Les Cahiers du GERAD ISSN: 0711–2440

Exact branch-price-and-cut for a hospital therapist scheduling problem with flexible service locations and time-dependent location capacity

A. Jungwirth, G. Desaulniers, M. Frey, R. Kolisch

G–2020–44

August 2020

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2020 – Bibliothèque et Archives Canada, 2020

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2020 – Library and Archives Canada, 2020

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Exact branch-price-and-cut for a hospital therapist scheduling problem with flexible service locations and time-dependent location capacity

Alexander Jungwirth [a]

Guy Desaulniers [b]

Markus Frey [c]

Rainer Kolisch [d]

[a] *BASF SE, Advanced Analytics in Procurement, 67056 Ludwigshafen am Rhein, Germany*

[b] *GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7*

[c] *Hilti AG, Advanced Business Analytics Hilti Global Logistics, 9494 Schaan, Liechtenstein*

[d] *TUM School of Management, Technical University of Munich, 80333 Munich, Germany*

jungwirth.research@gmail.com
guy.desaulniers@gerad.ca
markus.frey@hilti.com
rainer.kolisch@tum.de

August 2020
Les Cahiers du GERAD
G–2020–44

Abstract: We study a new variant of the vehicle routing problem, which arises in hospital-wide scheduling of physical therapists. Multiple service locations exist for patients, and resource synchronization for the location capacities is required as only a limited number of patients can be treated at one location at a time. Additionally, operations synchronization between treatments is required as precedence relations exist. We develop an innovative exact branch-price-and-cut algorithm including two approaches targeting the synchronization constraints: (1) based on branching on time windows, and (2) based on adding combinatorial Benders cuts. We optimally solve realistic hospital instances with up to 120 treatments, and find that branching on time windows performs better than adding cutting planes.

Keywords: Hospital therapist scheduling, time-dependent location capacity, flexible service locations, vehicle routing, exact branch-price-and-cut

---

Acknowledgments: We thank the Bavarian Research Alliance (BayFOR), whose financial support enabled the first author to complete two research stays at the Group for Research in Decision Analysis (GERAD) in Montreal.

1 Introduction

The shortage of qualified personnel is a crucial challenge for health care systems worldwide (WHO 2016). In aging societies, like most western and some Asian ones, demand for health services increases faster than supply. By far the most valuable resource is the health workers having very specialized and expensive education. It is thus crucial for hospitals to use their health workers as efficiently as possible to mitigate the negative effects of understaffed health care systems.

In this context, we study the hospital-wide therapist scheduling and routing problem (ThSRP), a daily planning problem arising at almost every hospital (Gartner et al. 2018). According to the health care planning matrix by Hans et al. (2012) the ThSRP can be classified as an offline operational resource capacity planning problem. On a daily basis, a hospital planner assigns therapists to treatments, treatments to rooms, and start times to treatments. The therapists work different shift patterns (morning, afternoon, or regular shift) and on different levels of qualifications, which define the type of treatment they are available and qualified for. Treatments have a start time window, in which service must begin, and a known treatment duration. If a patient receives multiple treatments in the same day, precedence relations may exist between them, i.e. the preceding treatment must be finished before the succeeding treatment can start. For most patients, two treatment locations exist, i.e. patients can receive service at a central therapy center (TC) or at the ward in which they are staying. There are two noteworthy exceptions: outpatients can only be treated at the TC, and bedridden patients can only be treated at the ward since they must not be moved. In a TC, only a limited number of patients can be treated at a time. Thus, for the TCs the hospital planner must ensure location availability at all times.

Current hospital planning is a manual and time-consuming task leading to unsatisfactory results. Every morning, the available therapists have to be scheduled and assigned to the patients and treatment rooms. For this planning, it is already difficult to generate feasible solutions and frequently the resulting schedules leave some patients unserved.

To facilitate planning, we develop an exact algorithm to solve realistic hospital problems. While in traditional scheduling approaches, the routing of the therapists would be an indirect result, we are addressing the routing decisions explicitly, which has practical as well as theoretical advantages. In large hospitals, therapists are spending a considerable part of their working time traveling between treatment locations. Minimizing traveling gives more time to treat patients, which can be used on a tactical level to increase the number of appointments scheduled per day. However, a hospital might have preferences for treating patients in specific locations and these preferences can conflict with minimizing traveling. Therefore, we associate a certain cost with treating a patient in a specific location, and minimize these location costs together with the traveling.

The resulting problem can be termed as a Vehicle Routing Problem with Time Windows (VRPTW), heterogeneous fleet, operations and resource synchronization, and flexible service locations. Precedence relations between treatments are considered by operations synchronization, and resource synchronization is needed to model the time-dependent location capacity that acts as a renewable resource. As this paper focuses on routing, we will occasionally fall back on the VRP terminology, i.e. instead of treatment and therapist, we use customer and vehicle, respectively. For other applications of routing in hospitals, see e.g., Beaudry et al. (2010), Hanne et al. (2009), and Schmid and Doerner (2014).

We solve the ThSRP by Branch-Price-and-Cut (BPC), in which we address the synchronization constraints using one of the following two alternatives: (1) We branch on the start time windows of the treatments to alter the possible start times such that the current violation cannot occur in subsequent branches, or (2) we add combinatorial Benders cuts to forbid the reoccurrence of the current solution by specifying which combinations of arcs must not appear together. Branching on start time windows received little attention over the past 25 years. However as we will show, it holds the potential of being a valuable building block of solving complex routing problems. To avoid unnecessary branching steps, we use additional branching strategies to break symmetry at higher levels and discuss strategies to correct infeasible solutions.

Our contribution is threefold: (1) We model the ThSRP as a VRPTW with flexible delivery locations and synchronization constraints, which are properties relevant to other VRP variants as well; (2) we develop an exact BPC algorithm for the ThSRP to solve realistic hospital instances, which can be used to derive better schedules with less manual work for hospital planners; and (3) we show that time window branching can be a valid alternative to cutting planes when addressing synchronization constraints in a BPC framework.

The remainder of this work is structured as follows. In §2, we give an overview of the related literature focusing on VRPs. A formal definition of the problem is presented in §3. We describe the details of our BPC algorithm in §4, including specifics of the graph structure of the pricing problem, the branching strategies, and the cuts. In §5, we evaluate the performance of the proposed algorithm and some of its features, and show the robustness of the method against changing inputs. We conclude in §6.

2 Literature

The literature review focuses on VRPs as our main contributions are related to routing problems. First we will give a brief overview of routing related applications in health care contexts, and then we specifically focus on the three components that distinguish our problem from traditional VRP and VRPTW: (a) flexible service location, (b) time-dependent location capacity, and (c) precedence relations. For general literature on VRPs see e.g., Toth and Vigo (2002, 2014) and Vidal et al. (2020). A very recent overview of exact BPC methods for VRPs is given in Costa et al. (2019).

Health care related routing applications are numerous and range from transporting patients to hospitals (Nemati et al. 2016, Lim et al. 2017) over routing of bloodmobiles (Gunpinar and Centeno 2016) to distributing pharmaceutical products to health care facilities (Kramer et al. 2019). A significant share of the heath care literature related to routing is concerned with home care applications, which are reviewed in Cissé et al. (2017) and Fikar and Hirsch (2017).

Only few papers study routing in hospitals. Hanne et al. (2009) and Beaudry et al. (2010) develop a heuristic to solve a routing problem with fixed locations and dynamically arriving transportation requests. Schmid and Doerner (2014) present a metaheuristic for an integrated planning problem combining scheduling and routing of patients. The ThSRP is studied in Gartner et al. (2018) and Jungwirth et al. (2020). Gartner et al. (2018) present an insertion heuristic and an exact algorithm. The exact algorithm solves a scheduling problem after relaxing the routing constraints, and if the routing constraints are violated in the optimal solution, a cutting plane is added and the problem is solved again. To the best of our knowledge, this is the only exact algorithm for the ThSRP so far. Jungwirth et al. (2020) are the first to model the ThSRP as a VRP, which they solve heuristically using a hybrid adaptive large neighborhood search.

In the remainder of this review, we will focus on routing problems showing similarities to the ThSRP in terms of modeling and methodology. In the literature, we found two types of problems involving flexible service locations: (1) Beasley and Nascimento (1996) introduced the vehicle routing-allocation problem, in which multiple possible service locations exist for customers. These locations are uncapacitated and customers can also be left unserved. Practical applications are, e.g. routing mobile clinics in rural areas or designing postal collection routes. (2) Reyes et al. (2017) introduced the VRP with roaming delivery locations in which parcels are delivered to different locations depending on the time of the day. A branch-and-price algorithm for this problem is developed in Ozbaygin et al. (2017). Locations are not capacitated and the structure of the time windows differs from our case as different locations have distinct and non-overlapping time windows.

The stream of literature sharing most similarities with our problem is the VRP with synchronization constraints. According to the taxonomy of Drexl (2012) our time-dependent location capacities belong to resource synchronization and our precedence constraints belong to operations synchronization. Generally, tours in VRPs are independent of one another, in the sense that a change within one

tour does not affect another tour. However, when resource synchronization is present, a change within one tour could potentially change all other tours.

If location capacity is considered in the VRP literature, these capacities generally do not affect the routing decisions of the vehicles, i.e. vehicles simply wait at the location until the resource becomes available. An example is log truck scheduling studied by Hachemi et al. (2013) and Rix et al. (2015). In this forestry application, log-loaders can only load one truck at a time, and if the loader is in use, other trucks have to wait. In the VRP with location congestion (Lam and Hentenryck 2016), vehicles are synchronized with regard to e.g., parking lots and forklifts. Dynamic scheduling of automated guided vehicles at airports is studied by Ebben et al. (2005) with scarce resources being docks for (un)loading and parking space. If the resources are unavailable, vehicles wait at the destination or begin their tours later.

Recently, limited availability of recharging and refueling has been studied in electric-VRPs (E-VRPs) and green VRPs (G-VRPs), cf. Keskin et al. (2019) and Froger et al. (2019) for E-VRPs, and Bruglieri et al. (2019) for G-VRPs. In Keskin et al. (2019), vehicles wait until charging becomes available. In contrast, in Bruglieri et al. (2019) and Froger et al. (2019) the time-dependent location capacity is a central part of the routing decision. From the routing literature, the problems the latter authors study are the closest to the ThSRP. As in our case, it is not known in advance which tasks will be needed to be synchronized and the synchronization affects the routing decisions.

In E-VRPs and G-VRPs, refueling between two customer visits is possible. In Froger et al. (2019) (E-VRP), refueling happens at a charging station (CS) equipped with a number of charging points, and in Bruglieri et al. (2019) (G-VRP), alternative fuel stations with multiple fuel pumps can be visited. Examples of alternative fuels are e.g., methane and electricity. Froger et al. (2019) include more specifics of recharging e.g., non-linear charging functions, and vehicles do not need to recharge to full capacity when visiting a CS.

Both works consider capacity at the level of the charging points. Each charger has a capacity of 1, i.e. exactly one vehicle can recharge at a time. However, there can be multiple charging points at a CS. In each work, two mathematical models are presented: (a) a replication (or arc-based) formulation, and (b) a path-based formulation. In the replication-based formulation, an upper bound of the maximum number of possible visits of one charger is determined in a pre-processing step, and then the corresponding number of virtual clones of the chargers are created. At most one vehicle can visit each clone and as the clones are temporarily synchronized, location capacity is ensured. The path-based formulation does not require copying the CSs, instead multiple arcs are combined into paths, and the paths are then synchronized to avoid location capacity violations. The main difference between these works and ours is that in their case, charging is capacitated and charging enables visiting additional customers. However, the service locations of the customers itself are not capacitated, neither do flexible service locations exist. In ThSRP, enforcing capacity at the service location has influence on the decision where to serve customer. Another central difference is the importance of time windows in our case. Time windows are not considered in Froger et al. (2019), and Bruglieri et al. (2019) only discuss to add time slots at CSs, which could be reserved, however these are not time windows at service sites.

The third distinctive component of the ThSRP are the precedence constraints, which belong to operations synchronization according to the taxonomy of Drexl (2012). This type of synchronization is more common in the VRP literature than the resource synchronization. It occurs e.g., in dial-a-ride problems and in pickup-and-delivery problems (see e.g., Battarra et al. 2014, Doerner and Salazar-González 2014, Ho et al. 2018).

Table 1 summarizes the properties of the most related routing problems. Only few papers consider flexible service locations or time-dependent location capacities. Precedence relations and time windows were either not or not much studied in these works. To the best of our knowledge, we are the first to address all four aspects combined in one problem.

