A Branch-and-Price Algorithm for the Scheduling of Customer Visits in the Context of Multi-Period Service Territory Design

A Branch-and-Price Algorithm for the Scheduling of Customer Visits in the Context of Multi-Period Service Territory Design

Matthias Bender[a,*], Jörg Kalcsics[b], Stefan Nickel[a,c], Martin Pouls[a]

[a] *Department of Logistics and Supply Chain Optimization, Research Center for Information Technology (FZI), Haid-und-Neu-Str. 10–14, 76131 Karlsruhe, Germany*
[b] *School of Mathematics, University of Edinburgh, James Clerk Maxwell Building, The Kings Buildings, Peter Guthrie Tait Road, Edinburgh, EH9 3FD, Scotland*
[c] *Institute of Operations Research, Karlsruhe Institute of Technology (KIT), Kaiserstr. 89, 76133 Karlsruhe, Germany*

Abstract

A problem that arises in the context of multi-period service territory design is the scheduling of customer visits. In this problem, customer visits must be assigned to the days of the planning horizon subject to customer-specific requirements. We consider a highly relevant planning scenario of this problem and present an exact branch-and-price algorithm. We propose specialized acceleration techniques, particularly a fast pricing heuristic and techniques to reduce the symmetry inherent to the problem. Experiments on real-world data sets show that instances with up to 55 customers and a planning horizon of four weeks with five days per week can be solved to optimality in reasonable running times.

Keywords: transportation, multi-period service territory design, scheduling of customer visits, branch-and-price, acceleration techniques

1. Introduction

Classical service territory design problems consist of grouping customers into larger clusters, which are called territories or districts, such that some relevant planning criteria, e.g., compactness and balance, are met (Kalcsics, 2015). In each district, a service provider, e.g., a salesperson or service technician, is responsible for providing services at the customers' sites. In many cases, these services must be provided several times during a given planning horizon, which extends the classical problem to a multi-period setting. The resulting problem, the Multi-Period Service Territory Design Problem (MPSTDP), has recently been introduced by Bender et al. (2016). One of the subproblems that arises in the MPSTDP is the scheduling subproblem MPSTDP-S. In this subproblem, the districts are already given and customer visits need to be scheduled for each district individually. In this paper, we consider a highly relevant planning scenario of the MPSTDP-S, which we denote by MPSTDP-S*. It can formally be described as follows.

Given a planning horizon consisting of weeks $W = \{1, ..., |W|\}$ and days $D = \{1, ..., |D|\}$, and given the set of customers $B = \{1, ..., |B|\}$ of a district, the task is to assign customer visits to the weeks and days of the planning horizon. Each customer $b \in B$ must receive on-site service

*Corresponding author
Email addresses: mbender@fzi.de (Matthias Bender), joerg.kalcsics@ed.ac.uk (Jörg Kalcsics), stefan.nickel@kit.edu (Stefan Nickel), pouls@fzi.de (Martin Pouls)

by the service provider who is responsible for the district, and the service must be provided according to a customer-specific *week rhythm* $r_b \in \mathbb{N}^+$, which means that each customer must be visited regularly every r_b weeks, with the first service taking place in the first r_b weeks of the planning horizon. The number of weeks $|W|$ in the planning horizon is defined as the least common multiple of week rhythms $\{r_b\}_{b \in B}$. Each service of a customer requires a *service time* $t_b \in \mathbb{R}^+$. In order to balance the service provider's workload over the time periods of the planning horizon, the total service time on each day must be within the interval $\left[LB^{day}, UB^{day}\right]$, and the total service time in each week is limited to the interval $\left[LB^{week}, UB^{week}\right]$, where LB^{day}, UB^{day}, LB^{week} and UB^{week} denote appropriate minimum and maximum cumulative service times. The distance from customer b to customer b' is given by $c_{bb'} \in \mathbb{R}^+$, b, $b' \in B$. In order to reduce the travel time required for serving the customers, the objective is to schedule the customer visits in such a way that customers who are served on the same day or in the same week are geographically close to each other (see Bender et al., 2016, for the benefits of having a geographically compact service area in each week). More precisely, the objective is to minimize the sum of the distances between all customers that are served in the same time period (day or week) and a customer that is selected as the center for that time period. We adopt the terminology of Bender et al. and call the latter customers *day centers* and *week centers*. Note that a week center does not have to be served in the week it acts as the week center. This applies analogously to day centers. Furthermore, we denote the subsets of customers that are served on the same day or in the same week as *day clusters* and *week clusters*, respectively.

One might argue that, rather than striving for geographically compact day and week clusters, the daily route lengths should be optimized. However, since service visits might have to be rescheduled in day-to-day business (e.g., due to short-term customer requests), explicitly considering routing decisions is only of little use. Moreover, geographically compact clusters provide a high degree of flexibility to cope with short-term customer requests and other unexpected events in day-to-day operations. A detailed discussion on these aspects is provided in Bender et al. (2016).

Compared to the problem studied in Bender et al. (2016), the MPSTDP-S* contains the following assumptions. As opposed to Bender et al., we do not consider the possibility that a customer demands more than one service per week. We assume that there are no restrictions with respect to the days on which a customer can be served, whereas Bender et al. take into account customer-specific weekday patterns, which can be used to restrict service to particular combinations of weekdays. Moreover, we assume that always the same service time t_b is incurred for customer $b \in B$, while Bender et al. allow the specification of different service times for each visit of a customer. Due to these assumptions, the MPSTDP-S* fails to cover some of the applications of the more general MPSTDP-S, such as the filling of beverage or cigarette vending machines, in which several service visits per week might be required, or the selling of hairdressing equipment to hair salons, where individual rest days might have to be considered. Nevertheless, the assumptions hold for the majority of the real-word projects of our industry partner PTV Group[1], a commercial provider of districting and clustering software. Hence, we study a highly

[1] `http://www.ptvgroup.com`

relevant planning scenario of the problem introduced by Bender et al.

For a recent review of related problems, we refer the reader to Bender et al. (2016). Since the problem under study has been introduced only recently, no specialized exact solution methods have been proposed yet. However, we are aware of three papers that use column generation for similar problems. Mehrotra et al. (1998) study a single-period political districting problem and propose a branch-and-price based heuristic. The master problem corresponds to a set-partitioning problem with an additional constraint enforcing the required number of territories. The objective is to optimize compactness. Each column in the master problem represents a feasible territory, i.e., a territory which is contiguous and balanced in terms of population. Accordingly, the pricing problems correspond to two-sided knapsack problems with contiguity constraints. The authors incorporate some heuristic elements to increase computational efficiency, e.g., simplified contiguity constraints and distance-based variable fixing. de Fréminville et al. (2015) deal with a special single-period districting problem which they call the financial product districting problem. In this problem, customers must be partitioned into territories such that the expected customer-dependent cost price of a financial product is relatively the same for all customers that belong to the same territory. The authors formulate the master problem as a set-partitioning problem with additional side constraints. They aim at minimizing a weighted sum of the cost price variances within the territories. Each column corresponds to a feasible territory, which means that it must be contiguous and contain a given minimum number of customers. As the reduced cost of a column includes the cost price variance, the objective function of the pricing problem is nonlinear. The authors propose a greedy multi-start heuristic to solve the pricing problem and two heuristic procedures to determine an integer solution to the master problem. Mourgaya and Vanderbeck (2007) study a tactical variant of the period vehicle routing problem. The objective is to obtain geographically compact clusters for each time period and vehicle, and to balance workload between vehicles. In the master problem of their column generation reformulation, clusters, i.e., subsets of customers whose workload does not exceed a given upper bound, are selected for the time periods of the planning horizon. The authors propose a greedy insertion heuristic to solve the pricing problems, which correspond to quadratic knapsack problems. They alternately solve the linear programming (LP) relaxation of the restricted master problem and fix some of the variables to construct an integer solution. The problems studied by Mehrotra et al., de Fréminville et al., and Mourgaya and Vanderbeck differ from our problem in the following aspects: The problems tackled by Mehrotra et al. and de Fréminville et al. consider a single-period setting where each customer must be assigned to exactly one territory. Furthermore, contiguity is explicitly required in both problems. In contrast to this, we deal with a multi-period problem in which customers have to be assigned to multiple clusters, and we do not consider contiguity as a relevant planning criterion. Moreover, geographical compactness, which is the objective in our problem, is not taken into account by de Fréminville et al. In the problem studied by Mourgaya and Vanderbeck (2007), geographical compactness is relevant only with respect to one time scale (days), whereas we consider geographical compactness with respect to two time scales (days and weeks). Finally, in terms of solution methodology, the authors of the three papers propose heuristics, whereas we strive for the development of an exact method.

The main contributions of this paper are as follows:

- This paper is the first to present an exact branch-and-price algorithm for the scheduling task of the MPSTDP.
- We propose specially-tailored techniques to speed up the algorithm, such as a fast greedy heuristic to solve the pricing problems and techniques to reduce the symmetry inherent to the MPSTDP-S*.
- We show the effectiveness of our algorithm through extensive computational experiments on real-world instances and investigate the impact of individual algorithmic features. Instances with up to 55 customers can be solved to optimality in reasonable running times.
- Compared to solving the compact formulation of the MPSTDP-S* with a general purpose mixed integer programming (MIP) solver, we achieve an average reduction in running time of more than 98.1%.

The remainder of this paper is organized as follows. In Section 2, we present a compact linear integer programming (IP) model for the MPSTDP-S*. This model is reformulated in Section 3 into a master problem and several pricing problems, which serve as the basis for our branch-and-price algorithm. Moreover, we introduce some definitions and basic concepts about symmetry in this section. In Section 4, we present the details of our algorithm, including specialized techniques that aim at reducing running time. In Section 5, we report the results of extensive experiments on real-word test instances, which prove the effectiveness of the proposed algorithm. Finally, we provide a short conclusion and an outlook on future research on this topic in Section 6.

