conclusions

In this work we propose and evaluate branch-and-price approaches for vehicle routing problems with picking, loading, and soft time windows. To boost the performance of branch-and-price, we introduce the new method of tree compatible labeling with non-dominance trees (TCL-ND).

To our knowledge this is the first work on branch-and-price for vehicle routing with picking, loading, and soft time windows. Our problem formulation is new and takes into account the fact that short optimization runtime are a requirement for real-time delivery services with high customer order frequencies. The presence of soft time windows in combination with loading durations and picking durations requires that the pricing algorithm for the problem must rely on comparisons of cost functions (instead of on comparisons of scalar costs, which are less challenging computationally). We propose a forward dynamic programming algorithm to solve the pricing problem and introduce TCL-ND to accelerate the algorithm. This method represents cost functions by a fixed number of breakpoints and enables the use of non-dominance trees as data structures to store Pareto-optimal labels. We prove that the forward dynamic programming algorithm with TCL-ND solves the pricing problem to optimality.

In addition to this exact TCL-ND approach, we propose a TCL-ND approach that uses a cost function approximation to further accelerate the pricing algorithm. We prove the theoretical soundness of branch-and-price with this TCL-ND variant (that we call optimistic TCL-ND). In particular, we show that optimistic TCL-ND can be used for exact branch-and-price by either using a modified forward dynamic programming algorithm, or by using our original forward dynamic programming algorithm. For the latter case we formulate and prove a lower bound on the optimal solution of the pricing problem.

We evaluate our new method numerically with two sets of problem instances. We consider a new set of instances for vehicle routing with picking, loading, and soft time windows, and we consider a well-known set of instances for the special case of vehicle routing with soft time windows (without picking and loading). With these problem sets, we conduct three sets of experiments.

In our first set of experiments, we evaluate the performance of exact TCL-ND for solving the pricing problem. Our results show that this approach significantly outperforms both tree compatible labeling with a standard list data structure, and state-of-the-art pricing approaches where cost functions are represented by dynamically sized lists of gradients. In our second set of experiments, we evaluate the additional performance gains in the column generation step of using optimistic TCL-ND. We compare column generation solvers that rely on exact TCL-ND with solvers that rely on optimistic TCL-ND, and with solvers that additionally use pricing heuristics. Our results show that the performance of column generation improves significantly by optimistic TCL-ND, and that in case of large problem instances it is beneficial to additionally solve the pricing problem heuristically. In our third set of experiments, we evaluate TCL-ND within a branch-and-price framework. The numerical results show that this new
method boosts the performance of branch-and-price, and that our approach is able to reliably generate close to optimal solutions for problem instances with 50 customers within only a few minutes of runtime. We conclude that TCL-ND enables branch-and-price approaches that are able to satisfy the runtime requirements of real-time delivery services.

Future works may study possible improvements of the proposed approach by developing pricing relaxations that are tailored to TCL-ND, and by developing specific strong branching approaches. Moreover, another path for future work is to study the additional benefits of adding cuts such as, e.g., 2-path cuts, subset-row cuts, or subset-row cuts with limited memory.

Acknowledgement

This research was supported by the German last mile delivery company flaschenpost SE. This support is gratefully acknowledged. We thank Niklas Plath (COO of flaschenpost SE) and his team for accompanying our research and for providing us with valuable practical feedback.
References


10 Cost Function

Figure 9: Two paths, \( p^1 \) and \( p^2 \), starting at the depot 0 and ending at customer 3. Path \( p^1 \) is highlighted by bold arrows. Travel times are written next to the arrows.

Figure 10: The cost of paths \( p^1 \) and \( p^2 \) (cf. Figure 9) is shown as a function of arrival time on the left (10a) and as a function of delay on the right (10b). The example assumes that \( t_{s_i} = 0 \) for all \( i \), \( t_{o_i} = t_{p_i} = \lambda v_i = \lambda - l = 0 \) for \( i \in \{0, 1, 2, 3\} \), \( \bar{t}_1 = \bar{t}_2 = 12 \), \( \bar{t}_0 = \bar{t}_3 = \bar{t}_4 = \infty \), and that \( t_{o_4} + t_{p_4} > 0 \).