Table 1: Overview of VPRs related to the ThSRP

	Br	Fr	Be	Re/Oz	Our work
Flexible service locations			✓	✓	✓
Time-dependent location capacity	✓	✓			✓
Time windows	(✓)			✓	✓
Precedence relations					✓

(Br) Bruglieri et al. (2019); (Fr) Froger et al. (2019); (Be) Beasley and Nascimento (1996); (Re) Reyes et al. (2017); (Oz) Ozbaygin et al. (2017).

Outside of the VRP domain, operations and resource synchronization naturally occur in resource-constrained project scheduling problems (RCPSP). See Brucker et al. (1999) for an overview of RCPSPs and De Reyck et al. (1999) particularly for precedence relations.

3 Problem statement and set covering formulation

In this part, we state the problem and introduce the required notation along the way. Patients are the central planning entity in the ThSRP. A patient has at least one treatment during the planning day, and the set of all treatments is denoted by $\mathcal{I}$. Depending on the type of patient, a treatment $i \in \mathcal{I}$ can take place in one of a set of potential service locations $l \in \mathcal{L}_i$ with $\mathcal{L}_i \subseteq \mathcal{L}$ being a subset of the set of all locations $\mathcal{L}$. Most treatments can take place either at the central TC or at the ward in which patients are staying. Since multiple patients can be treated at the TC at a time, we have to ensure that the location capacity of the TC is not exceeded. For ward rooms, we assume unlimited capacity as no patient will be scheduled to another patient's ward room. By $\mathcal{L}^{\text{bnd}}$ we denote the set of bounded locations, i.e. locations $l \in \mathcal{L}^{\text{bnd}}$ having a capacity limit Q_l. For bounded locations, the capacity has to be considered only at times when sufficient treatments can potentially overlap. The set of critical time points of a location $l \in \mathcal{L}^{\text{bnd}}$ is denoted by $\mathcal{T}_l^{\text{bnd}}$ and can be determined in a preprocessing step.

A treatment i requires a service time s_i and must begin within starting time window $[a_i, b_i]$. Outpatients i have a fixed start time, i.e. $a_i = b_i$, while for inpatients j we assume $a_j < b_j$. Since a patient might undergo several treatments per day, precedence relations between treatments can exist, i.e. the preceding treatment must be finished before the succeeding treatment can start. If the two treatments take place in different locations, the travel time between the locations has to be considered as well. By $\mathcal{P}$ we denote the set of precedence relations (i, j) with $i \in \mathcal{I}$ being the preceeding treatment and $j \in \mathcal{I}$ the succeeding treatment. The travel time between locations l_1 and l_2 is denoted by t_{l_1,l_2}. We assume that the travel times satisfy the triangle inequality. Furthermore, we assume that travel times are integer or can be scaled to integers. Treatments differ in complexity and thus require different qualifications. We assume that the qualifications are hierarchical and denote by $\mathcal{Q}$ the set of qualifications.

Treatments are performed by a set of therapists $\mathcal{K}$. We define i_0 and i_{n+1} to be dummy treatments for leaving and entering the depot location l_0 that corresponds to the break room in which therapists are staying when they are not treating patients. The therapists differ in their shift type $s \in \mathcal{S}$ and their level of qualification $q \in \mathcal{Q}$. To treat a patient the shift must overlap with the treatment's start time window and the therapist's qualification q_k must at least match the required qualification of the treatment q_i, i.e. $q_i \leq q_k$. The subset of therapists with shift type s and qualification q is denoted by $\mathcal{K}_{s,q}$. A therapist belongs only to one set $\mathcal{K}_{s,q}$, i.e. for two levels of qualification $q_1 < q_2$ we have $\mathcal{K}_{s,q_1} \cap \mathcal{K}_{s,q_2} = \emptyset$.

As a result of planning, each therapist is assigned a tour-schedule $r \in \Omega$. The tour-schedule contains information about the sequence of treatments, the service locations and the start times. The subset of tour-schedules for therapists of shift type s and qualification q is denoted by $\Omega_{s,q} \subseteq \Omega$. To remain concise, we will only use the term tour instead of tour-schedule in the remainder.

We associate a cost c_r with each tour $r \in \Omega$, which includes costs $c^{\text{travel}}_{l_1 l_2}$ for traveling between locations $l_1, l_2 \in \mathcal{L}$ and costs $c^{\text{location}}_{i,l}$ for treating i at location l. The latter is relevant to model that hospitals might have preferences for one location over the other. We assume travel costs to be equivalent to travel times, thus the triangle inequality holds for the travel costs as well.

In summary, the ThSRP consists in finding least-cost tours for the available therapists such that every treatment is scheduled at a valid location and start time with a qualified therapist, capacity is never exceeded in all bounded locations, precedence relations are satisfied, and tours are feasible with respect to time and corresponding therapist's shift.

To formulate the ThSRP mathematically, we introduce the following additional notation. Binary parameter $\alpha_{i,l,r}$ indicates if treatment i in location l is part of tour r. Binary parameter $\beta_{l,t,r}$ controls if location l is occupied at point in time t by tour r, and by parameter $T_{i,l,r}$ we denote the start time of treatment i in location l in tour r. Decision variable $\gamma_{s,q}$ holds the number of therapists in set $\mathcal{K}_{s,q}$ having at least one treatment. Decision variable λ_r is equal to 1 if tour $r \in \Omega$ is selected in the solution, and 0 otherwise. The objective is to minimize the sum of the costs c_r over all tours r. With the definitions above, the set covering formulation of the ThSRP can be stated as:

$$\min \sum_{(s,q)\in\mathcal{S}\times\mathcal{Q}} \sum_{r\in\Omega_{s,q}} c_r \cdot \lambda_r \tag{1}$$

subject to

$$\sum_{(s,q)\in\mathcal{S}\times\mathcal{Q}:i\in\mathcal{I}_{s,q}} \sum_{r\in\Omega_{s,q}} \sum_{l\in\mathcal{L}_i} \alpha_{i,l,r} \cdot \lambda_r \geq 1 \quad \forall\, i \in \mathcal{I} \tag{2}$$

$$\sum_{r\in\Omega} \beta_{l,t,r} \cdot \lambda_r \leq Q_l \quad \forall\, l \in \mathcal{L}^{\text{bnd}}, t \in \mathcal{T}^{\text{bnd}}_l \tag{3}$$

$$\left(T_{i,l_i,r} + s_i + t_{l_i,l_j} - T_{j,l_j,r'}\right) \cdot \alpha_{i,l_i,r} \cdot \alpha_{j,l_j,r'} \cdot \lambda_r \cdot \lambda_{r'} \leq 0 \quad \forall\, r, r' \in \Omega, (i,j) \in \mathcal{P} \tag{4}$$

$$\sum_{r\in\Omega_{s,q}} \lambda_r = \gamma_{s,q} \quad \forall\, (s,q) \in \mathcal{S}\times\mathcal{Q} \tag{5}$$

$$0 \leq \gamma_{s,q} \leq |\mathcal{K}_{s,q}| \quad \forall\, (s,q) \in \mathcal{S}\times\mathcal{Q} \tag{6}$$

$$\lambda_r \in \{0,1\} \quad \forall\, r \in \Omega \tag{7}$$

Objective (1) minimizes the cost over all selected tours. Assignment constraints (2) ensure that each treatment i is done at least once. Location capacity constraints (3) guarantee for every capacitated location l and for every point in time t that the capacity limit Q_l is not exceeded. Precedence constraints (4) establish that preceding treatments are finished before the succeeding treatments are started. To satisfy the precedence relations, the service duration and the travel times between the service locations must be considered. Constraints (4) are nonlinear and numerous. However, they will be relaxed in our solution algorithm and treated through branching decisions. Constraints (5) compute the number of therapists used in each shift and for each qualification level. These numbers are bounded in (6). Note that (5) and (6) could be merged and the $\gamma_{s,q}$ variables removed. We present them separately because the proposed algorithm will branch on these variables. Lastly, the domain of the λ_r variables is defined in (7).

4 Branch-price-and-cut algorithm

To solve directly the set covering formulation (1)–(7), we would need to enumerate all possible tours in Ω. As generating such a large number of tours is practically impossible, column generation (CG) is used and embedded in a branch-and-cut framework to yield an exact BPC algorithm. (cf. Vanderbeck and Wolsey 1996, Barnhart et al. 1998, Desaulniers et al. 2005). CG works with a reasonably small subset of tours $\Omega' \subseteq \Omega$, i.e. $\Omega'_{s,q} \subseteq \Omega_{s,q}$ in our case, and dynamically generates new tours as needed. Since a column represents a tour, we use these terms interchangeably. Before our CG algorithm starts, we first relax from formulation (1)–(7) the synchronization constraints for the location capacity (3) and

the precedence relations (4) to get a set-covering formulation equivalent to that of the heterogeneous VRPTW. Then, we solve by CG the resulting linear relaxation, which is then called the master problem. CG is an iterative method that solves at each iteration a restricted master problem (RMP) and a pricing problem (PP). The RMP considers a subset of the tour variables and writes as:

$$\min \sum_{(s,q)\in\mathcal{S}\times\mathcal{Q}} \sum_{r\in\Omega'_{s,q}} c_r \cdot \lambda_r \tag{8}$$

subject to

$$\sum_{(s,q)\in\mathcal{S}\times\mathcal{Q}:i\in\mathcal{I}_{s,q}} \sum_{r\in\Omega'_{s,q}} \sum_{l\in\mathcal{L}_i} \alpha_{i,l,r} \cdot \lambda_r \geq 1 \quad \forall\, i \in \mathcal{I} \tag{9}$$

$$\sum_{r\in\Omega'_{s,q}} \lambda_r - \gamma_{s,q} = 0 \quad \forall\, (s,q) \in \mathcal{S}\times\mathcal{Q} \tag{10}$$

$$0 \leq \gamma_{s,q} \leq |\mathcal{K}_{s,q}| \quad \forall\, (s,q) \in \mathcal{S}\times\mathcal{Q} \tag{11}$$

$$\lambda_r \geq 0 \quad \forall\, r \in \Omega' \tag{12}$$

In a CG iteration, once the RMP is solved, its dual solution is used to generate new promising tours by solving a (PP). The PP aims to find new tours with negative reduced cost. If new tours are found, they are added as columns to the RMP, and if no new columns are found, CG terminates and a valid lower bound for the minimization problem was found. Details on the PP are given in §4.1.

When an integer feasible solution is found during CG, we still need to check if the solution is also feasible for relaxed constraints (3) and (4). This feasibility check is applied at any CG iteration when we find an integer solution, not only when CG terminates. To derive integer solutions, CG is embedded in a branch-and-cut framework to yield a BPC algorithm. When CG ends with a feasible integer solution at a node of the search tree, the node is pruned. When it ends with an infeasible integer solution, we use one of the following two approaches with the goal of retrieving feasibility: (a) branching on start time windows (see §4.2.5), or (b) adding combinatorial Benders cuts (see §4.3). When CG ends with a fractional solution and the node cannot be pruned, branching or cutting is performed. To reach integrality, traditionally branching on the number of vehicles (therapists) and branching on arcs connecting customers (treatments) is used for VRPs (cf. Feillet 2010). We use both strategies and introduce three additional branching strategies before branching on arcs: (a) Branching on the type of therapist performing a treatment, (b) branching on possible service location of a treatment, and (c) branching on aggregated therapy arcs. Our branching strategies are described in detail in §4.2. To tighten the dual bound, we add limited-memory subset-row cuts (lm-SRCs) introduced by Pecin et al. (2017), and we propose location-capacity cuts (LCC) to exclude combinations of treatments that will result in a capacity violation of a specific service location. Details of the cuts are given in §4.3. A schematic overview of our BPC algorithm is displayed in Figure 1.