2. A Compact Formulation

In this section, we present a compact IP formulation for the MPSTDP-S*. It is based on the formulation of Bender et al. (2016), but adapts their formulation to the planning scenario studied in this paper. We introduce the following additional notation. Let $D(w) \subset D$ represent the days in week $w \in W$, and denote by $\lambda \in [0, 1]$ a user parameter to weight the importance of compact week clusters versus compact day clusters. 1 (0) means that the compactness of day clusters (week clusters) is irrelevant to the user, intermediate values represent trade-offs between the two extremes. Furthermore, define the following decision variables:

$$u_{ib}^{w} = \begin{cases} 1 & \text{if customer } b \in B \text{ is served in week } w \in W \text{ and assigned to week center} \\ & i \in B \\ 0 & \text{otherwise} \end{cases}$$

$$v_{ib}^{d} = \begin{cases} 1 & \text{if customer } b \in B \text{ is served on day } d \in D \text{ and assigned to day center} \\ & i \in B \\ 0 & \text{otherwise} \end{cases}$$

$$x_{b}^{w} = \begin{cases} 1 & \text{if customer } b \in B \text{ is the week center in week } w \in W \\ 0 & \text{otherwise} \end{cases}$$

$$y_b^d = \begin{cases} 1 & \text{if customer } b \in B \text{ is the day center on day } d \in D \\ 0 & \text{otherwise} \end{cases}$$

Using this notation, the MPSTDP-S* can be modeled as the following compact IP, which we denote by $(COMP)$:

$$(COMP) \quad \lambda \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} c_{ib}u_{ib}^w + (1-\lambda)\sum_{b\in B}\sum_{i\in B}\sum_{d\in D} c_{ib}v_{ib}^d \to \min \tag{1}$$

$$s.t. \quad \sum_{i\in B}\sum_{w\in W, w\le r_b} u_{ib}^w = 1 \qquad b \in B \tag{2}$$

$$\sum_{i\in B} u_{ib}^w = \sum_{i\in B} u_{ib}^{((w-1) \bmod r_b)+1} \qquad b \in B, w \in W, w > r_b \tag{3}$$

$$u_{ib}^w \le x_i^w \qquad b, i \in B, w \in W \tag{4}$$

$$\sum_{b\in B} x_b^w = 1 \qquad w \in W \tag{5}$$

$$\sum_{b\in B}\sum_{i\in B} t_b u_{ib}^w \ge LB^{week} \qquad w \in W \tag{6}$$

$$\sum_{b\in B}\sum_{i\in B} t_b u_{ib}^w \le UB^{week} \qquad w \in W \tag{7}$$

$$\sum_{i\in B}\sum_{d\in D(w)} v_{ib}^d = \sum_{i\in B} u_{ib}^w \qquad b \in B, w \in W \tag{8}$$

$$v_{ib}^d \le y_i^d \qquad b, i \in B, d \in D \tag{9}$$

$$\sum_{b\in B} y_b^d = 1 \qquad d \in D \tag{10}$$

$$\sum_{b\in B}\sum_{i\in B} t_b v_{ib}^d \ge LB^{day} \qquad d \in D \tag{11}$$

$$\sum_{b\in B}\sum_{i\in B} t_b v_{ib}^d \le UB^{day} \qquad d \in D \tag{12}$$

$$u_{ib}^w \in \{0,1\} \qquad b, i \in B, w \in W \tag{13}$$

$$v_{ib}^d \in \{0,1\} \qquad b, i \in B, d \in D \tag{14}$$

$$x_b^w \in \{0,1\} \qquad b \in B, w \in W \tag{15}$$

$$y_b^d \in \{0,1\} \qquad b \in B, d \in D \tag{16}$$

The Objective Function (1) optimizes the geographical compactness of the week and day clusters as a weighted sum. Constraints (2) ensure that the first service visit of each customer $b \in B$ is scheduled for the first r_b weeks, and Constraints (3) guarantee that the service recurs every r_b weeks. Constraints (4) make sure that assignments can only be made to customers that are selected as the week center of the respective week. Constraints (5) enforce that exactly one week center is selected for each week. The total service time of each week is guaranteed to be within the feasible time interval through Constraints (6) and (7). The weeks and days of the planning horizon are linked by Constraints (8). Constraints (9)–(12) impose restrictions at the level of days that are analogous to those defined by Constraints (4)–(7) for the level of weeks. Lastly, Constraints (13)–(16) define the binary decision variables.

Experiments have shown that model ($COMP$) can be solved only for very small problem instances to proven optimality in reasonable running time by a general purpose MIP solver (see the computational results in Section 5.5). This motivated the development of our branch-and-price algorithm.

3. A Column Generation Reformulation

In the following, we reformulate the compact model ($COMP$) of the previous section as a master problem and several pricing problems. Furthermore, we define what we understand by the term *symmetry* and show that the master problem exhibits a high degree of symmetry.

3.1. Master Problem

For the formulation of the master problem, we need to introduce some additional notation. Let the set S^{week} contain all feasible week clusters, i.e., all subsets of customers B that yield in total a service time within the interval $\left[LB^{week}, UB^{week}\right]$. Analogously, denote by S^{day} the set containing all feasible day clusters, i.e., all subsets of customers B that yield in total a service time in the interval $\left[LB^{day}, UB^{day}\right]$. Furthermore, denote by $S^w \subseteq S^{week}$ the clusters that can be selected for week $w \in W$ and by $S^d \subseteq S^{day}$ the clusters that can be selected for day $d \in D$. This notation is required since the restricted master problem in Section 4 may contain proper subsets $S^w \subset S^{week}$ and $S^d \subset S^{day}$ of all feasible clusters, and these subsets may vary from time period to time period. Moreover, let $c_s = \min_{i \in B} \sum_{b \in s} c_{ib}$ denote the compactness for each cluster $s \in S = S^{week} \cup S^{day}$. The lower the value of c_s, the more compact cluster $s \in S$ is. Let parameter a_{sb} be equal to 1 if cluster $s \in S$ contains customer $b \in B$, and 0 otherwise. Moreover, introduce the following binary decision variables:

$$\delta_s^w = \begin{cases} 1 & \text{if cluster } s \in S^w \text{ is selected for week } w \in W \\ 0 & \text{otherwise} \end{cases}$$

$$\delta_s^d = \begin{cases} 1 & \text{if cluster } s \in S^d \text{ is selected for day } d \in D \\ 0 & \text{otherwise} \end{cases}$$

Then, the master problem can be formulated as the following IP, which we denote by model (MP):

$$(MP) \qquad \lambda \sum_{w \in W} \sum_{s \in S^w} c_s \delta_s^w + (1-\lambda) \sum_{d \in D} \sum_{s \in S^d} c_s \delta_s^d \to \min \tag{17}$$

$$\text{s.t.} \qquad \sum_{s \in S^w} \delta_s^w = 1 \qquad w \in W \tag{18}$$

$$\sum_{w=1}^{r_b} \sum_{s \in S^w} a_{sb} \delta_s^w = 1 \qquad b \in B \tag{19}$$

$$\sum_{s \in S^w} a_{sb} \delta_s^w = \sum_{s \in S^w} a_{sb} \delta_s^{((w-1) \bmod r_b)+1} \qquad b \in B, w \in W, w > r_b \tag{20}$$

$$\sum_{s \in S^d} \delta_s^d = 1 \qquad d \in D \tag{21}$$

$$\sum_{d \in D(w)} \sum_{s \in S^d} a_{sb} \delta_s^d = \sum_{s \in S^w} a_{sb} \delta_s^w \qquad b \in B, w \in W \tag{22}$$

$$\delta_s^w \in \{0,1\} \qquad w \in W, s \in S^w \tag{23}$$

$$\delta_s^d \in \{0,1\} \qquad d \in D, s \in S^d \tag{24}$$

The Objective Function (17) optimizes the compactness. Constraints (18) make sure that exactly one cluster per week is selected. Constraints (19) guarantee that there is exactly one service visit of each customer $b \in B$ in the first r_b weeks, and Constraints (20) ensure that each customer $b \in B$ is served every r_b weeks. Constraints (21) make sure that exactly one cluster per day is selected. Weeks and days are linked by Constraints (22). Constraints (23) and (24) are the domain constraints.

3.2. Pricing Problems

Let π_0^w, π_1^b, $\pi_2^{b,w}$, π_3^d, and $\pi_4^{b,w}$ denote the dual variables for Constraints (18), (19), (20), (21), and (22), respectively. Then, the pricing problem can be formulated as follows:

$$\begin{aligned}
(PP) \qquad & \lambda \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} c_{ib}u_{ib}^w + (1-\lambda)\sum_{b\in B}\sum_{i\in B}\sum_{d\in D} c_{ib}v_{ib}^d \\
& - \sum_{w\in W}\pi_0^w - \sum_{b\in B}\pi_1^b \sum_{w=1}^{r_b}\sum_{i\in B} u_{ib}^w \\
& - \sum_{b\in B}\sum_{w\in W, w>r_b}\pi_2^{b,w}\sum_{i\in B}\left(u_{ib}^w - u_{ib}^{((w-1) \bmod r_b)+1}\right) \\
& - \sum_{d\in D}\pi_3^d - \sum_{b\in B}\sum_{w\in W}\pi_4^{b,w}\sum_{i\in B}\left(\sum_{d\in D(w)} v_{ib}^d - u_{ib}^w\right) \to \min \\
s.t. \qquad & (4)\text{–}(7), (9)\text{–}(16)
\end{aligned} \tag{25}$$

From the following observations it can be seen that model (MP) in combination with model (PP) indeed corresponds to model $(COMP)$ (for the sake of simplicity, we consider only the level of days): Day-related Constraints (9)–(12), (14), and (16) of model $(COMP)$ have been moved to the pricing problem (PP). The only day-related constraints that remain in model (MP) are Constraints (8), which link the level of days with the level of weeks. In model (MP), this is achieved by Constraints (22). Moreover, we ensure in model (MP) that exactly one day cluster per day is selected; observe that this is implicitly enforced in model $(COMP)$ since exactly one day center (and consequently exactly one day cluster) per day is selected. Since exactly one day cluster per day is generated in model (PP) and since each generated day cluster appears with a coefficient of one in Constraint (21) for the respective day in model (MP), dual values π_3^d are constants in model (PP). In contrast, dual values $\pi_4^{b,w}$ are included in the objective function of (PP) only if customer b is part of a generated day cluster for the respective week.

Model (PP) decomposes into $|W|$ independent pricing problems for the weeks and into $|D|$ independent pricing problems for the days, which gives us the following result.

Pricing Problems for the Weeks. Define parameters $\bar{c}^w_{ib}$, $b, i \in B$, $w \in W$, as follows:

$$\bar{c}^w_{ib} = \begin{cases} \lambda c_{ib} - \pi^b_1 + \sum\limits_{\hat{w}=1}^{\frac{|W|}{r_b}-1} \pi_2^{b,(w+\hat{w}r_b)} + \pi_4^{b,w} & \text{if } w \leq r_b \\ \lambda c_{ib} - \pi_2^{b,w} + \pi_4^{b,w} & \text{otherwise} \end{cases} \tag{26}$$

The pricing problem for week w can then be stated as the following IP (for better readability, the superscript w of the variables is omitted):

$$(PP^w) \qquad \sum_{b\in B}\sum_{i\in B} \bar{c}^w_{ib} u_{ib} - \pi_0^w \to \min \tag{27}$$

$$\text{s.t.} \qquad \sum_{b\in B} x_b = 1 \tag{28}$$

$$u_{ib} \leq x_i \qquad b, i \in B \tag{29}$$

$$\sum_{b\in B}\sum_{i\in B} t_b u_{ib} \geq LB^{week} \tag{30}$$

$$\sum_{b\in B}\sum_{i\in B} t_b u_{ib} \leq UB^{week} \tag{31}$$

$$u_{ib} \in \{0,1\} \qquad b, i \in B \tag{32}$$

$$x_b \in \{0,1\} \qquad b \in B \tag{33}$$

Pricing Problems for the Days. With $\phi(d) \in W$ representing the week that contains day $d \in D$ and parameter $\bar{c}^d_{ib} = (1-\lambda)c_{ib} - \pi_4^{b,\phi(d)}$, the pricing problem for day $d \in D$ can be formulated as follows (again, the superscript d of the variables is omitted):

$$(PP^d) \qquad \sum_{b\in B}\sum_{i\in B} \bar{c}^d_{ib} v_{ib} - \pi_3^d \to \min \tag{34}$$

$$\text{s.t.} \qquad \sum_{b\in B} y_b = 1 \tag{35}$$

$$v_{ib} \leq y_i \qquad b, i \in B \tag{36}$$

$$\sum_{b\in B}\sum_{i\in B} t_b v_{ib} \geq LB^{day} \tag{37}$$

$$\sum_{b\in B}\sum_{i\in B} t_b v_{ib} \leq UB^{day} \tag{38}$$

$$v_{ib} \in \{0,1\} \qquad b, i \in B \tag{39}$$

$$y_b \in \{0,1\} \qquad b \in B \tag{40}$$

3.3. Symmetry in model (MP)

As Bender et al. (2016) have already noted, problem MPSTDP-S contains a lot of symmetry. This applies also to model (MP) of the column generation reformulation. Symmetry can be present on the level of weeks and days. In the following, we formally define week and day symmetry, and derive a minimum amount of symmetry that can be found in any solution. Note that we use vectors in the remainder of this paper to specify week (day) clusters in chronological sequence. This means that the first component of such a vector represents the week (day) cluster

of the first week (day) of the planning horizon, the second component represents the week (day) cluster of the second week (day), and so on.