11 Optimality and Lower Bound

**Proof.** Proof of Theorem 2. Assume that \( \bar{L}_1^i \) dominates \( \bar{L}_2^i \) according to Eqs. (15)–(18). Hence, for each pair \( \delta_b, \delta_{b+1} \in \mathbb{R}^p \), we have \( R^c(p^2, \delta_b) \geq R^c(p^1, \delta_b) \) and \( R^c(p^2, \delta_{b+1}) \geq R^c(p^1, \delta_{b+1}) \). If \( \bar{L}_1^i \) does actually not dominate \( \bar{L}_2^i \), there must be a \( \delta \in [\delta_b, \delta_{b+1}] \) for which \( R^c(p^2, \delta) < R^c(p^1, \delta) \), i.e., for which the upper bound of the approximation error is \( \epsilon(p^1, p^2, \delta) > 0 \). It follows that \( \bar{L}_1^i \) actually dominates \( \bar{L}_2^i \), if the maximum upper bound of the approximation error within each interval \([\delta_b, \delta_{b+1}]\) is less or equal than
zero. Note that $R^+(p^2, \delta, b)$ is monotonically increasing for $\delta \to \infty$ and that $R^-(p^2, \delta, b)$ is monotonically decreasing for $\delta \to -\infty$ (remember that $\nabla R^+(p^2, \delta, b) \leq \nabla R^-(p^2, \delta, b)$). The upper bound $\epsilon(p^1, p^2, \delta)$ is thus maximal within $[b_0, b_{b+1}]$ at the point where $R^+(p^2, \delta, b)$ and $R^-(p^2, \delta, b)$ intersect. This intersection is located at

$$
\delta_b^* = \frac{R^+(p^2, \delta_{b+1}) - R^-(p^2, \delta_b) + \delta_b \nabla R^+(p^2, \delta_b^*) - \delta_{b+1} \nabla R^-(p^2, \delta_{b+1})}{\nabla R^+(p^2, \delta_b^*) - \nabla R^-(p^2, \delta_{b+1})}.
$$

\hfill \Box

**Proof.** Proof of Theorem 3. Let $\bar{L}_0 \in \eta_0^{\text{op}}$ be a label with route $p^1 = (0, \ldots, 0)$, and let $\bar{L}_1$ be the corresponding label for subpath $p^1 := p^1[1:i]$. Moreover, let $\bar{L}_2$ be a label that is dominated by $\bar{L}_1$ according to Eqs. (15)-(18). Denoting the path of $\bar{L}_2$ as $p^2$, we may conclude from Eq. (18) that for all $\delta \in [0, \delta^{\text{max}}]$: $R^-(p^2, \delta) \leq R^-(p^2, \delta)$. We use this relationship to show that for a given backward path $\bar{p} \in \mathcal{P}(p^2)$, the approximation error $\epsilon(p^1, \delta(p^1, \bar{p}))$ for $L_1$ is an upper bound for the actual cost difference between the two elementary routes $p^1 \cup \bar{p}$ and $p^2 \cup \bar{p}$. With the notation introduced in Section 5.1.3, this cost difference $R^c(p^1 \cup \bar{p}) - R^c(p^2 \cup \bar{p})$ can be written as

$$
R^c(p^1, \delta(p^1, \bar{p})) + R^c(\bar{p}, \delta(p^1, \bar{p})) - \left( R^c(p^2, \delta(p^2, \bar{p})) + R^c(\bar{p}, \delta(p^2, \bar{p})) \right),
$$

and due to $R^c(p^1 \cup \bar{p}) - R^c(p^2 \cup \bar{p}) \leq R^c(p^1, \delta(p^1, \bar{p})) - R^c(p^2, \delta(p^2, \bar{p}))$. (30)

Moreover, as $\delta(p^1, \bar{p}) \leq \delta(p^2, \bar{p})$, it follows that the RHS of Eq. (30) is less than or equal

$$
R^c(p^1, \delta(p^1, \bar{p})) - R^c(p^2, \delta(p^1, \bar{p})),
$$

where we now replace the first summand by its upper bound given by Eq. (25), and the second summand by its lower bound given by Eq. (28). Taking additionally into account that $R^c(p^1, \delta) \leq R^c(p^2, \delta)$, we conclude that

$$
R^c(p^1 \cup \bar{p}) - R^c(p^2 \cup \bar{p}) \leq R^c(p^1, \delta(p^1, \bar{p})) - R^c(p^2, \delta(p^1, \bar{p})),
$$

and see that the RHS of Eq. (31) equals the approximation error $\epsilon(p^1, \delta(p^1, \bar{p}))$ of $L_1$.

It follows that, given a specific backward path $\bar{p} = p^1[i:|p^1|]$, the actual cost difference between $L_1$ and any label $L_2$ that could potentially be dominated by $L_1$ is bounded from above by $\epsilon(p^1, \delta(p^1, p^1[i:|p^1|]))$. Applying this argument to all subpaths of $L_0 \in \eta_0^{\text{op}}$, and summing the resulting upper bounds creates an upper bound $\epsilon(p^1)$ for the approximation error of the objective function value of Problem 3 for route $p^1 = (0, \ldots, 0)$. It follows directly that Eq. (29) holds. \hfill \Box