4.1 Pricing problem

To generate new columns for the RMP, we solve $|\mathcal{S}\times\mathcal{Q}|$ PPs, each corresponding to a specific therapist type. The PPs, also called subproblems, correspond to elementary shortest path problems with resource constraints (ESPPRC) and aim at finding new feasible tours $r \in \Omega$ with negative reduced cost. Let $\pi_i^{(9)}$ and $\pi_{s,q}^{(10)}$ be the duals associated with constraints (9) and (10) of the RMP, respectively. Then, the ESPPRC is equivalent to the following minimization problem for a given shift type $s \in \mathcal{S}$ and qualification $q \in \mathcal{Q}$:

$$\min_{r\in\Omega_{s,q}} \bar{c}_r = c_r - \sum_{i\in\mathcal{I}_{s,q}} \sum_{l\in\mathcal{L}_i} \alpha_{i,l,r} \cdot \pi_i^{(9)} - \pi_{s,q}^{(10)} \tag{13}$$

The ESPPRC is modeled on a graph $G = (\mathcal{V}, \mathcal{A})$ with vertex set $\mathcal{V}$ and arc set $\mathcal{A}$. The definition of the node set $\mathcal{V}$ differs from classical VRPs (cf. Pecin et al. 2017, Costa et al. 2019). Instead of having only one node v_i per treatment for a customer, we have $|\mathcal{L}_i|$ nodes $v_{i,l}$, one for each possible service location $l \in \mathcal{L}_i$ for a treatment of a customer. We define node v_0 to represent the depot and the subset

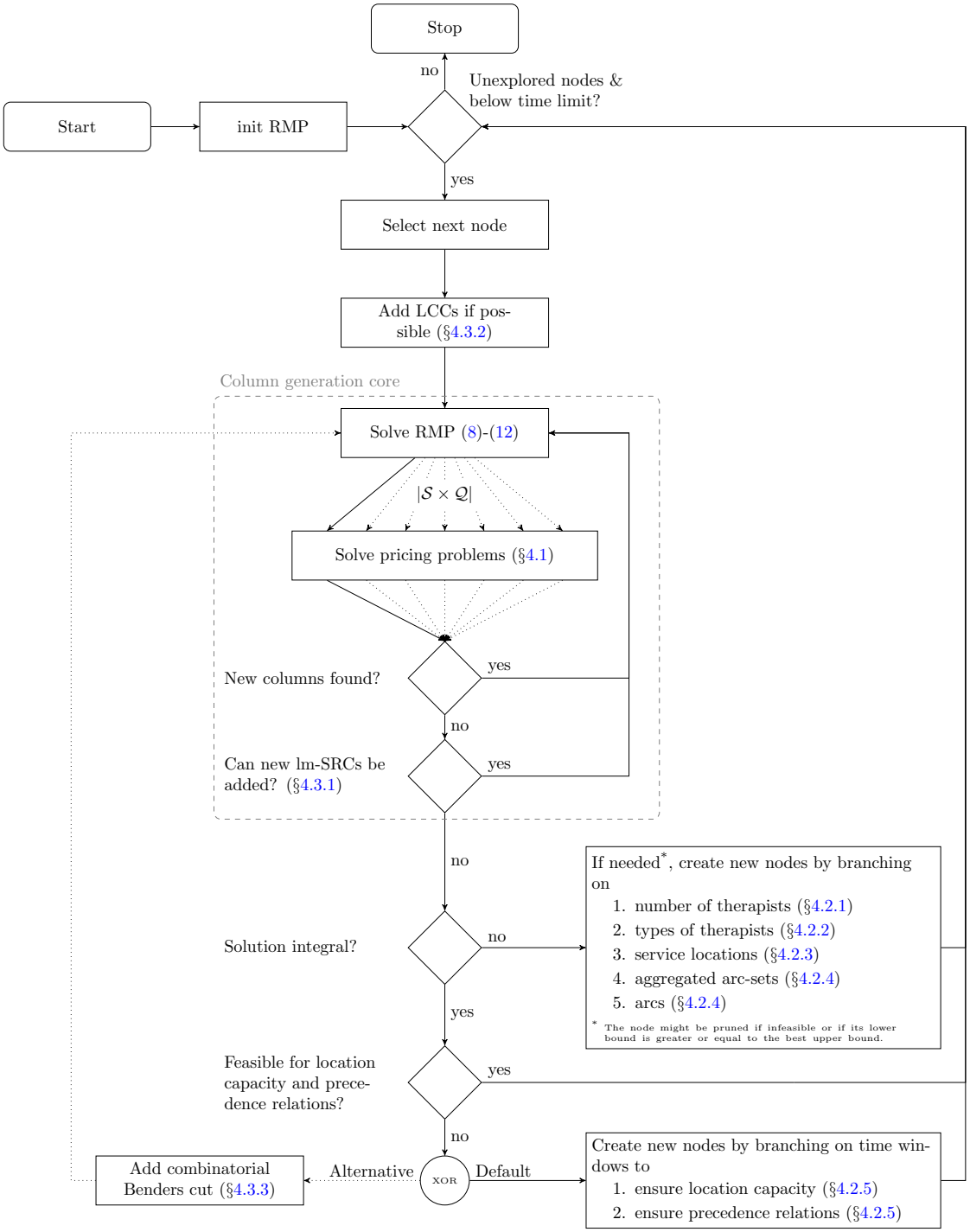

Figure 1: Schematic overview of our BPC algorithm

of nodes without the depot is denoted by $\mathcal{V}' = \mathcal{V} \setminus \{v_0\}$. Figure 2 displays an example of how the complexity of graph G increases in our problem compared to the VRP.

An individual start time window is assigned to each node, i.e. even nodes representing the same treatment i might have different time windows. Since a node contains location information, time window tightening might affect nodes belonging to the same treatment i but different location l, differently, and branching on time windows presented in §4.2.5 affects the time windows of the nodes rather than the ones of the treatments.

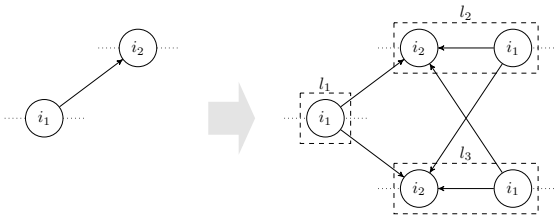

Figure 2: Arc (i_1, i_2) in the VRP (left) expands into a subgraph in the ThSRP (right)

For arc set $\mathcal{A} \subseteq \mathcal{V} \times \mathcal{V}$, we have $(v_{i,l}, v_{j,r}) \in \mathcal{A}$ with $v_{i,l_1}, v_{j,l_2} \in \mathcal{V}$ and $i \neq j$ if and only if treatment i can be serviced at location l_1 before treatment j is served at location l_2. The costs $\bar{c}_{v_{i,l_1}, v_{j,l_2}}$ along an arc $(v_{i,l_1}, v_{j,l_2}) \in \mathcal{A}$ are defined as:

$$\bar{c}_{v_{i,l_1}, v_{j,l_2}} = c^{\text{travel}}_{l_1,l_2} + c^{\text{location}}_{j,l_2} - \pi^{(9)}_i \tag{14}$$

Travel cost is defined between location $l_1, l_2 \in \mathcal{L}$ and costs for servicing in location $l_2 \in \mathcal{L}$ are considered for the head node. The dual $\pi^{(10)}_{s,q}$ associated with the convexity constraint (10) is also subtracted at the depot node v_0.

We solve the ESPPRC by using a dynamic programming labeling algorithm starting from the source v_0 and iteratively extends partial paths to new nodes $v_{i,l} \in \mathcal{V}$ until it reaches the sink $v_{n+1} = v_0$. Relevant information for a partial path p is stored in a label L_p associated with the end node of p. Specifically, L_p contains the following information: the parent label id, the cumulated reduced cost, the earliest start time of service at the node, and a binary vector stating which treatments can still be done. In order to not enumerate all paths, a dominance criterion is applied to discard non-useful labels. For further details on labeling algorithms, we refer the reader to Feillet et al. (2004), Irnich and Desaulniers (2005), and Irnich (2008).

Since each PP is called frequently, we apply the following acceleration techniques: time window tightening (Desrochers et al. 1992, Kontoravdis and Bard 1995), bidirectional search (Righini and Salani 2006), decremental state-space relaxation (Boland et al. 2006, Righini and Salani 2008), heuristic pricing as long as negative reduced cost columns can be found (Desaulniers et al. 2008), and adding buckets of multiple task-disjoint columns to the RMP in each pricing step (Desaulniers et al. 2002).

4.2 Branching

Our branching strategies serve two purposes: first we branch to reach integrality, and second we branch to enforce the previously relaxed location capacity and precedence relation constraints. To reach integrality, we use five branching strategies applied in the following sequence:

1. branching on the number of therapists (see §4.2.1),
2. branching on the types of therapists to perform a treatment (see §4.2.2),
3. branching on possible service locations of treatments (see §4.2.3),
4. branching on aggregated arc-sets (see §4.2.4), and
5. branching on arcs (see §4.2.4).

While branching strategies (1) and (5) are commonly used for VRPs, we introduce strategies (2), (3), and (4) to break fractional flows on a higher levels than in (5). When integrality is reached and if the precedence or location capacity constraints are violated, we branch on start time windows to forbid a reoccurrence of the violations. Branching is applied in the following arbitrarily chosen order:

1. branching on start time windows to ensure precedence (see §4.2.5),
2. branching on start time windows to ensure location capacity (see §4.2.5).

The search tree is explored using a best-first strategy. We are always applying strategies (1) to (5) to reach integrality, however instead of applying strategies (6) and (7) to reach feasibility with respect to the location capacity and precedence relation constraints, we could alternatively use combinatorial cuts as well (see §4.3.3). In the following, we will describe the seven branching strategies in detail.

4.2.1 Branching on the number of therapists.

Desirable properties of branching are: (a) a balanced branching tree, i.e. that the remaining problems in each branch are approximately equally complex, and (b) a limited growth of the branching tree. For routing problems, branching on the number of vehicles (therapists) can often have a significant positive influence on the tree size (Costa et al. 2019). Let $f_{s,q}$ be the fractional number of therapists of type (s,q) in the optimal solution at a branching node, then two branches are generated: (1) forcing $\gamma_{s,q} \leq \lfloor f_{s,q} \rfloor$ and (2) forcing $\gamma_{s,q} \geq \lceil f_{s,q} \rceil$. If several therapist types (s,q) with fractional flows exist, we branch first on the type of therapist (s,q) with most fractional flow as calculated in Equation (15).

$$\underset{(s,q)\in\mathcal{S}\times\mathcal{Q}}{\arg\min} = |0.5 - (f_{s,q} \bmod 1)| \tag{15}$$

4.2.2 Branching on the types of therapists

If a treatment is serviced by therapists of different types in an optimal solution of the current linear relaxation, we branch on the set of therapists, which can perform the service. Let $\mathcal{S}_i \times \mathcal{Q}_i$ be the set of therapist types that can perform treatment i. If $|\mathcal{S}_i \times \mathcal{Q}_i| \geq 2$, this set can be indexed and split in half. For both subsets, we generate a branch. To balance the branching tree, we split $\mathcal{S}_i \times \mathcal{Q}_i$ such that both halves account roughly for a flow of 0.5. We define the point μ_i, at which we split the set of therapist types as the last element that remains in subset 1 for treatment i. This split point μ_i is determined by using the formula (16), with f_k being the cumulated flow of therapist type k covering treatment i.

$$\mu_i = \underset{n\in\{1,\ldots,|\mathcal{S}_i\times\mathcal{Q}_i|\}}{\arg\min} \left| 0.5 - \sum_{k=1}^{n} f_k \right| \tag{16}$$

As each PP corresponds to a specific therapist type, we can impose the branching decisions by simply removing nodes from graph G that can no longer be covered by the specific therapist type.

4.2.3 Branching on the locations

When $|\mathcal{L}_i| > 1$ for a treatment i, it can happen that, in an optimal solution, treatment i is serviced by an integer number of therapists of the same type, but in more than one location. In this case, we branch on the set of possible service locations. Equivalently to §4.2.2, we split $\mathcal{L}_i$ such that the flow in the remaining halves is as close as possible to 0.5. The split position μ_i, i.e. the last location that is part of location subset 1, is determined by the following formula:

$$\mu_i = \underset{n\in\{1,\ldots,|\mathcal{L}_i|\}}{\arg\min} \left| 0.5 - \sum_{k=1}^{n} f_k \right| \tag{17}$$

We impose the branching decisions by removing from all PPs all nodes corresponding to the locations that can no longer host the treatment.

4.2.4 Branching on arcs flows

Desrosiers et al. (1984) introduced branching on arc flows for the VRPTW and it has become the most popular branching strategy ever since (Costa et al. 2019). Compared with standard VRPTWs, the graph G in our PP contains more edges as multiple nodes exist per treatment (cf. §4.1). Thus, if we would apply pure arc branching, we would need more branching steps than traditional VPRTWs require. Therefore, we first branch on aggregated arc-sets, and once the flows on the aggregated arcs are integral, we continue by branching on arcs $a \in \mathcal{A}$ as defined for graph G.