3.3.1. Week symmetry

By the term *week symmetry* we mean the symmetry that is due to the temporal rearrangement of a solution's week clusters. It is defined as follows.

Definition 1. *Given two feasible solutions with respective week clusters* $C = (C^1, ..., C^{|W|})$ *and* $\tilde{C} = (\tilde{C}^1, ..., \tilde{C}^{|W|})$*, the two solutions are said to be week-symmetric if there exists a permutation* $\sigma : W \mapsto W$ *with* $C^{\sigma(w)} = \tilde{C}^w$ *for each week* $w \in W$*.*

Next, we define what we mean by a *feasible week cluster permutation* for a solution and by a *maximally week-symmetry constrained solution.*

Definition 2. *Given the week clusters* $C = (C^1, ..., C^{|W|})$ *of a feasible solution, a permutation* $\sigma : W \mapsto W$ *is said to be a feasible week cluster permutation for that solution if in the week clusters* $(C^{\sigma(1)}, ..., C^{\sigma(|W|)})$ *each customer* $b \in B$ *is served every* r_b *weeks.*

Definition 3. *A solution consisting of week clusters* $C = (C^1, ..., C^{|W|})$ *is said to be maximally week-symmetry constrained with respect to the set of week rhythms* $R \subseteq \{r_b\}_{b \in B}$ *if each week cluster* C^w*,* $w \in W$*, contains for each week rhythm* $r \in R$ *a customer* $b \in B$ *with* $r_b = r$*.*

In the following, we state a special property of week cluster permutations that are feasible for maximally week-symmetry constrained solutions. This property will play an important role in the development of symmetry reduction techniques in Section 4.3.

Lemma 1. *If a week cluster permutation is feasible for a solution that is maximally week-symmetry constrained with respect to the set of week rhythms* R*, it is feasible for any other solution that consists only of customers* $b \in B$ *with* $r_b \in R$*.*

Proof. Consider a solution that is maximally week-symmetry constrained with respect to the set of week rhythms R. Clearly, removing a customer from the solution does not reduce the number of feasible week cluster permutations for that solution. Likewise, (feasibly) inserting an additional customer with $r_b \in R$ does not reduce the number of feasible week cluster permutations for that solution since each week cluster already contains a customer $b \in B$ with $r_b = r$ for each $r \in R$ and, hence, the newly inserted customer does not impose any additional restrictions. Since, starting from a maximally week-symmetry constrained solution, any other solution can be generated by inserting additional customers and removing present customers, a week cluster permutation that is feasible for a maximally week-symmetry constrained solution with respect to R is also feasible for any other solution that consists only of customers $b \in B$ with $r_b \in R$. □

From Lemma 1 we can derive a minimum amount of week symmetry inherent in any solution. Consider, for example, a planning horizon of $|W| = 4$ weeks, and suppose that $r_b \in R = \{1, 2, 4\}$ for each customer $b \in B$. The week cluster permutations shown in Table 1 are feasible for a maximally week-symmetry constrained solution with respect to R and, hence, also for any other solution in which the customers' week rhythms are restricted to the set R. This means that there are (at least) eight week-symmetric solutions to any solution consisting only of customers $b \in B$ with $r_b \in R$. Note that, when a solution is not maximally week-symmetry constrained, there might be even more week symmetry than given by Lemma 1.

Table 1: Feasible week cluster permutations for a maximally week-symmetry constrained solution with respect to $R = \{1, 2, 4\}$ and a planning horizon of $|W| = 4$ weeks. Example adopted from Bender et al. (2016).

Permutation no.	$\sigma(1)$	$\sigma(2)$	$\sigma(3)$	$\sigma(4)$	Permutation no.	$\sigma(1)$	$\sigma(2)$	$\sigma(3)$	$\sigma(4)$
1	1	2	3	4	5	3	4	1	2
2	1	4	3	2	6	3	2	1	4
3	2	3	4	1	7	4	1	2	3
4	2	1	4	3	8	4	3	2	1

3.3.2. Day symmetry

With $m = \frac{|D|}{|W|}$ denoting the number of days per week, we define *day symmetry* as follows.

Definition 4. *Given a week $w \in W$, two feasible solutions, and their respective day clusters $C = (C^{w,1}, ..., C^{w,m})$ and $\tilde{C} = (\tilde{C}^{w,1}, ..., \tilde{C}^{w,m})$ in week w, the two solutions are said to be day-symmetric with respect to week w if there exists a permutation $\sigma : \{1, ..., m\} \mapsto \{1, ..., m\}$ with $C^{w,\sigma(d)} = \tilde{C}^{w,d}$ for each weekday $d \in \{1, ..., m\}$ in week w.*

Since there are no restrictions with respect to the distribution of a customer's service visits to the days *within* a week, any rearrangement of the day clusters within a week is feasible. Consequently, there are $m!$ day-symmetric solutions for each week $w \in W$, which results in $(m!)^{|W|}$ day-symmetric solutions for the entire planning horizon. Consider again the example with a planning horizon of $|W| = 4$ weeks from Section 3.3.1 and suppose that each week consists of $m = 5$ days. Combining week and day symmetry, there are (at least) $8 \cdot (5!)^4$ symmetric solutions to any feasible solution. We will propose techniques to reduce this tremendous amount of symmetry in Section 4.3.

4. Branch-and-Price Algorithm

We propose a branch-and-price algorithm (see, e.g. Barnhart et al., 1998; Lübbecke and Desrosiers, 2005) to solve model (MP). A branch-and-price algorithm is a branch-and-bound algorithm for solving integer programs, in which the LP relaxation in each node of the branch-and-bound tree is solved using column generation. When the solution in a node is fractional and better than the current incumbent solution, branching is performed. In the following, we explain these steps in detail and present specialized techniques to reduce week and day symmetry. Furthermore, we present an extension of the algorithm which involves the generation of cutting planes to tighten the linear relaxation of model (MP). To the best of our knowledge, this is the first specially-tailored exact method for the scheduling task of the multi-period service territory design problem.

4.1. Column Generation

In each node of the branch-and-bound tree, we use column generation to solve the corresponding linear relaxation of model (MP), i.e., the linear relaxation of model (MP) extended by the branching decisions and, if applicable, by the cutting planes that are generated in the node. The basic idea of column generation is to work with a restricted master problem (RMP), which contains only a subset of the columns of model (MP) and to add new columns only if they

might improve the objective value. Two steps are performed iteratively. (1) The LP relaxation of the RMP is solved to obtain primal and dual solutions. (2) Pricing problems (PP^w) and (PP^d) are solved using the dual multipliers from step 1 to find negative reduced cost columns. If such columns exist, these columns are added to the RMP and the LP relaxation of the RMP is solved again; otherwise the current solution is an optimal solution to the LP relaxation of the RMP. An extensive introduction to column generation can be found in Desrosiers and Lübbecke (2005).

To obtain an initial set of feasible columns for the RMP, we solve the problem at hand with the location-allocation heuristic of Bender et al. (2016). Furthermore, we add one artificial binary variable with high objective function coefficient to each of Constraints (18), (19), and (21) to ensure feasibility when columns that would violate a branching decision are removed from the RMP.

To solve the pricing problems, we proceed as follows. As in Mehrotra et al. (1998), we break problems (PP^w) and (PP^d) down into smaller subproblems by fixing the week or day center $i \in B$. Fixing the week center i in problem (PP^w) yields the following IP, which we denote by (PP_i^w):

$$(PP_i^w) \qquad \sum_{b \in B} \bar{c}_{ib}^w u_b - \pi_0^w \to \min \tag{41}$$

$$\text{s.t.} \qquad \sum_{b \in B} t_b u_b \geq LB^{week} \tag{42}$$

$$\sum_{b \in B} t_b u_b \leq UB^{week} \tag{43}$$

$$u_b \in \{0,1\} \qquad b \in B \tag{44}$$

Analogously, we obtain problem (PP_i^d) when the day center i is fixed in problem (PP^d):

$$(PP_i^d) \qquad \sum_{b \in B} \bar{c}_{ib}^d v_b - \pi_3^d \to \min \tag{45}$$

$$\text{s.t.} \qquad \sum_{b \in B} t_b v_b \geq LB^{day} \tag{46}$$

$$\sum_{b \in B} t_b v_b \leq UB^{day} \tag{47}$$

$$v_b \in \{0,1\} \qquad b \in B \tag{48}$$

Note that we omitted the subscript i for the variables in both models.

As a result, we obtain $|B| \cdot |W|$ problems to generate promising week clusters and $|B| \cdot |D|$ problems to generate promising day clusters. The problems are similar to knapsack problems with two peculiarities: There can be negative profits for the items, and the weight of each knapsack must exceed a threshold value. We solve problems (PP_i^w) and (PP_i^d) for each center $i \in B$ and pass all columns with negative reduced costs to the RMP. The advantage of this procedure is that we can generate up to $|B| \cdot (|W| + |D|)$ negative reduced cost columns in a single pricing iteration. If we solved problems (PP^w) and (PP^d) instead, we could generate at most $|W| + |D|$ such columns per iteration.

We opted for a two-stage procedure to speed up the algorithm. First, we try to find promising

columns by means of a fast greedy heuristic. Only if the heuristic does not find any columns with negative reduced costs, we switch to an exact method to guarantee optimality of the overall algorithm.

The heuristic solves problem (PP_i^w) for a given center $i \in B$ and a given week $w \in W$ as illustrated by the pseudocode of Algorithm 1. Obviously, the heuristic has to take into account the fixations in the current node of the branch-and-bound tree. As will be explained in more detail in Section 4.2, a fixation may either enforce or forbid the assignment of a customer to a week or a day. For the remainder of this paper, we denote by $B^{avail}(d, N) \subseteq B$ and $B^{avail}(w, N) \subseteq B$ the subset of customers that are available for being scheduled to day $d \in D$ and week $w \in W$, respectively, in node N of the branch-and-bound tree. A customer is considered available for a day or a week in node N if there is no fixation in the node which prohibits the customer's assignment to that time period, e.g., through a fixation to a week which, in combination with the customer's week rhythm r_b, is not compatible with a visit in the considered time period. The basic idea of the heuristic is to first add all customers to the cluster that must be served in week w (i.e., weekly customers and customers fixed to week w), and then to add more customers in non-decreasing order of parameters $\bar{c}_{ib}^w$. Irrespective of whether the exact or the heuristic pricing method has been used, we set the cluster center to the customer $j \in B$ that minimizes the sum of the distances to all customers in the cluster. Thus, the reduced cost of the final cluster $s \subseteq B$ for week w can be computed as

$$\min_{j \in B} \sum_{b \in s} \bar{c}_{jb}^w - \pi_0^w. \tag{49}$$

If this value is negative, the cluster is passed to the RMP. The time complexity of Algorithm 1 is dominated by the calculation of the optimal center in step 13, i.e., its complexity is $\mathcal{O}(|B|^2)$.