Branching on aggregated arc-sets. Let v_i be an aggregation set of all nodes belonging to the same treatment i, i.e. $v_i = \bigcup_{l \in \mathcal{L}_i} v_{i,l}$, and let (v_i, v_j) be an aggregated arc connecting two aggregated nodes v_i and v_j, i.e. (v_i, v_j) contains all arcs connecting all nodes in v_i with all nodes in v_j. Branch 1 enforcing that aggregated arc (v_i, v_j) cannot belong to the solution is constructed by removing all arcs $(v_{i,l_1}, v_{j,l_2}) \in \mathcal{A}$ (with $l_1 \in \mathcal{L}_i$ and $l_2 \in \mathcal{L}_j$) connecting any node in v_i to any node in v_j, and deleting all columns $r \in \Omega'$ containing any of these arcs. Branch 2 enforcing that arc (v_i, v_j) is used in the solution is constructed by removing all arcs $(v_{i'}, v_j) \in \mathcal{A}$ such that $i' \neq i$ and $i' \neq i_0$, and all arcs $(v_i, v_{j'}) \in \mathcal{A}$ such that $j' \neq j$ and $j' \neq i_{n+1}$ from the PPs with i_0 and i_{n+1} being the dummy treatments for leaving and entering the depot.

Branching on arcs. If the flow of all aggregated arcs is integer, there might still exist fractional flows on the arcs in $\mathcal{A}$ on which we can branch. Branch 1 enforcing that arc (v_{i,l_1}, v_{j,l_2}) cannot belong to the solution is constructed by removing arc (v_{i,l_1}, v_{j,l_2}) from the PPs. Branch 2 enforcing that arc (v_{i,l_1}, v_{j,l_2}) is used in the solution is constructed by removing all arcs $(v_{i',l_1'}, v_{j,l_2}) \in \mathcal{A}$ such that $(i' \neq i \wedge i' \neq i_0) \vee l_1' \neq l_1$, and by removing all arcs $(v_{i,l_1}, v_{j',l_2'}) \in \mathcal{A}$ such that $(j' \neq j \wedge j' \neq i_{n+1}) \vee l_2' \neq l_2$.

4.2.5 Branching on time windows

Once an integer feasible solution is reached, we check if this solution is also feasible for the precedence constraints and location capacity constraints. If the solution is infeasible, we branch on the start time windows of the treatments such that the same violation is prevented from reoccurring. Note that violations involving the same precedence relation or capacity violations can still occur after branching on time windows, however only at a different time. Branching on start time windows belongs to branching on resource windows and has been applied in the context of vehicle routing, e.g. in Desrosiers et al. (1986) and Gélinas et al. (1995).

Ensure precedence. A precedence relation $(i, j) \in \mathcal{P}$ ensures that treatment i must have been finished before treatment j can start, i.e. $T_i + s_i + t_{l_i,l_j} \leq T_j$. As our algorithm works on nodes $v_{i,l}$ representing a combination of treatment i and location l, we consider start time $T_{i,l}$ associated with the node. Therefore, a precedence relation violation occurs if $T_{i,l_i} + s_i + t_{l_i,l_j} > T_{j,l_j}$ with $l_i \in \mathcal{L}_i$ and $l_j \in \mathcal{L}_j$. In that case, we branch on the start time windows of nodes v_{i,l_i} and v_{j,l_j}. In branch 1, the start time window of node v_{i,l_i} is adjusted such that service must start slightly earlier, and in branch 2, the start time window of node v_{i,l_i} is adjusted such that service must start slightly later. Since i and j are part of a precedence relation, j must also start later in branch 2, i.e. the time window of node v_{j,l_j} is adjusted as well. Figure 3 illustrates the resulting branches and the updates of the time windows TW associated with the nodes.

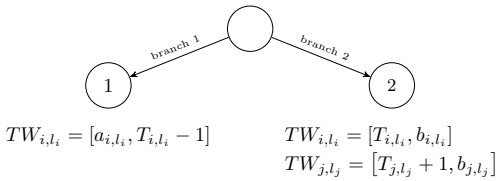

Figure 3: Resulting branches for branching on time windows to ensure precedence relations

If a column contains a node $v_{i,l_i} \in \mathcal{V}$ and the start time of that node is outside of the updated time window TW_{i,l_i}, then the column is deleted from Ω'. If the time windows can no longer be adjusted, because $a_{i,l_i} = b_{i,l_i}$ for any node $v_{i,l_i} \in \mathcal{V}$, then node v_{i,l_i} is removed from the PPs and all columns containing v_{i,l_i} are removed from Ω'. Note that, treatment i can still be served in the remaining locations $l \in \mathcal{L}_i \setminus \{l_i\}$. If no further location exists for i, infeasibility has been proven and the current branching node can be pruned. If more than one precedence violation exists, we first branch on the largest violation in terms of time units as given in (18).

$$\underset{\substack{v_{i,l_i}, v_{j,l_j} \in \mathcal{V}: \\ (i,j) \in \mathcal{P}}}{\arg\max} \left(T_{i,l_i} + s_i + t_{l_i,l_j} - T_{j,l_j} \right) \tag{18}$$

Ensure location capacity. A location capacity Q_l is violated if more than Q_l patients are scheduled to receive treatment in parallel at location l. We give an example of capacity violations in Figure 4 where the upper part displays the time periods during which the treatments are in execution, and the lower part displays the cumulated capacity usage. Since all treatments are assigned to location l, we omit index l and write T_i, a_i and b_i instead of $T_{i,l}, a_{i,l}$ and $b_{i,l}$, etc. Let $\mathcal{T}$ be the set of start times T_i for all treatments i in an integer feasible solution, then $q_l(\mathcal{T}, t) = \{i \in \mathcal{I}_l \,|\, T_i \leq t \leq T_i + s_i\}$ is a function returning the set of treatments, which are in process at time t in location l. In the example, the capacity limit is $Q_l = 2$, and a capacity violation occurs between time points $t = 6$ and $t = 10$. There are actually two violations: the first for $t = [6, 9]$ by treatments $\{i_1, i_3, i_4\}$, and the second between $t = [9, 10]$ by treatments $\{i_2, i_3, i_4\}$.

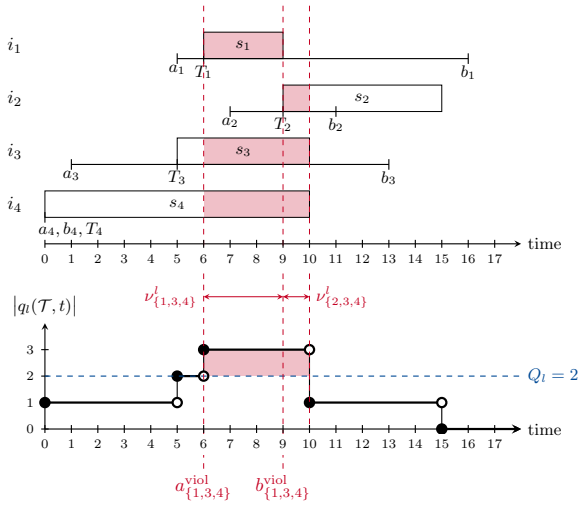

Figure 4: Location capacity violation

At a branching node, we only branch on one violation at a time and choose first the violation with the longest period of resource excess. We define an arbitrary time point τ, at which we split the violation. Without loss of generalization, we assume for the example that the violation is split in half, i.e. τ is defined as $\tau = (a^{\text{viol}} + b^{\text{viol}})/2$ with a^{viol} and b^{viol} being the bounds of the violation. We generate two branches for each treatment i involved in the violation and adjust the start time windows of the treatment-location nodes: (1) such that the treatment starts either early enough that service is done before τ, i.e. $b_i^{\text{new}} = \lceil \tau \rceil - s_i - 1$, or (2) that service starts later than τ, i.e. $a_i^{\text{new}} = \lceil \tau \rceil$. The node is deleted from the PP, if the time windows cannot be adjusted, i.e. $(a_i + s_i > \lceil \tau - 1 \rceil) \vee (b_i < \lceil \tau \rceil)$.

The branches resulting from the first violation $\{i_1, i_3, i_4\}$ in Figure 4 are given in Figure 5. In branch 1, the first treatment i_1 cannot start so early that the violation is prevented ($a_1 + s_1 > \tau$), therefore node $v_{1,l}$ is removed from graph G. In branch 2 however, treatment i_1 can start later than τ, and thus the earliest start time of $v_{1,l}$ is adjusted by setting $a_{1,l} = 8$. In branch 3, treatment i_3 can start so early that the violation is prevented, therefore the latest start time of $v_{3,l}$ is set to $b_{3,l} = 2$. Branch 4 acts equivalently to branch 2, and branch 5 equivalently to branch 1. Branch 6 is a special case. A later start is not possible for treatment i_4 and node $v_{4,l}$ would have to be removed from the PP. However, node $v_{4,l}$ has already been removed in branch 5 and thus the resulting PPs in branch 5 and branch 6 would be identical. Therefore, branching node 6 is not generated.

If a node has been deleted from the PPs, the columns $r \in \Omega'$ that contain it are also removed from the RMP. Columns are removed as well if the start time of a node is outside of the updated time windows. Note that our branching strategy reduces the capacity violation by one unit at a time. Therefore, if capacity is violated by more than one unit, multiple branching steps might be needed to fully eliminate the violation.

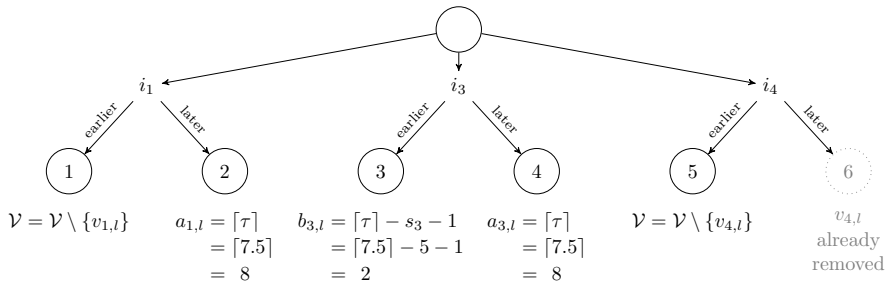

Figure 5: Resulting branches for branching on time windows to ensure location capacity

4.3 Cuts

We use three types of cuts in our BPC algorithm: limited-memory subset-row cuts to tighten the dual bound (§4.3.1), location capacity cuts to forbid infeasible assignments of treatments to locations (§4.3.2), and combinatorial Benders cuts to ensure feasibility for the location capacity and precedence relation constraints if branching on start time windows is disabled (§4.3.3).

4.3.1 Limited-memory subset-row cuts

To obtain a better dual bound subset-row cuts (SRC) introduced by Jepsen et al. (2008) have been shown to be very successful in this regard (Costa et al. 2019). We use a 3-SRC of the following form:

$$\sum_{r \in \Omega} h_{\mathcal{C},r} \cdot \lambda_r \leq 1 \tag{19}$$

with $\mathcal{C} \subseteq \mathcal{I}$, $|\mathcal{C}| = 3$ and $h_{\mathcal{C},r} = \lfloor \frac{1}{2} \cdot \sum_{i \in \mathcal{C}} \sum_{l \in \mathcal{L}_i} \alpha_{i,l,r} \rfloor$ takes value 1 if and only if tour r contains at least two treatments in $\mathcal{C}$. The intuition behind the 3-SRC is that, for every triplet of treatments $\mathcal{C}$, we know that at most one tour containing at leat two of these treatments can be part of an integer feasible solution.

SRCs are non-robust cuts, i.e. adding these cuts changes the structure of the PPs (Jepsen et al. 2008). Specifically, an additional resource associated with each cut must be stored in the labels, which impede efficient label dominance. Pecin et al. (2017) have introduced a limited-memory version of the SRCs (lm-SRCs), which counteracts the negative influence of the SRCs on the label dominance. When extending a partial path in the labeling algorithm, a previously visited treatment node in a set $\mathcal{C}$ is forgotten as soon as the path visits a node that is not in the corresponding SRC memory $\mathcal{M} \supseteq \mathcal{C}$. Thus, the lm-SRCs are a weaker version of the SRCs that are easier to handle.

4.3.2 Location-capacity cuts

We introduce a type of feasibility cuts, which we call location-capacity cut (LCC). For certain subsets of treatments $\mathcal{C} \subseteq \mathcal{I}$, we know that their treatments cannot be assigned to the same location l with violating its capacity because of their start time windows, i.e., regardless of how the start times are selected, the treatments always overlap. For a given location l and subset of treatments $\mathcal{C}$, the LCCs have the form:

$$\sum_{r \in \Omega} \sum_{i \in \mathcal{C}} \beta_{i,l,r} \cdot \lambda_r \leq Q_l \tag{20}$$

Figure 6 provides an example of three treatments $\{i_1, i_3, i_7\}$ that will always require location capacity of three between time points 5 and 7 if serviced in location l. Note that, while SRCs are added when CG terminates, LCCs are added at a branching node before CG is started. The reason is that new LCCs can only be found either directly at the root node or after branching has shrunk the start time windows. As start time windows shrink, the potential overlap grows. The LCCs are robust, i.e. the cuts do not change the structure of the PPs and duals associated with these cuts are simply subtracted when a node $v_{i,l}$ is visited with i belonging to $\mathcal{C}$.