Algorithm 1 Heuristic to solve problem (PP_i^w) for given center $i \in B$ and given week $w \in W$

Input: Center $i \in B$; week $w \in W$; fixations in node N
Output: $s \subseteq B$: A cluster with negative reduced cost if such a cluster can be found

1: determine $B^{avail}(w, N)$ and sort it in non-decreasing order of $\bar{c}_{ib}^w$
2: $s \leftarrow \emptyset$
3: **for** $b \in B^{avail}(w, N)$ **do**
4: **if** (b is fixed to week w) or ($r_b = 1$) **then**
5: $s \leftarrow s \cup \{b\}$
6: **end if**
7: **end for**
8: **for** $b \in B^{avail}(w, N) \setminus s$ **do**
9: **if** $(\sum_{\hat{b} \in s} t_{\hat{b}} + t_b \leq UB^{week})$ and $(\bar{c}_{ib}^w < 0$ or $\sum_{\hat{b} \in s} t_{\hat{b}} < LB^{week})$ **then**
10: $s \leftarrow s \cup \{b\}$
11: **end if**
12: **end for**
13: **if** $(\sum_{\hat{b} \in s} t_{\hat{b}} \geq LB^{week})$ and $(\min_{j \in B} \sum_{b \in s} \bar{c}_{jb}^w - \pi_0^w < 0)$ **then**
14: **return** s
15: **end if**

The heuristic to solve pricing problem (PP_i^d) for a given center $i \in B$ and a given day $d \in D$

is analogous to Algorithm 1, therefore, we refrain from giving an explicit explanation. But we want to point out one peculiarity. To this end, we introduce the concept of *day groups*:

Definition 5. *A day group with respect to node N of the search tree is an equivalence class based on the following equivalence relation on the set of days D: Days $d_1 \in D$ and $d_2 \in D$ are equivalent if and only if they are in the same week, i.e., $\phi(d_1) = \phi(d_2)$, and have identical sets of available customers, i.e., $B^{avail}(d_1, N) = B^{avail}(d_2, N)$.*

Note that $\phi(d_1) = \phi(d_2)$ implies $\bar{c}_{ib}^{d_1} = \bar{c}_{ib}^{d_2}$ for all $b \in B$ and $i \in B$. Hence, it follows from this definition that, in a certain node of the search tree, pricing problems $(PP_i^{d_1})$ and $(PP_i^{d_2})$ have the same optimal solutions for any two days d_1 and d_2 that are in the same day group. The reduced costs of the resulting day clusters differ only by the difference in the values of constants π_3^d, $d \in \{d_1, d_2\}$. Thus, to save computation time, we solve problems (PP_i^d) only for one day of each day group explicitly. As our heuristic pricing method also yields the same solutions for all days of a day group, we proceed the same way in heuristic pricing.

4.2. Branching

When we obtain a fractional solution in a node of the branch-and-bound tree, branching is necessary. As other authors have already noted (e.g., Savelsbergh, 1997; Savelsbergh and Sol, 1998), branching on the variables of the master problem changes the structure of the pricing problems and makes them harder to solve as one needs to take care that forbidden columns are not re-generated in the pricing problems. Therefore, our branching is based on the compact formulation $(COMP)$, i.e., we branch on the assignment of customers to time periods. These assignments can easily be derived from the solution to the LP relaxation of the RMP. The assignment of customer $b \in B$ to week $w \in W$ is calculated as $u_b^w = \sum_{s \in S^w} a_{sb} \delta_s^w$. Analogously, the assignment of customer $b \in B$ to day $d \in D$ is given by $v_b^d = \sum_{s \in S^d} a_{sb} \delta_s^d$.

Branching is performed hierarchically. As long as there are fractional assignments of customers to weeks, we branch on the week assignments u_b^w. Only if all customers are unambiguously assigned to weeks, we branch on the assignments of customers to days v_b^d. In both cases, we generate two child nodes, with one node forcing the corresponding assignment to take on a value of one and the other forcing it to zero. The fixations must be taken into account in the RMP of the newly generated nodes and in the corresponding pricing problems. In the RMP, we take care of the fixations by removing all clusters from the model that would violate a fixation. In the exact pricing method, we simply adopt the fixations into the IP model, and in the pricing heuristic we consider all fixations in the sets $B^{avail}(d, N)$ and $B^{avail}(w, N)$ as explained in Section 4.1.

We implement two different rules to decide which assignments to branch on. We illustrate this in the following using the example of week assignments, but the procedure is analogous for day assignments. Our first branching rule is largest split (LS) branching. In LS branching, we select a fractional customer-week assignment with maximum value, i.e., we select

$$\langle b^\star, w^\star \rangle \in \arg\max_{\langle b,w \rangle} \{u_b^w \mid u_b^w \notin \{0, 1\}, w \leq r_b\}. \tag{50}$$

Since $u_b^w = u_b^{\hat{w}}$ if $w \bmod r_b = \hat{w} \bmod r_b$, we consider only the first r_b weeks for each customer $b \in B$.

Our second branching rule is pseudocost (PSD) branching. This rule is inspired by the works of Achterberg et al. (2005) and Linderoth and Savelsbergh (1999). The basic idea is to estimate the increase of the objective value when a fractional assignment is forced to take on an integer value compared to the objective value of the parent node. Branching priority is given to assignments that are expected to lead to a large deterioration in the objective value. Thus, this rule aims at a quickly rising lower bound.

Consider a particular node N in the branch-and-bound tree. Denote by f^N its objective value, and by f^+ and f^- the objective values of the two child nodes when the branching variable u_b^w is forced to one and zero, respectively. Then, the increase in the objective value per unit change in the branching variable can be calculated as follows:

$$\Delta_{b,w}^+ = \frac{f^+ - f^N}{1 - u_b^w}, \tag{51}$$

$$\Delta_{b,w}^- = \frac{f^- - f^N}{u_b^w}. \tag{52}$$

We could now calculate scores for each possible branching variable u_b^w. But our preliminary tests have shown that the number of branching decisions is not large enough to derive meaningful scores on such a fine-grained scale. Therefore, we do not calculate customer- and week-specific scores, but aggregate the scores per customer. With $\Delta_{b,d}^+$ being the counterpart of $\Delta_{b,w}^+$ for day assignments, we denote by θ_b^+ the sum over all $\Delta_{b,w}^+$ and $\Delta_{b,d}^+$ for all past upward branching decisions on a week or day assignment of customer $b \in B$. θ_b^- is defined analogously for the case of downward branching. Moreover, we denote by n_b^+ and n_b^- the number of upward and downward branching decisions, respectively, on a week or day assignment of customer b. Then, two score values $Score_b^+$ and $Score_b^-$ are calculated for each customer b. They represent the average relative increase in the objective value for upward and downward branching on a week or day assignment of the customer.

$$Score_b^+ = \frac{\theta_b^+}{n_b^+} \tag{53}$$

$$Score_b^- = \frac{\theta_b^-}{n_b^-} \tag{54}$$

Finally, the score for each customer-week assignment is calculated as

$$Score_b^w = (1 - u_b^w) \cdot Score_b^+ + u_b^w \cdot Score_b^-. \tag{55}$$

As the customer-week assignment to be branched on we select a fractional assignment with maximum score, i.e., we select

$$\langle b^\star, w^\star \rangle \in \arg\max_{\langle b,w \rangle} \{Score_b^w \mid u_b^w \notin \{0,1\}, w \leq r_b\}. \tag{56}$$

We always use LS branching for the first n_{min} branching decisions to initialize the scores. If, after n_{min} iterations, either all $Score_b^+$ values are uninitialized or all $Score_b^-$ values are uninitialized, we perform additional iterations with LS branching until we obtain at least one initialized

$Score_b^+$ value and one initialized $Score_b^-$ value. Afterwards, we switch to PSD branching. During the course of the algorithm, uninitialized scores $Score_b^+$ and $Score_b^-$ are set to the average of the respective initialized scores, i.e.,

$$Score_b^+ = \frac{\sum_{\hat{b} \in B^+} Score_{\hat{b}}^+}{|B^+|}, \tag{57}$$

$$Score_b^- = \frac{\sum_{\hat{b} \in B^-} Score_{\hat{b}}^-}{|B^-|}, \tag{58}$$

where B^+ and B^- denote the set of customers with initialized values of $Score_b^+$ and $Score_b^-$, respectively. This way of initializing the scores seems to be more plausible than other alternatives, e.g., taking the maximum or minimum values.

Irrespective of the selected branching rule, we adopt the idea of early branching (see, e.g., Desaulniers et al., 2002) to accelerate our algorithm. The potential benefit of early branching becomes obvious through the following observations: The exact solution of the pricing problems (PP_i^w) and (PP_i^d) is computationally expensive. Moreover, preliminary tests showed that, in many cases, exact pricing does not find any negative reduced cost columns, but is executed only to prove optimality. Even if negative reduced cost columns are found, their impact on the objective value of the node is usually fairly small. Therefore, we skip the exact pricing step under certain conditions. More precisely, when the pricing heuristic does not find any more negative reduced cost columns, we skip exact pricing if the current solution to the LP relaxation of the RMP is fractional and if its objective value is better than that of the current incumbent solution. As a consequence, exact pricing is called less often. Note that when early branching is applied, the objective value of a node might be better than that of its parent node, i.e., the objective value does not provide a valid lower bound any more. Hence, before a node can be pruned, re-optimization with our exact pricing method must be performed. A node is pruned only if the objective value after re-optimization is not better than that of the incumbent solution.

We use a best-first strategy to explore the branch-and-bound tree, i.e., we always select the node with the best initial objective value, which is inherited from the parent node, to be processed next.

4.3. Symmetry Reduction

As illustrated in Section 3.3, model (MP) contains a lot of symmetry. Thus, efficient symmetry handling is crucial for the design of a successful branch-and-price algorithm. In this section, we propose two techniques to reduce symmetry. In the first technique, we fix a single customer a priori to a particular day of the planning horizon. In the second, more sophisticated technique, we introduce additional variable fixations during the course of the algorithm and prune certain subtrees if we can guarantee that they contain only solutions that are symmetric to solutions in other parts of the search tree. In the following, we explain the techniques in detail.

4.3.1. Fixing a Reference Customer

A simple, yet effective way to eliminate some of the symmetry inherent to the MPSTDP-S* is to fix one service of a particular customer, which is called the *reference customer*, to a particular

day of the planning horizon. This approach is similar to the idea presented by Mourgaya and Vanderbeck (2007) in the context of the periodic vehicle routing problem. In the following, we prove that such a fixation can be done without losing optimality.

Lemma 2. *A right-shift of week clusters, defined as the week cluster permutation $\sigma : W \mapsto W$ with $\sigma(1) = |W|$ and $\sigma(w) = w - 1$ for each $w > 1$, is a feasible week cluster permutation for any feasible solution.*

Proof. Consider the week clusters $C = (C^1, ..., C^{|W|})$ of a feasible solution. Since the solution is feasible, each customer $b \in B$ of the solution is served regularly every r_b weeks with the first service in the first r_b weeks. This means that there exists for each customer $b \in B$ an $n_b \in \{0, ..., r_b - 1\}$ such that each C^w, $w \in W$, contains customer b if and only if $w \bmod r_b = n_b$. Let $\tilde{C} = (\tilde{C}^1, ..., \tilde{C}^{|W|})$ denote the week clusters obtained by a right-shift of C. Since $|W|$ is the least common multiple of the week rhythms $\{r_b\}_{b \in B}$ and, hence, $|W| \bmod r_b = 0$ for each b, $\tilde{C}^w$ contains b if and only if $w \bmod r_b = (n_b + 1) \bmod r_b = \tilde{n}_b$. Thus, a right-shift of the week clusters of a feasible solution yields a feasible solution and, hence, is a feasible week cluster permutation. □

Proposition 1. *For any arbitrary customer $b^* \in B$ and any day $d^* \in D$ of the planning horizon, there exists an optimal solution with customer b^* being scheduled to day d^*.*

Proof. Given any optimal solution, one can, according to Lemma 2, obtain a feasible week-symmetric solution in which customer b^* is served in week $\phi(d^*)$ by performing an appropriate number of right-shifts of the week clusters. Afterwards, as there are no restrictions on the re-orderings of the day clusters within a week, a day-symmetric solution with respect to week $\phi(d^*)$ can be obtained in which customer b^* is served on day d^*. Two week- or day-symmetric solutions consist of the same week and day clusters (merely arranged in a different order) and, hence, have the same objective value. Therefore, the resulting solution is optimal. □

Obviously, the extent of symmetry reduction that can be achieved by such a fixation depends on the selected reference customer. The reduction of week symmetry depends on the customer's week rhythm. The greater the week rhythm r_b of a customer $b \in B$, the more possibilities exist to assign the customer to the weeks of the planning horizon. Hence, to achieve maximal week symmetry reduction, we select a reference customer $b^* \in B$ with $r_{b^*} = max_{b \in B} r_b$. Then, we fix customer b^* to a day $d^* \in D(w^*)$ with $w^* \leq r_{b^*}$ in the root node of the branch-and-bound tree. Through this simple technique we can already reduce symmetry by factor $m \cdot r_{b^*}$. Clearly, if $r_{b^*} < |W|$, customer b^* can additionally be fixed to an arbitrary day in each of weeks $w \in \{w^* + r_{b^*}, w^* + 2r_{b^*}, ..., |W| + w^* - r_{b^*}\}$.

4.3.2. Symmetry-reduced Branching

In this section, we introduce a technique which we call symmetry-reduced branching. It was developed by Pouls (2016) and is an enhancement of the branching scheme introduced in Section 4.2 with the aim of reducing both week and day symmetry.

Reduction of Week Symmetry. Suppose that $u_{b^*}^{w^*}$ is the week assignment that is selected to be branched on in a particular node N of the branch-and-bound tree. Recall that, in standard branching, we always create two child nodes N^+ and N^- of N. We fix $u_{b^*}^{w^*} = 1$ in node N^+, and $u_{b^*}^{w^*} = 0$ in node N^-. The idea of symmetry-reduced branching is to add additional week fixations to node N^- if we can guarantee that, to any solution that becomes infeasible in node N^- by such an additional fixation, there is a week-symmetric solution in the other branch.

We denote by $\mathcal{S}$ the set of feasible week cluster permutations for a solution that is maximally week-symmetry constrained with respect to the set of week rhythms $R = \{r_b \mid b \in B, 1 < r_b < |W|\}$. Note that customers $b \in B$ with week rhythm $r_b = 1$ or $r_b = |W|$ do not have to be considered since they do not restrict the feasibility of the permutations. By fixing the week or day assignments of customers, as done in the nodes of the branch-and-bound tree, permutations from the set $\mathcal{S}$ are gradually rendered infeasible in the course of the algorithm. We denote by $\mathcal{S}(N) = \{\sigma \in \mathcal{S} \mid \sigma \text{ is feasible with respect to all fixations present in node } N\}$. Week symmetry can be reduced as follows.

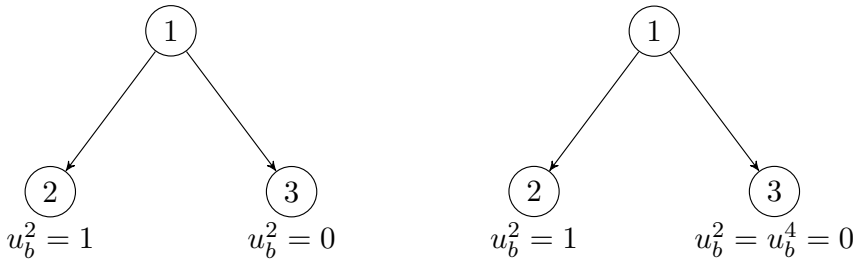
Proposition 2. *If there exists a permutation $\sigma \in \mathcal{S}(N)$ in node N of the search tree and a week $\hat{w} \in W$ with $\hat{w} \neq w^*$, $\hat{w} \leq r_{b^*}$ and $((\sigma(\hat{w}) - 1) \bmod r_{b^*}) + 1 = w^*$, the additional fixation $u_{b^*}^{\hat{w}} = 0$ can be added to node N^- without losing optimality.*

Proof. If the condition above is fulfilled, then there exists a feasible week cluster permutation which maps the first service of customer b^* from week $\hat{w}$ to week w^*. In this case, we can guarantee to find a solution in the subtree of node N^+ that is week-symmetric to any solution in the subtree of node N^- in which customer b^* is served in week $\hat{w}$. Hence, we cannot forfeit optimality if we introduce the additional fixation $u_{b^*}^{\hat{w}} = 0$ to node N^-. □

If, after the insertion of additional fixations, there are no more feasible week assignments left for customer b^* in node N^-, we immediately prune node N^-.

Consider the following example. Suppose again that the planning horizon consists of $|W| = 4$ weeks and that the week rhythms $r_b \in R = \{1, 2, 4\}$ for all customers $b \in B$. As we can see in Table 1, there are at least eight feasible permutations of the week clusters in this setting. Figure 1 illustrates the difference between standard branching and symmetry-reduced branching. We assume that a reference customer $b^* \in B$ with week rhythm $r_{b^*} = 4$ has been fixed to the first day and, hence, also to the first week of the planning horizon. This reduces the feasible permutations to permutations no. 1 and 2 from Table 1, i.e., to $(1, 2, 3, 4)$ and $(1, 4, 3, 2)$. Moreover, we assume that no other fixations exist in node 1. Suppose that we branch on the week assignment u_b^2 in node 1 and that $r_b = 4$. In standard branching, this would lead to two child nodes, with node 2 fixing the assignment to one, and node 3 fixing it to zero. But for each solution in which the customer is assigned to week $w = 4$ in node 3, permutation no. 2 gives us a week-symmetric solution in which the customer is served in week $w = 2$, which is identical to the situation in node 2. Hence, in symmetry-reduced branching, we add the additional fixation $u_b^4 = 0$ to node 3.

Reduction of Day Symmetry. Suppose that we branch on the day assignment $v_{b^*}^{d^*}$ in node N of the search tree. As in the branching on week assignments, two child nodes N^+ and N^- are generated in standard branching with fixations $v_{b^*}^{d^*} = 1$ in node N^+ and $v_{b^*}^{d^*} = 0$ in node N^-. In symmetry-reduced branching, we add, again, additional fixations to node N^- in order to reduce symmetry.



(a) Standard branching (b) Symmetry-reduced branching

Figure 1: Comparison of standard branching and symmetry-reduced branching for an exemplary week assignment

Recall that there are no restrictions with respect to the assignment of customers to days *within* a week. Hence, as long as there are no day fixations in a particular week, the day clusters of the week can be arbitrarily rearranged. But even when some day fixations have already been introduced, day symmetry might still be present. Based on the concept of day groups (see Definition 5), day symmetry can be reduced as follows.

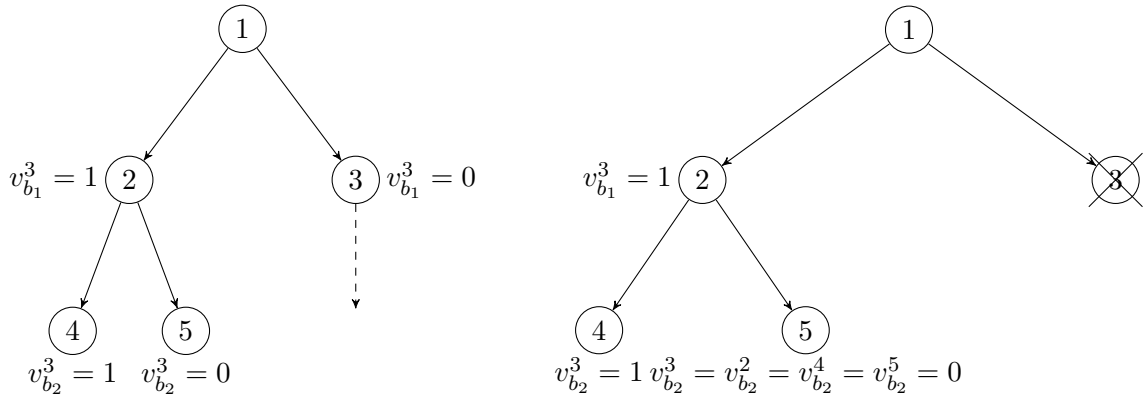
Proposition 3. *Let G be a day group in week $\phi(d^*)$, i.e., $G \subseteq D(\phi(d^*))$, with respect to node N of the search tree. Then, the following fixations can be added to node N^- without losing optimality. If G does not contain the branching day d^*, additional fixations $v_{b^*}^d = 0$ can be added for all days $d \in G$ except one. Otherwise, these fixations can be added for each day $d \in G$, $d \neq d^*$.*

Proof. Since all days $d \in G$ have the same set of available customers $B^{avail}(d, N)$, all rearrangements of the corresponding day clusters yield day-symmetric solutions. If G does not contain d^*, we can therefore forbid the assignment of customer b^* to any but one of the days of day group G in node N^-. If G contains d^*, we can forbid any solution in which b^* is served on a day $d \in G$ in node N^- since node N^+ contains a day-symmetric solution. Hence, optimality is guaranteed in both cases. □

Consequently, we check for each day group in week $\phi(d^*)$ if additional fixations can be introduced. If the additional fixations leave no feasible day assignments for customer b^* and week $\phi(d^*)$ in node N^-, we immediately prune node N^-. There is, however, one peculiarity. It might occur that we obtain an integer week assignment for a customer, although the customer is not fixed to a particular week. When we branch on the day assignment of such a customer, the customer's week assignment is implicitly fixed to week $\phi(d^*)$ in node N^+. If, at the same time, the available customers $B^{avail}(d, N)$ are identical for each day $d \in D(\phi(d^*))$, we prune node N^-, and, hence, discard the possibility of the customer being assigned to a different week. Therefore, if this situation occurs, we generate an additional child node, in which we force the customer to be scheduled to a different week, i.e., in which we set $u_{b^*}^{\phi(d^*)} = 0$.