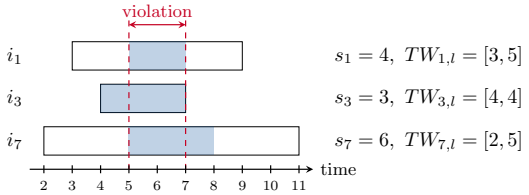

Figure 6: Subset of treatments forming valid location-capacity cuts for location l with capacity $Q_l \in \{1, 2\}$

4.3.3 Combinatorial Benders cuts

In the literature, the most common approach to address violations of previously relaxed constraints is to add cuts (cf. Bruglieri et al. 2019, Froger et al. 2019). Therefore, as an alternative to branching on start time windows, we propose adding combinatorial Benders cuts to forbid the reoccurrence of solutions violating the location capacity or the precedence relation constraints.

In this branch-and-check approach, our BPC algorithm corresponds to solving the Benders master problem and the cut separation to solving the Benders subproblem. Branch-and-check is a generalization of logic-based Benders decomposition, in which the subproblem is solved whenever a feasible solution of the Benders master problem is found, i.e. in our case, a cut is generated whenever an integer feasible solution is found that does not satisfy the location capacity and the precedence relation constraints of the original problem. For details on these topics, we refer the reader to Beck (2010), Hooker and Ottosson (2003), and Codato and Fischetti (2004).

To exclude an infeasible integer solution s, we forbid the reuse of the exact same arc set $\mathcal{A}_s$ used in s by adding the following cut:

$$\sum_{a \in \mathcal{A}} x_a \leq |\mathcal{A}_s| - 1 \quad \forall\, s \in \mathcal{S}^{\mathbb{N}} \tag{21}$$

where x_a is a binary variable indicating whether arc $a \in \mathcal{A}$ is used in s and $\mathcal{S}^{\mathbb{N}}$ is the set of all integer feasible solutions. Expressed in terms of the variables of model (8)–(12), the cut can be stated as follows:

$$\sum_{r \in \Omega'} \rho_{s,r} \cdot \lambda_r \leq |\mathcal{A}_s| - 1 \quad \forall\, s \in \mathcal{S}^{\mathbb{N}} \tag{22}$$

where $\rho_{s,r} \in \mathbb{N}_0$ is the number of arcs $a \in \mathcal{A}_s$ from an integer feasible solution $s \in \mathcal{S}^{\mathbb{N}}$ that column $r \in \Omega'$ contains. Note that the combinatorial Benders cuts are valid throughout the entire branching tree, and thus do not have to be reverted when moving up the branching tree.

4.4 MIP to correct infeasible solution

Even if an integer feasible solution of the RMP does not fulfill the location capacity (3) or precedence relation constraints (4), we can possibly transform this solution into a feasible one. We will only outline the idea here as prior tests revealed that the approach does neither yield a significant performance increase nor decrease on our test instances. A detailed description of the algorithm and the computational results can be found in Jungwirth (2020).

The general idea is that within a tour, the start time windows of the treatments $[a_i, b_i]$ may leave a leeway δ_i to shift start times such that the violations are eliminated. However, within each tour, the sequence of the nodes and the nodes itself must be prevailed. Otherwise, the costs of the tours could change and thus, the solution would no longer be guaranteed to be optimal for the current linear relaxation. To efficiently check if an infeasible integer solution can be corrected, we call a MIP every time we encounter such a solution. This MIP tries to assign start times to treatments such that the tours become feasible for the location capacity and precedence relation constraints. Our MIP is based on the continuous time formulation of the RCPSP presented in Artigues et al. (2003) and

the optimization is stopped once the first feasible solution is found as this already guarantees that a feasible schedule can be found for the proposed tours.

5 Computational study

This section presents the results of our computational study and is structured as follows. In the first part §5.1, we detail our problem instances, and in the subsequent three parts, we investigate various aspects of our BPC algorithm. Our algorithm contains multiple components and cross-testing all possible combinations is out of scope of this work. Instead, we determined a well-working combination of components for our algorithm in prior tests, and present computational results for which each component was individually turned off to show its added value. Therefore, unless stated otherwise, our algorithm always uses branching strategies (1) to (5) to reach integrality, adds lm-SRCs and LCCs to improve the bounds, and applies the MIP checker to attempt correcting infeasible solutions.

In §5.2, we compare the efficiency of branching on time windows and of using combinatorial Benders cuts to eliminate violations of the location capacity and precedence relation constraints. The superior approach will be part of the baseline algorithm for the subsequent tests. In §5.3, we evaluate the core algorithmic components of our algorithm by comparing the baseline algorithm to versions without the specific component. The evaluated components are: branching strategies (2)-(4), lm-SRCs, and LCCs. In §5.4, we investigate the robustness of our algorithm against varying input data. Specifically, we modify the capacity limit of the locations and adjust the number of precedence relations within the instances.

The notation that we use in the computational study is summarized in Table 2. Our algorithms were coded in JAVA using Amazon Corretto 11 as JDK and executed on a Windows 10 platform employing an Intel Core i7-10510U CPU @ 2.30GHz with 16 GB of RAM. We used Gurobi 9.0.0 to solve the RMP and the checker MIP.

Table 2: Notation used in the computational study

B	Number of hospital buildings
F	Number of floors per building
gap	Optimality gap [%]
$\lvert\mathcal{I}\rvert$	Number of treatments
$\mathcal{K}^{\text{used}}$	Number of therapist serving patients in the best feasible solution
LB	Lower bound (dual bound)
$N^{\text{Br}}_{(\cdot)}$	Number of branchings for branching strategy (1)-(7)
N^{Br}_{Σ}	Sum of all branchings performed
$N^{\text{Branch}}_{\text{TW prec}}$	Number of time window branchings due to precedence violation
$N^{\text{Branch}}_{\text{TW loc}}$	Number of time window branchings due to location capacity violation
$N^{\text{Cut}}_{\text{combi}}$	Number of combinatorial cuts
$N^{\text{Cut}}_{\text{LCC}}$	Number of location capacity cuts
$N^{\text{Cut}}_{\text{SR}}$	Number of lm-SRCs
N^{infeas}	Number of instances proven to be infeasible
$N^{\text{inst}}_{\text{Branch}}$	Number of instances for which time windows branching was used
N^{iter}	Number of processed branching nodes
N^{non}	Number of instances, for which no feasible solution was found for the original problem
N^{opt}	Number of instances solved to optimality
time	Runtime [sec.]
UB	Upper bound (best feasible solution cost)

5.1 Instances

To the best of our knowledge, only Gartner et al. (2018) have studied the ThSRP. The authors worked on eight instances, three of which matched the data from a German hospital. As we needed considerably more data to reliably evaluate our algorithm and to account for different types of hospitals, we created a new instance set which is available in JavaScript Object Notation (json) format at

https://www.gerad.ca/~guyd/bench-en.html. We consider three hospital layouts varying in the number of buildings and floors in each building. The therapy center is located at one of the buildings. We consider three demand scenarios with 40, 80, and 120 treatments per day. The daily scheduling problem decomposes into a morning and afternoon part and the number of treatments in each part is divided by approximately two. There are three hierarchical levels of qualification. Therapists work either a regular shift or a short shift. During short shifts, therapists are only available either in the morning or the afternoon. For each combination of layout and demand scenario, we generated five instances with a morning and afternoon part, which results in $3 \cdot 3 \cdot 5 \cdot 2 = 90$ instances in total. Further details of the instance generation such as how we determine the number of outpatients, the width of the time windows, the treatment durations and the precedence relations can be found in Jungwirth et al. (2020). For our numerical test, we set the location costs $c_{i,l}^{\text{location}}$ for treating i in location $l \in \mathcal{L}_i$ to the travel time between the preferred location and l.

5.2 Branching on time windows vs. combinatorial cuts

In this part of the computational study, we compare how well branching on start time windows and generating combinatorial cuts function to ensure the satisfaction of the relaxed location capacities and precedence relations. Tables 3 and 4 summarize the results for branching on time windows and imposing combinatorial cuts, respectively. Each line represents the combination of a demand scenario ($|\mathcal{I}|$) and hospital layout (combination of number of buildings B and number of floors per building F). Average values are aggregated over ten instances. Note that the ThSRP decomposes into a morning and an afternoon problem with approximately half the number of treatments and thus, instead of five instances, we solve ten instances. For each instance, a time limit of one hour was set. We display the number of optimal solutions N^{opt}, the number of non-feasible solution found N^{non}, and we provide information about the bounds, runtime and number of iterations, cuts, and the number of time window branchings. N^{non} is the number of runs in which we could not find feasible solutions within the given runtime limit, and this number should not be confused with the number of infeasible solutions N^{infeas}.

Table 3: Results for branching on start time windows

$\|\mathcal{I}\|$	B	F	N^{opt}	N^{non}	$\mathcal{K}^{\text{used}}$	LB	UB	gap	time	N^{iter}	$N^{\text{Cut}}_{\text{SR}}$	$N^{\text{Cut}}_{\text{LCC}}$	$N^{\text{Branch}}_{\text{TW prec}}$	$N^{\text{Branch}}_{\text{TW loc}}$
40	1	6	10	0	4.90	57.70	57.70	0.00	119.6	1051.9	1139.6	211.0	0.1	292.1
	2	3	10	0	4.60	52.30	52.30	0.00	4.6	5.4	40.7	0.4	0.0	2.4
	6	1	10	0	4.60	50.40	50.40	0.00	9.2	18.2	86.9	3.7	0.3	6.6
80	1	6	10	0	7.90	85.70	85.70	0.00	116.7	129.1	409.1	0.0	22.8	0.0
	2	3	9	0	7.78	91.56	91.70	0.15	367.7	33.3	345.6	0.0	0.1	0.0
	6	1	10	0	8.10	85.30	85.30	0.00	7.4	1.4	20.0	0.0	0.1	0.0
120	1	6	9	1	11.00	122.70	122.78	0.00	601.4	78.0	938.6	0.0	0.0	0.0
	2	3	9	0	11.00	116.21	116.40	0.17	422.7	53.2	724.7	0.0	0.1	0.0
	6	1	10	0	11.00	116.70	116.70	0.00	124.0	10.5	172.0	0.0	0.0	0.0
Avg.			87/90	1/90	7.88	86.51	86.55	0.04	197.03	153.44	430.80	23.90	2.61	33.46

Table 4: Results for using combinatorial cuts

$\|\mathcal{I}\|$	B	F	N^{opt}	N^{non}	$\mathcal{K}^{\text{used}}$	LB	UB	gap	time	N^{iter}	$N^{\text{Cut}}_{\text{SR}}$	$N^{\text{Cut}}_{\text{LCC}}$	$N^{\text{Cut}}_{\text{combi}}$
40	1	6	9	0	4.67	57.45	57.80	0.49	364.0	515.6	607.4	1.6	208.0
	2	3	10	0	4.50	52.30	52.30	0.00	5.8	7.6	33.9	1.7	10.3
	6	1	8	0	4.38	50.10	50.40	0.52	721.6	374.2	435.2	94.6	393.3
80	1	6	9	0	8.00	85.60	85.70	0.13	387.5	167.3	345.8	0.0	89.4
	2	3	8	0	7.75	91.46	92.20	0.76	727.4	93.1	520.3	0.0	18.2
	6	1	10	0	8.10	85.30	85.30	0.00	8.7	1.7	17.9	0.0	4.8
120	1	6	9	1	11.00	122.70	122.78	0.00	658.8	71.8	852.2	0.0	4.9
	2	3	9	0	11.11	116.21	116.40	0.17	444.8	43.7	625.3	0.0	4.4
	6	1	10	0	11.00	116.70	116.70	0.00	178.1	10.6	173.2	0.0	4.7
Avg.			82/90	1/90	7.83	86.42	86.62	0.23	388.52	142.84	401.24	10.88	82.00

Branching on time windows performs better than adding combinatorial cuts. Indeed, with the former strategy, five additional instances could be solved to optimality and the average optimality gap is 0.19 lower for time window branching, which is a relative reduction of 82.6%. Interestingly, branching on time windows required less time while performing more iterations, which we interpret as that combinatorial cuts add significant complexity to the RMP that cannot be justified by its performance. Therefore, branching on time windows will serve as the baseline algorithm in the remainder of this computational study. The individual results for each of the 90 instances are given in the Appendix in Tables 12 and 13 where we detail the results for branching on time windows and for using combinatorial cuts, respectively.

5.3 Value of algorithmic components

We investigate the performance improvements achieved by three of our core algorithmic components: (1) additional branching strategies, (2) limited-memory subset-row cuts (lm-SRCs), and (3) location capacity cuts (LCCs). To measure the benefit of using each specific component, we run the baseline algorithm on all instances with and without the specific component.

Value of additional branchings. We have introduced three additional branching strategies to impact the solution on higher levels before branching on arcs: strategy (2) branching on the types of therapists to perform a treatment (see §4.2.2), strategy (3) branching on the possible service locations of treatments (see §4.2.3), and strategy (4) branching on aggregated arc-sets (see §4.2.4). Tables 5 and 6 display the numbers of how often each of the seven branching strategies was used in the baseline run and if the additional branchings were turned off, respectively.

In the baseline run, arc branching was never used as branchings (1)-(4) were sufficient to generate integer feasible solutions. When branchings (2)-(4) are switched off, the total number of branchings N^{Br}_{Σ} increased by 133% and the runtime increased by 18.4%. To our surprise, without additional branchings a feasible solution was found for the one instance for which the baseline algorithm could not find a feasible solution. However, this seems to be a coincident as there is also an instance less for which optimality could be proven.

Table 5: Number of branchings (baseline)

$\lvert\mathcal{I}\rvert$	N^{opt}	N^{non}	gap	time	branching strategies							
					$N^{\text{Br}}_{(1)}$	$N^{\text{Br}}_{(2)}$	$N^{\text{Br}}_{(3)}$	$N^{\text{Br}}_{(4)}$	$N^{\text{Br}}_{(5)}$	$N^{\text{Br}}_{(6)}$	$N^{\text{Br}}_{(7)}$	N^{Br}_{Σ}
40	30	0	0.00	44.47	3.3	8.17	1.2	0.13	0.0	0.13	100.37	113.3
80	29	0	0.05	163.93	2.6	18.90	0.5	2.23	0.0	7.67	0.00	31.9
120	28	1	0.06	382.70	7.4	37.30	0.0	0.07	0.0	0.03	0.00	44.8
Avg.	87/90	1/90	0.04	197.03	4.43	21.46	0.57	0.81	0.0	2.61	33.46	63.33

Table 6: Number of branchings (no additional branchings)

$\lvert\mathcal{I}\rvert$	N^{opt}	N^{non}	gap	time	branching strategies							
					$N^{\text{Br}}_{(1)}$	$N^{\text{Br}}_{(2)}$	$N^{\text{Br}}_{(3)}$	$N^{\text{Br}}_{(4)}$	$N^{\text{Br}}_{(5)}$	$N^{\text{Br}}_{(6)}$	$N^{\text{Br}}_{(7)}$	N^{Br}_{Σ}
40	29	0	0.21	126.70	2.03				192.83	0.17	199.57	394.60
80	29	0	0.03	136.10	2.37				9.10	1.03	0.00	12.50
120	28	0	0.10	437.47	7.57				28.90	0.00	0.00	36.47
Avg.	86/90	0/90	0.11	233.42	3.99	0.0	0.0	0.0	76.94	0.4	66.52	147.86
Δ	-1	-1	+0.07	+36.39	-0.44	-21.46	-0.57	-0.81	+76.94	-2.21	+33.06	+84.53

Value of lm-SRCs. The results for evaluating the lm-SRCs are given in Table 7. Without lm-SRCs, 10 instances less could be solved to optimality, no feasible solution was found for 4 additional instances, and the runtime increased by a factor of 3.62. Our results confirm the literature and emphasize that lm-SRCs should also be applied to the ThSRP.

Table 7: Value of limited-memory subset-row cuts

$\|\mathcal{I}\|$	baseline					no lm-SRCs			
	N^{opt}	N^{non}	gap	time	$N^{\text{Cut}}_{\text{SR}}$	N^{opt}	N^{non}	gap	time
40	30	0	0.00	44.47	422.40	30	0	0.00	124.73
80	29	0	0.05	163.93	258.23	28	0	0.11	356.90
120	28	1	0.06	382.70	611.77	19	5	1.17	1660.23
Avg.	87/90	1/90	0.04	197.03	430.8	77/90	5/90	0.43	713.95
Δ						-10	+4	+0.39	+516.92

Value of LCCs. The value of adding LCCs to the RMP are shown in Table 8. The solution quality remains and the difference in runtime is insignificant. However, on average only 23.9 cuts were added per instance.

Table 8: Value of location capacity cuts

$\|\mathcal{I}\|$	baseline					no LCCs			
	N^{opt}	N^{non}	gap	time	$N^{\text{Cut}}_{\text{LCC}}$	N^{opt}	N^{non}	gap	time
40	30	0	0.00	44.47	71.7	30	0	0.00	44.00
80	29	0	0.05	163.93	0.0	29	0	0.05	163.33
120	28	1	0.06	382.70	0.0	28	1	0.06	384.43
Avg.	87/90	1/90	0.04	197.03	23.9	87/90	1/90	0.04	197.25
Δ						+0	+0	+0.00	+0.22

In a separate run, we decreased the location capacity of the therapy centers by one unit such that the instances become more restrictive. Note that reducing the location capacity (Q_l is replaced by $Q_l - 1$) leads to higher probability of encountering infeasible solutions. For the case of reduced capacity, Table 9 shows that adding LCCs yields positive effects. Without LCCs, runtime increases by a factor of 2.28, and for 7 fewer instances infeasibility could be proved.

Table 9: Value of location capacity cuts with reduced location capacity ($Q_l - 1$)

$\|\mathcal{I}\|$	baseline ($Q_l - 1$)						no LCCs ($Q_l - 1$)				
	N^{opt}	N^{infeas}	N^{non}	gap	time	$N^{\text{Cut}}_{\text{LCC}}$	N^{opt}	N^{infeas}	N^{non}	gap	time
40	15	14	0	0.15	129.83	573.3	15	9	5	0.23	746.40
80	25	4	0	0.06	160.83	0.4	25	2	2	0.06	405.47
120	28	0	1	0.06	381.27	0.0	28	0	1	0.06	381.87
Avg.	68/90	18/90	1/90	0.09	223.98	191.23	68/90	11/90	8/90	0.12	511.25
Δ							+0	-7	+7	+0.03	+287.27

5.4 Influence of instance properties

To investigate how our algorithm behaves if input data changes, we (a) modify the capacity of the TCs, and (b) gradually reduce the number of precedence relations. To isolate the effects of modified inputs, we specifically evaluate the subsets of instances in which branching on time windows was applied in the baseline runs (§5.2) to eliminate violations of the location capacity or precedence relation constraints. For the location capacity, these are instances 5, 7, 11, 13, 21 and 29, all with approximately 20 customers (cf. Table 12 in the Appendix). For the precedence relations, these are instances 6, 27, 29, 31, 43, 55 and 79 with approximately 20 customers (3x), 40 customers (3x), and 60 customers (1x). Note that all instances include both location capacity and precedence constraints. However, only in one instance (number 29) we had to apply both types of time window branching to enforce these two constraints.

Influence of location capacity. Table 10 displays the results for modified location capacities. If capacity is increased by one unit ($Q_l + 1$), the runtime, the number of branchings on time windows, and the number of generated LCCs decreased substantially compared to the baseline. All instances were solved to optimality in 0.02% of the time required to solve the baseline. Increasing the capacity limit by another unit ($Q_l + 2$) changes the results only in details. Capacity is no longer restrictive in any instance, and the runtime and solution quality remains compared to $Q_l + 1$. However, if capacity is reduced by one unit ($Q_l - 1$), complexity increases extensively.

Table 10: Results for changed location capacity

Setup	N^{infeas}	LB	UB	gap	time	N^{iter}	$N^{\text{Branch}}_{\text{TW loc}}$	$N^{\text{inst}}_{\text{Branch}}$	$N^{\text{Cut}}_{\text{LCC}}$	N^{MIP}
$Q_l - 1$	5	79.00	81.00	0.0247	601.83	6663.0	2836.67	1	2805.5	0.50
baseline	0	56.67	56.67	0.0000	213.83	1788.5	501.83	6	358.5	35.67
$Q_l + 1$	0	55.33	55.33	0.0000	4.50	6.5	0.17	1	0.0	0.17
$Q_l + 2$	0	55.33	55.33	0.0000	4.50	6.5	0.00	0	0.0	0.00

Influence of precedence relations. To evaluate the influence of the number of precedence relations, we generated two additional scenarios: one in which we reduce the number of precedence relations by 50% compared to the baseline, and another one, in which we remove all precedence relations. The −50%-case is generated as follows. For each instance, we iterate through the list of treatments sorted by their id and only keep every other precedence relation. The baseline has 5.46 precedence relations per instance on average, and the −50%-scenario 2.48 (observe that more than 50% of the relations are omitted when there is an even number of relations). The results are displayed in Table 11. Only small differences between the baseline and the −50%-scenario can be observed for the solution quality and runtime. The reason is that in instance 31 a precedence relation exists, which is extremely difficult to resolve. In fact, over 90% of the time to solve the 7 instances is spent for this instance and it accounts for over 97% of the time window branchings. If all precedence relations are removed, the problem becomes an easy one, which can be solved in 0.07% of the runtime compared to the baseline.

Table 11: Results for changed number of precedence relations

Setup	LB	UB	gap	time	N^{iter}	$N^{\text{Branch}}_{\text{TW prec}}$	$N^{\text{inst}}_{\text{Branch}}$	N^{MIP}
baseline	73.43	73.43	0.0	147.00	176.71	33.57	7	0.43
precedence -50%	73.29	73.29	0.0	143.43	174.00	32.71	2	0.43
precedence -100%	73.14	73.14	0.0	10.57	3.86	0.00	0	0.00

Although all instances contain location capacities and precedence relations, we observe that if either one of the latter constraints is relaxed, the problem becomes relatively easy to solve. Our explanation is that the probability of encountering a violation of one of these constraints in an integer solution is already fairly low (7.78% for a violation of the capacity constraints, 6.67% for a violation of the precedence constraints), and if one of these constraints is relaxed a violation becomes even less likely. In fact, in our tests only for one instance we needed to branch to correct location capacity violations after the precedence constraints were relaxed. Nevertheless, even if these constraints are binding only occasionally, once they are binding, the problem becomes significantly harder to solve.

6 Conclusion

We addressed the ThSRP and modeled it as a variant of the VRPTW with heterogenous fleet, operations and resource synchronizations, and flexible service locations. We presented a BPC algorithm that solves realistic hospital instances in reasonable times. Branching on start time windows and combinatorial Benders cuts were used to cope with the location capacity and precedence relation constraints, and branching on time windows was superior for our tests. In total, we presented seven branching strategies serving different purposes and breaking structures on different levels of aggregation. Our

computational results showed that the proposed BPC algorithm with time window branching succeeds to solve to optimality instances involving up to 120 treatments (resulting in two subproblems with about half of the treatments) in less than 1200 seconds (2 times 600 seconds) on average.

Considering location capacity is relevant for real-world applications, also outside of the hospital context. We hope that branching on time windows or other resource windows will help better solving problems involving e.g., capacitated charging and fuel stations, parking lots, or parcel delivery boxes. Consequently, developing BPC algorithms similar to the one proposed in this paper for such applications seems an interesting research avenue.

Appendix

In the following, we state the individual results for each of the 90 instances. Table 12 corresponds to Table 3 and Table 13 corresponds to Table 4. The instance name is composed of the number of customers in the daily scheduling problem (i), the number of buildings (b), the number of floors (f), the number of the problem setting for a specific combination of treatments, buildings and floors (v), and the information if the instance represents either the morning (m) or afternoon (a) part of the planning problem. N^{MIP} states how often the MIP to correct infeasible integer solutions was called.