Consider the example shown in Figure 2 and assume that there are $m = 5$ days per week. Further, assume that we branch on day assignment $v_{b_1}^3$ in node 1 and that the available customers $B^{avail}(d, N)$ are the same for each day $d \in D(\phi(3))$, i.e., there exists only one day group $G_1 = \{1, 2, 3, 4, 5\}$ consisting of all days of the week. In standard branching, this would again lead to the creation of two child nodes, one with $v_{b_1}^3 = 1$ and the

other with $v^3_{b_1} = 0$. In symmetry-reduced branching, we would add the additional fixations $v^1_{b_1} = v^2_{b_1} = v^4_{b_1} = v^5_{b_1} = 0$ to node 3, which would leave no feasible day assignments left for customer b_1 in week $\phi(3)$. Hence, node 3 can immediately be pruned. Note that for this example we assume that customer b_1 has previously been fixed to week $\phi(3)$ such that we do not have to create an additional child node which allows the assignment to a different week. Suppose that the next branching is performed on the day assignment $v^3_{b_2}$ in node 2. Due to the fixation $v^3_{b_1} = 1$ we now have the two day groups $G_2 = \{3\}$ and $G_3 = \{1, 2, 4, 5\}$. Since the branching day 3 is not part of day group G_3, we can forbid the assignment of customer b_2 to any of the days of day group G_3 except one. Hence, we add the additional fixations $v^2_{b_2} = v^4_{b_2} = v^5_{b_2} = 0$ to node 5.



(a) Standard branching

(b) Symmetry-reduced branching

Figure 2: Comparison of standard branching and symmetry-reduced branching for exemplary day assignments

4.4. Cut Generation

In an attempt to strengthen the LP relaxation of the RMP, we experimented with an extension of the proposed algorithm by the incorporation of cutting planes. After the column generation phase, we look for valid inequalities that are violated by the current solution to the LP relaxation of the RMP and add them to the RMP. Note that model (MP) has set-partitioning-like components, e.g., Constraints (19) define a set-partitioning polytope. Hence, valid inequalities for the set-packing and set-partitioning polytope, such as the well-known clique inequalities and odd-hole inequalities (see, e.g., Padberg, 1973), could be used to strengthen the LP relaxation of the RMP. For the week clusters in the RMP, we could formulate clique or odd-hole inequalities based on a conflict graph derived from Constraints (18)–(20) (or a subset of them). However, adding these inequalities to the RMP significantly changes the structure of the pricing problems. While our pricing heuristic could easily be adapted to consider these changes, solving the pricing problems to optimality would become much more complex. The difficulty is to determine whether a column participates in a certain clique or odd-hole inequality of the RMP, and, hence, whether the associated reduced costs must be considered in the pricing problem. Preliminary tests confirmed that the solution of models (PP^w_i) and (PP^d_i), extended to consider the reduced costs of clique cuts, becomes computationally too expensive. Therefore, we opted to use subset-row (SR) inequalities (Jepsen et al., 2008). They were proposed for a

set-partitioning formulation of the vehicle routing problem with time windows and mitigate to some extent the above mentioned disadvantage of clique and odd-hole inequalities.

We define an SR inequality q on a subset of Constraints (19), and, since each constraint corresponds to a customer, also on a subset $B_q \subseteq B$ of customers. An SR inequality can be stated as

$$\sum_{w \in W} \sum_{s \in S^w} \left\lfloor \frac{1}{k} \sum_{b \in B_q | w \leq r_b} a_{sb} \right\rfloor \delta_s^w \leq \left\lfloor \frac{|B_q|}{k} \right\rfloor, \tag{59}$$

where k is a parameter with $0 < k \leq |B_q|$. It can be interpreted as follows. For every k customers $b \in B_q$ that are contained in a week cluster s of a week w with $w \leq r_b$, the coefficient of the cluster on the left-hand side of the inequality increases by one. Since each customer b must be served exactly once in the first r_b weeks of the planning horizon, at most $\left\lfloor \frac{|B_q|}{k} \right\rfloor$ such clusters may be selected in an integer solution.

For the separation of SR inequalities, we set parameter k to a fixed value and restrict ourselves to subsets B_q of cardinality n_{sr}. We check for each subset B_q with $|B_q| = n_{sr}$ if Inequality (59) is satisfied in the current solution to the LP relaxation of the RMP and add all violated inequalities to the RMP.

Integrating SR inequalities into pricing problems (PP_i^w) yields the following result:

$$(PP_i^w\text{–}SR) \qquad \sum_{b \in B} \bar{c}_{ib}^w u_b - \pi_0^w - \sum_{q \in Q} \pi_5^q z_q \to \min \tag{60}$$

$$\text{s.t.} \qquad \sum_{b \in B} t_b u_b \geq LB^{week} \tag{61}$$

$$\sum_{b \in B} t_b u_b \leq UB^{week} \tag{62}$$

$$z_q \geq \left(\frac{1}{k} \sum_{b \in B_q | w \leq r_b} u_b \right) - 1 + \epsilon \qquad q \in Q \tag{63}$$

$$u_b \in \{0, 1\} \qquad b \in B \tag{64}$$

$$z_q \in \mathbb{N}_0 \qquad q \in Q \tag{65}$$

In model (PP_i^w–SR), Q denotes the set of SR cuts contained in the RMP, π_5^q denotes the dual variable for SR cut $q \in Q$, and ϵ represents a parameter with value slightly greater than zero. Assuming that $k \in \mathbb{N}^+$, ϵ must be set to a value $0 < \epsilon \leq \frac{1}{k}$. This makes sure that Constraints (63) in conjunction with the integrality requirements on variables z_q as defined in Constraints (65) mimic the floor function of the left-hand side of Inequality (59): For each SR cut $q \in Q$, variable z_q is increased by one for every k customers $b \in B_q$ with $w \leq r_b$ which are contained in the week cluster that is generated in the pricing problem. Note that it is not necessary to add constraints which define an upper bound for z_q since all π_5^q are nonpositive and, therefore, z_q implicitly takes on the smallest feasible value.

We adapt our pricing heuristic to reflect the modification of the pricing problems. The mechanism to generate new week clusters remains the same as described in Section 4.1, but we need to consider the values of π_5^q in the calculation of the reduced cost of the final cluster. For

cluster s and week w the reduced cost is calculated as follows:

$$\min_{j \in B} \sum_{b \in s} \bar{c}_{jb}^{w} - \pi_0^w - \sum_{q \in Q} \pi_5^q \left\lfloor \frac{1}{k} \sum_{b \in B_q | w \leq r_b} a_{sb} \right\rfloor . \tag{66}$$

We pass the corresponding column to the RMP only if this value is smaller than zero.

In addition to SR cuts for week clusters, we also generate SR cuts for day clusters. Recall that Constraints (19) enforce that each customer $b \in B$ must be served exactly once in the first r_b weeks of the planning horizon. From this we can derive the following constraints on the level of day clusters:

$$\sum_{w=1}^{r_b} \sum_{d \in D(w)} \sum_{s \in S^d} a_{sb} \delta_s^d = 1 \qquad\qquad b \in B \tag{67}$$

Based on these constraints, we formulate SR cuts for day clusters. The formulation, separation, and pricing is analogous to the SR cuts for week clusters. Therefore, we do not give any additional explanations. We found out in preliminary tests that the impact of cutting planes on the optimal objective value of the LP relaxation of the RMP declines rapidly with the number of performed cutting phases. Since the first cutting phase yields by far the largest impact on the objective value, we decided to execute the cutting phase only once in each node of the branch-and-bound tree, namely after the first column generation phase. In all other cases, we proceed from the column generation phase directly to the branching phase. Moreover, we decided that a node does not inherit the cuts from its parent node.

5. Computational Evaluation

In the following, we evaluate our algorithm on real-world test instances provided by PTV. The test set comprises 16 service territories of a German manufacturer of paints and coatings. The week rhythms r_b of the customers are from the set $\{1, 2, 4\}$. The total number of visits per territory ranges from 71 to 107, the time to serve a customer, t_b, ranges from ten to 330 minutes. The planning horizon consists of $|W| = 4$ weeks and $m = 5$ days per week. A detailed overview of the test instances is given in Table 2.

For all tests, we weight the compactness of week clusters with $\lambda = \frac{1}{3}$ and the compactness of day clusters with $1 - \lambda = \frac{2}{3}$. With $T = \sum_{b \in B} t_b \cdot \frac{|W|}{r_b}$ denoting the total service time over all customers, we limit the total service time of each week to the interval $\left[LB^{week}, UB^{week}\right] = \left[0.9 \cdot \frac{T}{|W|}, 1.1 \cdot \frac{T}{|W|}\right]$ and the total service time of each day to the interval $\left[LB^{day}, UB^{day}\right] = \left[0.8 \cdot \frac{T}{|D|}, 1.2 \cdot \frac{T}{|D|}\right]$. To initialize the scores in the case of PSD branching, we use LS branching for at least the first $n_{min} = 5$ branching decisions. For the separation of SR inequalities, we set parameters $k = 2$ and $n_{sr} = 3$ since this configuration yielded the best results in our preliminary tests. The algorithm was coded in Java. All tests are performed under Ubuntu 16 on a machine with an Intel Xeon E5-2650 v2 CPU at 2.6 GHz and 128 GB of RAM. We use Gurobi 7.0.1[2] to solve the LP relaxation of the RMP and the IPs in the exact pricing step.

[2] `http://www.gurobi.com`

Table 2: Overview of test instances.

Instance no.	1	2	3	4	5	6	7	8
Number of customers	31	26	32	25	35	55	36	33
Week rhythms	1, 2, 4	1, 4	1, 2, 4	1, 2, 4	1, 2, 4	1, 2, 4	1, 4	1, 2, 4
Number of visits	80	74	76	71	84	106	72	78
Instance no.	9	10	11	12	13	14	15	16
Number of customers	33	31	32	50	39	42	37	52
Week rhythms	1, 2, 4	1, 2, 4	1, 2, 4	1, 2, 4	1, 2, 4	1, 2, 4	1, 2, 4	1, 2, 4
Number of visits	88	89	88	107	86	94	96	88

We analyze the impact of different features of our algorithm on its running time. In particular, we evaluate the impact of the proposed symmetry reduction techniques, we compare the two branching rules LS and PSD, and we analyze the effect of early branching and cutting planes. If not stated otherwise, the algorithm is configured as follows:

- Full symmetry reduction is applied, i.e., the fixation of a reference customer is combined with symmetry-reduced branching.
- PSD branching in combination with the presented early branching strategy is used.
- The generation of cutting planes is deactivated.

5.1. Impact of Symmetry Reduction Techniques

To evaluate the impact of symmetry reduction techniques on running time, we test three different variants of the algorithm: No symmetry reduction at all (NONE), fixing the first visit of a reference customer (FRC), and a combination of reference customer fixing and the symmetry-reduced branching scheme (FRC+SRB). We restrict the experiments in this section to the nine instances with at most 35 customers since for variants NONE and FRC it is not possible to solve larger instances in reasonable time. Furthermore, we set a time limit of ten hours per instance. The running times (T_{total}, in seconds), the number of processed nodes (Num_{Nodes}) as well as the objective values (Obj) for each instance are reported in Table 3. Furthermore, we report the percentage deviation in the running time and in the number of processed nodes relative to variant NONE. Negative values indicate an improvement in the respective value. Note that the deviation between FRC and NONE could not be calculated for those test instances for which in both variants no optimal solution could be found within the time limit. In the table we denote these cases by N/A. Additionally, we use bold-faced numbers to indicate the most successful variant on each instance with respect to running time and number of nodes.

The results show the tremendous impact of the proposed symmetry reduction techniques both on the running time and the number of processed nodes. Without any symmetry reduction techniques, two out of nine instances cannot be solved to optimality within the time limit. Although the fixation of a reference customer is a relatively simple technique, its effect is already remarkable. The average reduction in running time amounts to 75.0%, the average reduction in the number of processed nodes to 74.5%. However, one instance (no. 4) can still not be solved to optimality within the time limit, and for one instance (no. 9) the optimal solution is

Table 3: Impact of symmetry reduction techniques: Running time, number of processed nodes, objective values and deviations with respect to variant NONE.

	Absolute values									Deviations relative to NONE			
	NONE			FRC			FRC+SRB			FRC		FRC+SRB	
Instance no.	T_{total}	Num_{Nodes}	Obj	T_{total}	Num_{Nodes}	Obj	T_{total}	Num_{Nodes}	Obj	T_{total}	Num_{Nodes}	T_{total}	Num_{Nodes}
1	606	2,251	1908.5	130	660	1908.5	**54**	**167**	1,908.5	-78.5%	-70.7%	-91.2%	-92.6%
2	8	37	1228.6	2	**6**	1228.6	**2**	7	1,228.6	-77.3%	-83.8%	-78.3%	-81.1%
3	4,284	12,799	1893.7	504	2,172	1893.7	**114**	**429**	1,893.7	-88.2%	-83.0%	-97.3%	-96.6%
4	36,000[1]	669,752	1761.0	36,000[1]	661,801	1710.7	**458**	**6,181**	**1,702.5**	N/A	N/A	-98.7%[2]	-99.1%[2]
5	33,995	140,657	2006.4	5,768	22,862	2006.4	**2,113**	**8,231**	2,006.4	-83.0%	-83.7%	-93.8%	-94.1%
8	171	421	2070.6	103	234	2070.6	**84**	**201**	2,070.6	-39.5%	-44.4%	-50.8%	-52.3%
9	36,000[1]	219,801	1946.8	36,000[1]	189,585	**1946.6**	**416**	**2,031**	**1,946.6**	N/A	N/A	-98.8%[2]	-99.1%[2]
10	1,787	8,191	1714.8	177	782	1714.8	**29**	**97**	1,714.8	-90.1%	-90.5%	-98.4%	-98.8%
11	315	1,539	2067.8	100	532	2067.8	**16**	**60**	2,067.8	-68.4%	-65.4%	-94.9%	-96.1%
Average	12,574	117,272	1,844.2	8,754	97,626	1838.6	**365**	**1,934**	**1,837.7**	-75.0%	-74.5%	-89.1%	-90.0%

[1] No proven optimal solution found within the time limit.

[2] Compared to the values obtained for variant NONE at the time limit.

found but optimality cannot be proven within the time limit. When the fixation of a reference customer is combined with symmetry-reduced branching, all nine instances can be solved to proven optimality within the time limit. Moreover, the average reduction in running time and in the number of explored nodes is 89.1% and 90.0%, respectively, compared to the case without symmetry reduction.

5.2. Impact of Different Branching Rules

In the following, we compare the performance of the two branching rules LS and PSD. We report in Table 4 the running time, the number of processed nodes, and the relative deviation between PSD and LS branching with respect to the two performance figures.

PSD branching clearly outperforms LS branching. While LS branching is not able to solve three out of the 16 test instances to optimality within the time limit, PSD branching solves all instances to proven optimality. On average, the running times of PSD branching are 37.4% below those of LS branching. The average reduction in the number of processed nodes is 45.9%.

Table 4: Comparison of LS and PSD branching: Running time, number of processed nodes, and deviation of PSD branching relative to LS branching.

	LS		PSD		Relative deviation	
Instance no.	T_{total}	Num_{Nodes}	T_{total}	Num_{Nodes}	T_{total}	Num_{Nodes}
1	93	290	**54**	**167**	-42.3%	-42.4%
2	2	7	2	7	-1.4%	0.0%
3	178	738	**114**	**429**	-35.7%	-41.9%
4	2,235	32,590	**458**	**6,181**	-79.5%	-81.0%
5	36,000[1]	189,426	**2,113**	**8,231**	-94.1%[2]	-95.7%[2]
6	36,000[1]	28,725	**17,385**	**6,639**	-51.7%[2]	-76.9%[2]
7	1,181	4,776	**738**	**2,269**	-37.5%	-52.5%
8	125	332	**84**	**201**	-32.9%	-39.5%
9	1,887	9,802	**416**	**2,031**	-78.0%	-79.3%
10	38	137	**29**	**97**	-23.4%	-29.2%
11	18	78	**16**	**60**	-9.8%	-23.1%
12	4,403	4,097	**3,844**	**2,914**	-12.7%	-28.9%
13	4,099	7,290	**1,700**	**3,023**	-58.5%	-58.5%
14	476	639	**435**	**528**	-8.7%	-17.4%
15	40	36	40	36	0.6%	0.0%
16	36,000[1]	23,527	**24,208**	**7,436**	-32.8%[2]	-68.4%[2]
Average	7,673	18,906	**3,227**	**2,516**	-37.4%	-45.9%

[1] No proven optimal solution found within the time limit.
[2] Compared to the values obtained for LS branching at the time limit.

5.3. Impact of Early Branching

Next, we analyze the impact of early branching on the performance of the algorithm. Table 5 contains the computational results for two variants of the algorithm, namely a variant in which early branching is deactivated (No EB), and a variant in which early branching is enabled (EB). We report again the running times, the number of processed nodes, and the relative deviation

between the two variants for each test instance. Additionally, we include the number of times that the exact pricing method was called (Num_{EP}).

The aim of early branching is to reduce the number of times that the exact pricing method is called. As the results show, this effect is achieved for 13 of the 16 test instances with an average reduction of 22.4%. Unfortunately, the reduction in the number of exact pricing calls does not translate into reduced running times. In fact, running time is reduced only on five instances, whereas it is increased on 11 instances. The average increase in running time amounts to 11.8% and is largely caused by an increase in the number of processed nodes by 16.9%. Since the search trees in the two variants of the algorithm might differ greatly on the same test instance, the reason for the increase in the number of nodes cannot conclusively be explained. We conclude that early branching does, on average, not have the desired effect on the performance of the algorithm, although on specific instances early branching might be beneficial.

5.4. Impact of Subset-row Cuts

In the following, we evaluate the impact of SR cuts on running time and on the number of processed nodes. Again, we compare two variants of the algorithm, one with cut generation being disabled (No Cuts), and one with activated cut generation (SR Cuts). The results are shown in Table 6.

There is no clear tendency whether SR cuts improve the performance of the algorithm. On the one hand, the number of processed nodes can be reduced on ten test instances by enabling cut generation, whereas it is increased only on three test instances. Note that for test instances 6 and 16 it is not possible to evaluate the impact of cut generation as the instances could not be solved optimally within the time limit when cut generation is enabled. The average reduction in the number of processed nodes for the remaining test instances amounts to 8.3%. Without the large outlier obtained on test instance 10, this reduction would even amount to 26.5%. On the other hand, the reduction in the number of processed nodes does not consistently translate into shorter running times. SR cuts reduce the running time on eight test instances, and they also increase the running time on eight test instances. This effect can be explained by the results in Table 7. When SR cuts are applied, the average number of column generation iterations per node ($\frac{Num_{Iter}}{Num_{Nodes}}$) increases. At the same time, the LP relaxation of the RMP and the exact pricing problems become more complex by the inclusion of SR cuts, which can be seen by the increase in the average time per column generation iteration for solving the LP relaxation of the RMP ($\frac{Time_{RMP}}{Num_{Iter}}$, in milliseconds) and by the increase in the average time per call of the exact pricing method ($\frac{Time_{EP}}{Num_{EP}}$, in milliseconds). The latter effect can be observed particularly on test instances 6 and 16, the two test instances with the highest average number of generated cuts per node ($\frac{Num_{Cuts}}{Num_{Nodes}}$). Here, the solution times of Gurobi rise dramatically for some exact pricing problems due to the complexity induced by the large number of SR cuts. These results suggest that, in principle, SR cuts have the potential to accelerate the algorithm, but adding too many of them is detrimental. A more successful strategy could be obtained by adding only a subset of the violated SR inequalities to the RMP such that, on the one hand, the size of the LP relaxation of the RMP and the resulting exact pricing problems is manageable and, on the other hand, still a significant improvement in the number of processed nodes is achieved. Further research is required to investigate such an approach.

Table 5: Impact of early branching: Running time, number of processed nodes, number of exact pricing calls and deviation of variant EB relative to the variant without early branching.

	No EB			EB			Relative deviation		
Instance no.	T_{total}	Num_{Nodes}	Num_{EP}	T_{total}	Num_{Nodes}	Num_{EP}	T_{total}	Num_{Nodes}	Num_{EP}
1	76	233	655	**54**	**167**	**302**	-29.3%	-28.3%	-53.9%
2	2	**4**	17	2	7	**10**	1.6%	75.0%	-41.2%
3	**95**	**308**	742	114	429	**693**	20.3%	39.3%	-6.6%
4	**343**	**3,810**	**5,441**	458	6,181	5,803	33.6%	62.2%	6.7%
5	**1,924**	**6,910**	13,300	2,113	8,231	**10,234**	9.8%	19.1%	-23.1%
6	18,546	**6,628**	38,511	**17,385**	6,639	**25,273**	-6.3%	0.2%	-34.4%
7	757	2,406	4,288	**738**	**2,269**	**2,717**	-2.6%	-5.7%	-36.6%
8	**66**	**156**	361	84	201	**287**	26.4%	28.8%	-20.5%
9	457	**1,870**	3,898	**416**	2,031	**3,152**	-9.0%	8.6%	-19.1%
10	35	113	305	**29**	**97**	**184**	-17.4%	-14.2%	-39.7%
11	**15**	60	99	16	60	**67**	7.7%	0.0%	-32.3%
12	**2,330**	**2,059**	**4,602**	3,844	2,914	5,289	65.0%	41.5%	14.9%
13	**1,105**	**2,192**	**4,744**	1,700	3,023	5,896	53.9%	37.9%	24.3%
14	**392**	**513**	1,107	435	528	**734**	10.8%	2.9%	-33.7%
15	**37**	40	92	40	**36**	**56**	7.1%	-10.0%	-39.1%
16	**20,685**	**6,568**	35,926	24,208	7,436	**27,132**	17.0%	13.2%	-24.5%
Average	**2,929**	**2,117**	7,131	3,227	2,516	**5,489**	11.8%	16.9%	-22.4%

Table 6: Impact of cut generation: Running time, number of processed nodes, and deviation of variant SR Cuts relative to the variant without cutting planes.

	No Cuts		SR Cuts		Relative deviation	
Instance no.	T_{total}	Num_{Nodes}	T_{total}	Num_{Nodes}	T_{total}	Num_{Nodes}
1	54	167	**49**	**123**	-8.3%	-26.3%
2	2	7	2	7	12.6%	0.0%
3	**114**	429	155	**288**	36.0%	-32.9%
4	458	6,181	**239**	**2,854**	-47.7%	-53.8%
5	2,113	8,231	**1,989**	**5,119**	-5.9%	-37.8%
6	**17,385**	6,639	36,000[1]	1,424	107.1%[2]	N/A
7	**738**	2,269	805	**1,637**	9.2%	-27.9%
8	**84**	**201**	124	217	47.4%	8.0%
9	**416**	**2,031**	683	3,240	64.2%	59.5%
10	**29**	**97**	107	318	271.2%	227.8%
11	16	60	**12**	**30**	-26.5%	-50.0%
12	3,844	2,914	**1,338**	**561**	-65.2%	-80.7%
13	1,700	3,023	**1,097**	**1,717**	-35.5%	-43.2%
14	435	528	**334**	**361**	-23.2%	-31.6%
15	40	36	**38**	**26**	-4.9%	-27.8%
16	**24,208**	7,436	36,000[1]	5,041	48.7%[2]	N/A
Average	**3,227**	2,516	4,936	1,435	23.7%	-8.3%

[1] No proven optimal solution found within the time limit.
[2] According to the values obtained for variant SR Cuts at the time limit.