Table 12: Detailed results for branching on start time windows

id	name	$\lvert\mathcal{I}\rvert$	$\mathcal{K}^{\text{used}}$	LB	UB	gap	time	N^{iter}	$N^{\text{Cut}}_{\text{SR}}$	$N^{\text{Cut}}_{\text{LCC}}$	$N^{\text{Branch}}_{\text{TW prec}}$	$N^{\text{Branch}}_{\text{TW loc}}$	N^{MIP}
1	i040-b1-f6-v1-m	17	4	56.00	56	*	1	0	0	0	0	0	0
2	i040-b1-f6-v1-a	23	5	71.00	71	*	7	8	64	0	0	0	0
3	i040-b1-f6-v2-m	22	5	62.00	62	*	2	0	7	0	0	0	0
4	i040-b1-f6-v2-a	18	4	49.00	49	*	1	0	7	0	0	0	0
5	i040-b1-f6-v3-m	19	7	70.00	70	*	1160	10468	10985	2110	0	2918	214
6	i040-b1-f6-v3-a	21	5	47.00	47	*	3	2	13	0	1	0	0
7	i040-b1-f6-v4-m	21	5	60.00	60	*	16	38	299	0	0	3	0
8	i040-b1-f6-v4-a	19	4	49.00	49	*	2	0	4	0	0	0	0
9	i040-b1-f6-v5-m	21	5	51.00	51	*	3	3	17	0	0	0	0
10	i040-b1-f6-v5-a	19	5	62.00	62	*	1	0	0	0	0	0	0
11	i040-b2-f3-v1-m	23	5	50.00	50	*	25	31	249	4	0	16	0
12	i040-b2-f3-v1-a	17	4	41.00	41	*	1	0	0	0	0	0	0
13	i040-b2-f3-v2-m	17	4	44.00	44	*	5	16	66	0	0	8	0
14	i040-b2-f3-v2-a	23	5	57.00	57	*	2	0	4	0	0	0	0
15	i040-b2-f3-v3-m	19	4	53.00	53	*	2	0	0	0	0	0	0
16	i040-b2-f3-v3-a	21	6	71.00	71	*	2	0	13	0	0	0	0
17	i040-b2-f3-v4-m	16	4	44.00	44	*	1	0	0	0	0	0	0
18	i040-b2-f3-v4-a	24	5	51.00	51	*	5	7	73	0	0	0	0
19	i040-b2-f3-v5-m	18	4	53.00	53	*	1	0	2	0	0	0	1
20	i040-b2-f3-v5-a	22	5	59.00	59	*	2	0	0	0	0	0	0
21	i040-b6-f1-v1-m	24	6	77.00	77	*	67	156	716	32	0	55	0
22	i040-b6-f1-v1-a	16	4	44.00	44	*	1	0	0	0	0	0	0
23	i040-b6-f1-v2-m	24	6	54.00	54	*	3	2	7	0	0	0	0
24	i040-b6-f1-v2-a	16	3	34.00	34	*	1	0	0	0	0	0	0
25	i040-b6-f1-v3-m	22	5	59.00	59	*	2	0	0	0	0	0	0
26	i040-b6-f1-v3-a	18	4	53.00	53	*	2	0	8	0	0	0	0
27	i040-b6-f1-v4-m	23	5	53.00	53	*	3	2	0	0	1	0	0
28	i040-b6-f1-v4-a	17	4	46.00	46	*	2	0	16	0	0	0	0
29	i040-b6-f1-v5-m	23	5	39.00	39	*	10	22	121	5	2	11	0
30	i040-b6-f1-v5-a	17	4	45.00	45	*	1	0	1	0	0	0	0
31	i080-b1-f6-v1-m	34	7	79.00	79	*	938	1195	2978	0	228	0	2
32	i080-b1-f6-v1-a	46	8	91.00	91	*	10	0	10	0	0	0	0
33	i080-b1-f6-v2-m	49	9	86.00	86	*	79	33	351	0	0	0	0
34	i080-b1-f6-v2-a	31	6	78.00	78	*	16	20	171	0	0	0	0
35	i080-b1-f6-v3-m	40	8	88.00	88	*	49	16	286	0	0	0	0
36	i080-b1-f6-v3-a	40	10	96.00	96	*	19	11	93	0	0	0	0
37	i080-b1-f6-v4-m	36	8	75.00	75	*	27	12	113	0	0	0	0
38	i080-b1-f6-v4-a	44	8	96.00	96	*	18	4	60	0	0	0	0
39	i080-b1-f6-v5-m	45	9	94.00	94	*	6	0	18	0	0	0	0
40	i080-b1-f6-v5-a	35	6	74.00	74	*	5	0	11	0	0	0	0
41	i080-b2-f3-v1-m	40	9	89.00	89	*	10	8	18	0	0	0	0

Detailed results for branching on start time windows (continued)

id	name	$\|\mathcal{I}\|$	$\mathcal{K}^{\text{used}}$	LB	UB	gap	time	N^{iter}	$N^{\text{Cut}}_{\text{SR}}$	$N^{\text{Cut}}_{\text{LCC}}$	$N^{\text{Branch}}_{\text{TW prec}}$	$N^{\text{Branch}}_{\text{TW loc}}$	N^{MIP}
42	i080-b2-f3-v1-a	40	7	80.00	80	*	4	0	0	0	0	0	0
43	i080-b2-f3-v2-m	48	8	92.00	92	*	22	6	59	0	1	0	0
44	i080-b2-f3-v2-a	32	6	79.00	79	*	5	3	40	0	0	0	0
45	i080-b2-f3-v3-m	37	7	92.00	92	*	4	0	5	0	0	0	0
46	i080-b2-f3-v3-a	43	8	108.00	108	*	11	2	32	0	0	0	0
47	i080-b2-f3-v4-m	32	7	70.00	70	*	7	4	56	0	0	0	0
48	i080-b2-f3-v4-a	48	9	102.00	102	*	7	0	3	0	0	0	0
49	i080-b2-f3-v5-m	42	9	109.00	109	*	7	2	20	0	0	0	0
50	i080-b2-f3-v5-a	38	—	94.57	96	1.49%	3600	308	3223	0	0	0	0
51	i080-b6-f1-v1-m	34	7	78.00	78	*	11	3	37	0	0	0	0
52	i080-b6-f1-v1-a	46	10	92.00	92	*	7	2	24	0	0	0	0
53	i080-b6-f1-v2-m	31	6	59.00	59	*	5	0	7	0	0	0	0
54	i080-b6-f1-v2-a	49	9	108.00	108	*	11	2	24	0	0	0	0
55	i080-b6-f1-v3-m	44	9	84.00	84	*	8	2	20	0	1	0	1
56	i080-b6-f1-v3-a	36	8	84.00	84	*	5	2	28	0	0	0	0
57	i080-b6-f1-v4-m	39	7	79.00	79	*	6	0	0	0	0	0	0
58	i080-b6-f1-v4-a	41	10	99.00	99	*	7	3	45	0	0	0	0
59	i080-b6-f1-v5-m	35	7	77.00	77	*	5	0	6	0	0	0	0
60	i080-b6-f1-v5-a	45	8	93.00	93	*	9	0	9	0	0	0	0
61	i120-b1-f6-v1-m	60	10	124.00	124	*	337	103	727	0	0	0	0
62	i120-b1-f6-v1-a	60	11	128.00	128	*	12	0	17	0	0	0	0
63	i120-b1-f6-v2-m	58	10	110.00	110	*	150	23	401	0	0	0	0
64	i120-b1-f6-v2-a	62	12	118.00	118	*	129	16	221	0	0	0	0
65	i120-b1-f6-v3-m	48	9	105.00	105	*	733	342	3373	0	0	0	0
66	i120-b1-f6-v3-a	72	14	153.00	153	*	199	41	720	0	0	0	0
67	i120-b1-f6-v4-m	56	11	113.00	113	*	768	104	1478	0	0	0	0
68	i120-b1-f6-v4-a	64	—	121.98		—	3600	135	2324	0	0	0	0
69	i120-b1-f6-v5-m	59	10	120.00	120	*	35	4	58	0	0	0	0
70	i120-b1-f6-v5-a	61	12	134.00	134	*	51	12	67	0	0	0	0
71	i120-b2-f3-v1-m	63	11	126.00	126	*	15	0	0	0	0	0	0
72	i120-b2-f3-v1-a	57	11	140.00	140	*	92	21	195	0	0	0	0
73	i120-b2-f3-v2-m	61	12	108.00	108	*	177	35	620	0	0	0	0
74	i120-b2-f3-v2-a	59	12	100.00	100	*	102	37	464	0	0	0	0
75	i120-b2-f3-v3-m	65	12	115.00	115	*	101	13	217	0	0	0	0
76	i120-b2-f3-v3-a	55	10	96.00	96	*	18	2	29	0	0	0	0
77	i120-b2-f3-v4-m	56	9	113.00	113	*	26	4	58	0	0	0	0
78	i120-b2-f3-v4-a	64	11	135.00	135	*	51	3	40	0	0	0	0
79	i120-b2-f3-v5-m	61	11	120.00	120	*	45	8	117	0	1	0	0
80	i120-b2-f3-v5-a	59	—	109.10	111	1.71%	3600	409	5507	0	0	0	0
81	i120-b6-f1-v1-m	55	10	111.00	111	*	21	4	57	0	0	0	0
82	i120-b6-f1-v1-a	65	12	128.00	128	*	14	0	10	0	0	0	0
83	i120-b6-f1-v2-m	72	11	119.00	119	*	381	30	534	0	0	0	0
84	i120-b6-f1-v2-a	48	9	95.00	95	*	13	4	57	0	0	0	0
85	i120-b6-f1-v3-m	73	12	126.00	126	*	544	26	448	0	0	0	0
86	i120-b6-f1-v3-a	47	11	103.00	103	*	13	7	43	0	0	0	0
87	i120-b6-f1-v4-m	54	10	106.00	106	*	21	2	32	0	0	0	0
88	i120-b6-f1-v4-a	66	11	117.00	117	*	49	5	70	0	0	0	0
89	i120-b6-f1-v5-m	63	12	143.00	143	*	158	24	417	0	0	0	0
90	i120-b6-f1-v5-a	57	12	119.00	119	*	26	3	52	0	0	0	0

Table 13: Detailed results for using combinatorial cuts

id	name	$\|\mathcal{I}\|$	$\mathcal{K}^{\text{used}}$	LB	UB	gap	time	N^{iter}	$N^{\text{Cut}}_{\text{SR}}$	$N^{\text{Cut}}_{\text{LCC}}$	$N^{\text{Cut}}_{\text{combi}}$	N^{MIP}
1	i040-b1-f6-v1-m	17	4	56.00	56	*	1	0	0	0	2	0
2	i040-b1-f6-v1-a	23	5	71.00	71	*	7	8	63	0	6	0
3	i040-b1-f6-v2-m	22	5	62.00	62	*	2	0	7	0	2	0
4	i040-b1-f6-v2-a	18	4	49.00	49	*	1	0	7	0	1	0
5	i040-b1-f6-v3-m	19	—	67.50	71	4.93%	3600	5108	5657	16	2043	1
6	i040-b1-f6-v3-a	21	5	47.00	47	*	5	2	22	0	8	0
7	i040-b1-f6-v4-m	21	5	60.00	60	*	18	35	291	0	8	0
8	i040-b1-f6-v4-a	19	4	49.00	49	*	2	0	10	0	3	0
9	i040-b1-f6-v5-m	21	5	51.00	51	*	3	3	17	0	2	0
10	i040-b1-f6-v5-a	19	5	62.00	62	*	1	0	0	0	5	0
11	i040-b2-f3-v1-m	23	5	50.00	50	*	23	27	170	15	28	0
12	i040-b2-f3-v1-a	17	4	41.00	41	*	1	0	0	0	5	0