Table 7: SR cuts tend to increase the complexity of the exact pricing problems and of the linear relaxation of the RMP, and they result in a higher number of column generation iterations per node.

	No Cuts			SR Cuts			
Instance no.	$\frac{Num_{Iter}}{Num_{Nodes}}$	$\frac{Time_{EP}}{Num_{EP}}$	$\frac{Time_{RMP}}{Num_{Iter}}$	$\frac{Num_{Iter}}{Num_{Nodes}}$	$\frac{Time_{EP}}{Num_{EP}}$	$\frac{Time_{RMP}}{Num_{Iter}}$	$\frac{Num_{Cuts}}{Num_{Nodes}}$
1	8.0	41.7	18.5	10.3	42.7	17.8	89.8
2	7.6	38.5	9.8	7.9	42.5	9.8	0.4
3	8.4	53.4	12.1	11.5	113.5	14.3	58.0
4	3.9	34.4	3.9	3.9	35.0	4.4	2.1
5	8.2	47.5	14.2	10.0	67.1	15.1	78.9
6	19.6	137.5	81.4	23.4	89,963.5	181.0	718.0
7	9.0	56.1	17.2	11.5	82.8	17.3	64.0
8	7.8	48.9	30.2	11.7	84.8	23.4	117.7
9	5.6	54.9	10.7	6.0	55.8	9.1	1.1
10	8.9	44.2	13.7	8.1	45.1	16.4	18.1
11	6.2	42.3	22.3	8.8	42.7	23.2	26.8
12	9.9	81.3	92.0	16.9	132.8	84.9	400.0
13	12.1	69.4	23.5	14.1	70.4	21.3	32.7
14	11.9	61.1	44.7	15.5	65.4	35.7	68.5
15	11.9	73.1	63.3	17.0	79.8	57.5	46.3
16	15.0	107.9	163.3	20.7	20,042.5	228.2	805.0
Average	9.6	62.0	38.8	12.3	6,935.4	47.5	158.0

5.5. Comparison with Gurobi

We compare the running time of the proposed branch-and-price algorithm with the running time we obtain when we solve the compact formulation ($COMP$) using the general purpose MIP solver Gurobi. To ensure a fair comparison, we extend model ($COMP$) as follows.

We add symmetry breaking constraints to sort the day clusters within each week by the smallest customer index:

$$\sum_{i \in B} v_{ib}^{d} \leq \sum_{i \in B} \sum_{b'=1}^{b-1} v_{ib'}^{d-1} \qquad b \in B \setminus \{1\}, w \in W, d \in D(w) \setminus \{(w-1)m+1\} \tag{68}$$

Sorting day clusters in this way implies that variables v_{ib}^{d} can be fixed to zero for all $i \in B$ and $b < ((d-1) \bmod m) + 1$.

Based on Proposition 1, we fix the service visits of reference customer $b = 1$ as follows:

$$\sum_{i \in B} v_{i1}^{d} = 1 \qquad d \in \{1, mr_1 + 1, 2mr_1 + 1, ..., |D| - mr_1 + 1\} \tag{69}$$

Moreover, we warm-start Gurobi with the solution computed by the location-allocation heuristic of Bender et al. (2016) because we use this solution also to obtain an initial set of columns for our branch-and-price algorithm.

Since only very small instances can be solved with Gurobi, we restrict our experiments again to the nine instances with at most 35 customers. We set the time limit for Gurobi to ten hours per instance and its optimality tolerance with respect to the relative MIP gap to 0.01%. Table 8 contains for both solution methods their respective running times and objective values. Furthermore, we include the relative MIP gap as reported by Gurobi (Gap) and the relative percentage deviation in running time obtained by using the branch-and-price algorithm instead of Gurobi. A star behind the objective value of Gurobi indicates that Gurobi has found an optimal solution.

While Gurobi is able to solve eight of the nine instances to optimality, it fails to prove optimality on four of these eight instances. The average running time obtained with Gurobi is roughly seven hours, whereas it is only about six minutes for the branch-and-price algorithm. The average relative reduction in the running time amounts to more than 98.1%. These results show the huge benefit of a specially-tailored algorithm over a general purpose MIP solver to solve problem MPSTDP-S*. Our branch-and-price algorithm is able to solve instance sizes to proven optimality that are far out of reach for Gurobi.

6. Conclusions and Outlook

In this paper, we studied a highly relevant planning scenario of the scheduling task arising in the context of multi-period service territory design. As far as we are aware, this is the first paper to present an exact branch-and-price algorithm for this problem. In order to accelerate our algorithm, we introduced a fast heuristic to solve the pricing problems and we presented specially-tailored symmetry reduction techniques. In addition, we adopted well-known techniques from literature, such as PSD branching, early branching and SR cuts. We performed

Table 8: Comparison of the performance of Gurobi and the branch-and-price algorithm

	Gurobi			B&P		Relative deviation
Instance no.	T_{total}	Obj	Gap	T_{total}	Obj	T_{total}
1	1,132	1,908.5*	0.01%	54	1,908.5	-95.27%
2	36,000	1,228.6*	1.91%	2	1,228.6	-100.00%
3	36,000	1,893.7*	0.64%	114	1,893.7	-99.68%
4	12,493	1,702.5*	0.01%	458	1,702.5	-96.34%
5	36,000	2,006.4*	0.17%	2,113	2,006.4	-94.13%
8	24,468	2,070.6*	0.01%	84	2,070.6	-99.66%
9	36,000	1,949.1	2.99%	416	1,946.6	-98.84%
10	36,000	1,714.8*	1.27%	29	1,714.8	-99.92%
11	9,844	2,067.8*	0.00%	16	2,067.8	-99.84%
Average	25,326	1,838.0	0.78%	365	1,837.7	-98.19%

extensive computational experiments on real-world instances and investigated the impact of the individual techniques. In particular, the symmetry reduction techniques and PSD branching have proven to increase the performance of the algorithm significantly. On the contrary, early branching did not show the expected effect and the computational experiments on the SR cuts yielded ambivalent results, which necessitates further research. Overall, the results show the effectiveness of our algorithm as all test instances could be solved to proven optimality in reasonable running time. A comparison with the general purpose MIP solver Gurobi revealed that the branch-and-price algorithm reduces running time by over 98.1% on average. This emphasizes the benefit of using a highly specialized algorithm for the problem under study.

The work presented in this paper provides several opportunities for future research. On the one hand, we intend to work on further accelerating the algorithm such that larger problem instances can be tackled. One promising approach for this purpose is the identification of additional families of valid inequalities to tighten the linear relaxation and reduce the number of explored nodes. It is also interesting to investigate if a different decomposition into master and pricing problems is more favorable, e.g., with respect to the development of a fast exact pricing method that is capable of considering additional families of valid inequalities. Moreover, we plan to transform the proposed algorithm into a fast column generation-based heuristic, e.g., by omitting the exact pricing step or by using a heuristic symmetry reduction scheme. Lastly, the individual components of the proposed algorithm can be used as building blocks in problem-specific solution methods for similar problems. On the other hand, the proposed algorithm can be extended to take into account additional planning criteria. Criteria that arise in some applications are, e.g., multiple visits of a customer per week, the restriction to serve a customer only on certain weekdays, different service times for different visits of a customer, and the requirement to serve a customer always on the same weekdays (Bender et al., 2016). While each of these criteria alone could be integrated relatively easily into our algorithm, their combination requires further research.

Acknowledgments

This work was partly supported by the German Academic Exchange Service (DAAD) under Grant No. 57131828. This support is gratefully acknowledged. The authors thank PTV for providing real-world test instances. Moreover, the authors thank the two anonymous referees for their valuable comments, which helped improve the paper considerably.

References

Achterberg, T., Koch, T., Martin, A., 2005. Branching rules revisited. Operations Research Letters 33, 42–54. doi:10.1016/j.orl.2004.04.002.

Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W., Vance, P.H., 1998. Branch-and-price: Column generation for solving huge integer programs. Operations Research 46, 316–329. doi:10.1287/opre.46.3.316.

Bender, M., Meyer, A., Kalcsics, J., Nickel, S., 2016. The multi-period service territory design problem – An introduction, a model and a heuristic approach. Transportation Research Part E: Logistics and Transportation Review 96, 135–157. doi:10.1016/j.tre.2016.09.007.

Desaulniers, G., Desrosiers, J., Solomon, M.M., 2002. Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems, in: Essays and Surveys in Metaheuristics. Springer US, pp. 309–324. doi:10.1007/978-1-4615-1507-4_14.

Desrosiers, J., Lübbecke, M.E., 2005. A primer in column generation, in: Column generation. Springer, pp. 1–32. doi:10.1007/0-387-25486-2_1.

de Fréminville, P.d.l.P., Desaulniers, G., Rousseau, L.M., Perron, S., 2015. A column generation heuristic for districting the price of a financial product. Journal of the Operational Research Society 66, 965–978. doi:10.1057/jors.2014.64.

Jepsen, M., Petersen, B., Spoorendonk, S., Pisinger, D., 2008. Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research 56, 497–511. doi:10.1287/opre.1070.0449.

Kalcsics, J., 2015. Districting problems, in: Laporte, G., Nickel, S., Saldanha da Gama, F. (Eds.), Location Science. 1 ed.. Springer International Publishing, pp. 595–622. doi:10.1007/978-3-319-13111-5.

Linderoth, J.T., Savelsbergh, M.W.P., 1999. A computational study of search strategies for mixed integer programming. INFORMS Journal on Computing 11, 173–187. doi:10.1287/ijoc.11.2.173.

Lübbecke, M.E., Desrosiers, J., 2005. Selected topics in column generation. Operations Research 53, 1007–1023. doi:10.1287/opre.1050.0234.

Mehrotra, A., Johnson, E.L., Nemhauser, G.L., 1998. An optimization based heuristic for political districting. Management Science 44, 1100–1114. doi:10.1287/mnsc.44.8.1100.

Mourgaya, M., Vanderbeck, F., 2007. Column generation based heuristic for tactical planning in multi-period vehicle routing. European Journal of Operational Research 183, 1028–1041. doi:10.1016/j.ejor.2006.02.030.

Padberg, M.W., 1973. On the facial structure of set packing polyhedra. Mathematical Programming 5, 199–215. doi:10.1007/BF01580121.

Pouls, M., 2016. A branch-price-and-cut algorithm for customer scheduling in the context of the multi-period service territory design problem. Master's thesis. Institute of Operations Research (IOR) at the Karlsruhe Institute of Technology. Karlsruhe, Germany.

Savelsbergh, M., 1997. A branch-and-price algorithm for the generalized assignment problem. Operations Research 45, 831–841. doi:10.1287/opre.45.6.831.

Savelsbergh, M., Sol, M., 1998. Drive: Dynamic routing of independent vehicles. Operations Research 46, 474–490. doi:10.1287/opre.46.4.474.