Detailed results for using combinatorial cuts (continued)

id	name	$\|\mathcal{I}\|$	$\mathcal{K}^{\text{used}}$	LB	UB	gap	time	N^{iter}	$N^{\text{Cut}}_{\text{SR}}$	$N^{\text{Cut}}_{\text{LCC}}$	$N^{\text{Cut}}_{\text{combi}}$	N^{MIP}
13	i040-b2-f3-v2-m	17	4	44.00	44	*	14	39	58	2	45	1
14	i040-b2-f3-v2-a	23	5	57.00	57	*	2	0	2	0	7	0
15	i040-b2-f3-v3-m	19	4	53.00	53	*	2	0	2	0	3	0
16	i040-b2-f3-v3-a	21	5	71.00	71	*	3	4	18	0	3	0
17	i040-b2-f3-v4-m	16	4	44.00	44	*	2	0	16	0	3	0
18	i040-b2-f3-v4-a	24	5	51.00	51	*	6	6	50	0	2	0
19	i040-b2-f3-v5-m	18	4	53.00	53	*	2	0	7	0	2	1
20	i040-b2-f3-v5-a	22	5	59.00	59	*	3	0	16	0	5	0
21	i040-b6-f1-v1-m	24	—	75.00	77	2.6%	3600	2023	3604	519	1491	1
22	i040-b6-f1-v1-a	16	4	44.00	44	*	1	0	0	0	3	0
23	i040-b6-f1-v2-m	24	6	54.00	54	*	3	0	11	0	2	0
24	i040-b6-f1-v2-a	16	3	34.00	34	*	1	0	0	0	5	0
25	i040-b6-f1-v3-m	22	5	59.00	59	*	1	0	0	0	4	0
26	i040-b6-f1-v3-a	18	4	53.00	53	*	2	0	8	0	2	0
27	i040-b6-f1-v4-m	23	5	53.00	53	*	3	2	0	0	4	0
28	i040-b6-f1-v4-a	17	4	46.00	46	*	2	0	11	0	4	0
29	i040-b6-f1-v5-m	23	—	38.00	39	2.56%	3600	1717	717	427	2409	1
30	i040-b6-f1-v5-a	17	4	45.00	45	*	3	0	1	0	9	0
31	i080-b1-f6-v1-m	34	—	78.00	79	1.27%	3600	1582	2370	0	869	5
32	i080-b1-f6-v1-a	46	8	91.00	91	*	12	0	12	0	4	0
33	i080-b1-f6-v2-m	49	9	86.00	86	*	102	33	348	0	1	0
34	i080-b1-f6-v2-a	31	6	78.00	78	*	22	20	177	0	2	0
35	i080-b1-f6-v3-m	40	8	88.00	88	*	60	16	285	0	2	0
36	i080-b1-f6-v3-a	40	10	96.00	96	*	24	11	97	0	6	0
37	i080-b1-f6-v4-m	36	8	75.00	75	*	21	7	80	0	2	0
38	i080-b1-f6-v4-a	44	8	96.00	96	*	22	4	64	0	4	0
39	i080-b1-f6-v5-m	45	9	94.00	94	*	7	0	14	0	3	0
40	i080-b1-f6-v5-a	35	6	74.00	74	*	5	0	11	0	1	0
41	i080-b2-f3-v1-m	40	9	89.00	89	*	13	8	18	0	1	0
42	i080-b2-f3-v1-a	40	7	80.00	80	*	6	0	1	0	6	0
43	i080-b2-f3-v2-m	48	—	91.09	97	6.09%	3600	664	2240	0	145	0
44	i080-b2-f3-v2-a	32	6	79.00	79	*	8	3	44	0	2	0
45	i080-b2-f3-v3-m	37	7	92.00	92	*	7	0	17	0	1	0
46	i080-b2-f3-v3-a	43	8	108.00	108	*	14	2	33	0	3	0
47	i080-b2-f3-v4-m	32	7	70.00	70	*	9	4	56	0	1	0
48	i080-b2-f3-v4-a	48	9	102.00	102	*	9	0	7	0	3	0
49	i080-b2-f3-v5-m	42	9	109.00	109	*	8	2	20	0	6	0
50	i080-b2-f3-v5-a	38	—	94.50	96	1.56%	3600	248	2767	0	14	0
51	i080-b6-f1-v1-m	34	7	78.00	78	*	7	0	17	0	3	0
52	i080-b6-f1-v1-a	46	10	92.00	92	*	10	2	28	0	5	0
53	i080-b6-f1-v2-m	31	6	59.00	59	*	6	0	8	0	1	0
54	i080-b6-f1-v2-a	49	9	108.00	108	*	13	2	23	0	3	0
55	i080-b6-f1-v3-m	44	9	84.00	84	*	15	8	18	0	19	1
56	i080-b6-f1-v3-a	36	8	84.00	84	*	6	2	35	0	3	0
57	i080-b6-f1-v4-m	39	7	79.00	79	*	7	0	0	0	8	0
58	i080-b6-f1-v4-a	41	10	99.00	99	*	9	3	39	0	2	0
59	i080-b6-f1-v5-m	35	7	77.00	77	*	5	0	6	0	1	0
60	i080-b6-f1-v5-a	45	8	93.00	93	*	9	0	5	0	3	0
61	i120-b1-f6-v1-m	60	10	124.00	124	*	524	119	857	0	17	0
62	i120-b1-f6-v1-a	60	11	128.00	128	*	16	0	17	0	1	0
63	i120-b1-f6-v2-m	58	10	110.00	110	*	189	23	410	0	2	0
64	i120-b1-f6-v2-a	62	12	118.00	118	*	155	16	223	0	7	0
65	i120-b1-f6-v3-m	48	9	105.00	105	*	516	213	2189	0	6	0
66	i120-b1-f6-v3-a	72	14	153.00	153	*	267	41	719	0	1	0
67	i120-b1-f6-v4-m	56	11	113.00	113	*	1210	168	1876	0	7	0
68	i120-b1-f6-v4-a	64	—	121.95		—	3600	122	2102	0	0	0
69	i120-b1-f6-v5-m	59	10	120.00	120	*	55	6	84	0	2	0
70	i120-b1-f6-v5-a	61	12	134.00	134	*	56	10	45	0	6	0
71	i120-b2-f3-v1-m	63	11	126.00	126	*	15	0	0	0	6	0
72	i120-b2-f3-v1-a	57	11	140.00	140	*	107	27	182	0	20	0
73	i120-b2-f3-v2-m	61	12	108.00	108	*	269	35	617	0	1	0
74	i120-b2-f3-v2-a	59	12	100.00	100	*	132	33	441	0	2	0
75	i120-b2-f3-v3-m	65	12	115.00	115	*	127	13	214	0	3	0
76	i120-b2-f3-v3-a	55	10	96.00	96	*	25	2	29	0	4	0
77	i120-b2-f3-v4-m	56	10	113.00	113	*	42	6	72	0	4	0
78	i120-b2-f3-v4-a	64	11	135.00	135	*	74	3	40	0	1	0
79	i120-b2-f3-v5-m	61	11	120.00	120	*	57	5	86	0	2	0

Detailed results for using combinatorial cuts (continued)

id	name	$\|\mathcal{I}\|$	$\mathcal{K}^{\text{used}}$	LB	UB	gap	time	N^{iter}	$N^{\text{Cut}}_{\text{SR}}$	$N^{\text{Cut}}_{\text{LCC}}$	$N^{\text{Cut}}_{\text{combi}}$	N^{MIP}
80	i120-b2-f3-v5-a	59	—	109.08	111	1.73%	3600	313	4572	0	1	0
81	i120-b6-f1-v1-m	55	10	111.00	111	*	27	4	57	0	1	0
82	i120-b6-f1-v1-a	65	12	128.00	128	*	20	0	17	0	4	0
83	i120-b6-f1-v2-m	72	11	119.00	119	*	542	30	533	0	4	0
84	i120-b6-f1-v2-a	48	9	95.00	95	*	18	3	52	0	3	0
85	i120-b6-f1-v3-m	73	12	126.00	126	*	764	26	446	0	6	0
86	i120-b6-f1-v3-a	47	11	103.00	103	*	19	7	43	0	1	0
87	i120-b6-f1-v4-m	54	10	106.00	106	*	32	2	30	0	2	0
88	i120-b6-f1-v4-a	66	11	117.00	117	*	90	5	72	0	13	0
89	i120-b6-f1-v5-m	63	12	143.00	143	*	228	24	420	0	4	0
90	i120-b6-f1-v5-a	57	12	119.00	119	*	41	5	62	0	9	0

References

Artigues C, Michelon P, Reusser S (2003) Insertion techniques for static and dynamic resource-constrained project scheduling. European Journal of Operational Research 149(2):249–267, sequencing and Scheduling.

Barnhart C, Johnson E, Nemhauser G, Savelsbergh M, Vance P (1998) Branch–and–price: Column generation for solving huge integer programs. Operations Research 46(3):316–329.

Battarra M, Cordeau JF, Iori M (2014) Chapter 6: Pickup-and-delivery problems for goods transportation. Toth P, Vigo D, eds., Vehicle Routing, 161–191 (Philadelphia, PA: Society for Industrial and Applied Mathematics).

Beasley JE, Nascimento EM (1996) The vehicle routing-allocation problem: A unifying framework. TOP 4(1):65–86.

Beaudry A, Laporte G, Melo T, Nickel S (2010) Dynamic transportation of patients in hospitals. OR Spectrum 32(1):77–107.

Beck JC (2010) Checking-up on branch-and-check. Cohen D, ed., Principles and Practice of Constraint Programming – CP 2010, 84–98 (Berlin, Heidelberg: Springer Berlin Heidelberg), ISBN 978-3-642-15396-9.

Boland N, Dethridge J, Dumitrescu I (2006) Accelerated label setting algorithms for the elementary resource constrained shortest path problem. Operations Research Letters 34(1):58–68.

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112(3):3–41.

Bruglieri M, Mancini S, Pisacane O (2019) The green vehicle routing problem with capacitated alternative fuel stations. Computers & Operations Research 112:104759.

Cissé M, Yalçındağ S, Kergosien Y, Şahin E, Lenté C, Matta A (2017) OR problems related to home health care: A review of relevant routing and scheduling problems. Operations Research for Health Care 13-14:1–22.

Codato G, Fischetti M (2004) Combinatorial Benders' cuts. Bienstock D, Nemhauser G, eds., Integer Programming and Combinatorial Optimization, 178–195 (Berlin, Heidelberg: Springer Berlin Heidelberg), ISBN 978-3-540-25960-2.

Costa L, Contardo C, Desaulniers G (2019) Exact branch-price-and-cut algorithms for vehicle routing. Transportation Science 53(4):946–985.

De Reyck B, Demeulemeester E, Herroelen W (1999) Algorithms for scheduling projects with generalized precedence relations. Weglarz J, ed., Project Scheduling: Recent Models, Algorithms and Applications, 77–105 (Boston, MA: Springer US), ISBN 978-1-4615-5533-9.

Desaulniers G, Desrosiers J, Solomon M, eds. (2005) Column Generation (Springer), 1 edition.

Desaulniers G, Desrosiers J, Solomon MM (2002) Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. Ribeiro CC, Hansen P, eds., Essays and Surveys in Metaheuristics, 309–324 (Boston, MA: Springer US), ISBN 978-1-4615-1507-4.

Desaulniers G, Lessard F, Hadjar A (2008) Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. Transportation Science 42(3):387–404.

Desrochers M, Desrosiers J, Solomon M (1992) A new optimization algorithm for the vehicle routing problem with time windows. Operations Research 40(2):342–354.

Desrosiers J, Soumis F, Desrochers M (1984) Routing with time windows by column generation. Networks 14(4):545–565.

Desrosiers J, Soumis F, Desrochers M, Sauvé M (1986) Methods for routing with time windows. European Journal of Operational Research 23(2):236–245.

Doerner KF, Salazar-González JJ (2014) Chapter 7: Pickup-and-delivery problems for people transportation. Toth P, Vigo D, eds., Vehicle Routing, 193–212 (Philadelphia, PA: Society for Industrial and Applied Mathematics).

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

Ebben M, van der Heijden; MC, van Harten A (2005) Dynamic transport scheduling under multiple resource constraints. European Journal of Operational Research 167:320–335.

Feillet D (2010) A tutorial on column generation and branch-and-price for vehicle routing problems. 4OR 8(4):407–424.

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

Fikar C, Hirsch P (2017) Home health care routing and scheduling: A review. Computers & Operations Research 77:86–95.

Froger A, Mendoza JE, Jabali O, Laporte G (2019) The electric vehicle routing problem with capacitated charging stations, working paper or preprint.

Gartner D, Frey M, Kolisch R (2018) Hospital-wide therapist scheduling and routing: Exact and heuristic methods. IISE Transactions on Healthcare Systems Engineering 8(4):268–279.

Gélinas S, Desrochers M, Desrosiers J, Solomon MM (1995) A new branching strategy for time constrained routing problems with application to backhauling. Annals of Operations Research 61(1):91–109.

Gunpinar S, Centeno G (2016) An integer programming approach to the bloodmobile routing problem. Transportation Research Part E: Logistics and Transportation Review 86:94–115.

Hachemi NE, Gendreau M, Rousseau LM (2013) A heuristic to solve the synchronized log-truck scheduling problem. Computers & Operations Research 40(3):666–673.

Hanne T, Melo T, Nickel S (2009) Bringing robustness to patient flow management through optimized patient transports in hospitals. Interfaces 39(3):241–255.

Hans EW, van Houdenhoven M, Hulshof PJH (2012) A framework for healthcare planning and control. Hall R, ed., Handbook of Healthcare System Scheduling, 303–320 (Boston, MA: Springer US), ISBN 978-1-4614-1734-7.

Ho SC, Szeto W, Kuo YH, Leung JM, Petering M, Tou TW (2018) A survey of dial-a-ride problems: Literature review and recent developments. Transportation Research Part B: Methodological 111:395–421.

Hooker J, Ottosson G (2003) Logic-based Benders decomposition. Mathematical Programming 96:33–60.

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

Irnich S, Desaulniers G (2005) Shortest path problems with resource constraints. Desaulniers G, Desrosiers J, Solomon MM, eds., Column Generation, 33–65 (Boston, MA: Springer US), ISBN 978-0-387-25486-9.

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.

Jungwirth A (2020) Vehicle Routing with Time Windows and Flexible Delivery Locations. Ph.D. thesis, Technical University of Munich, Munich, Germany.

Jungwirth A, Frey M, Kolisch R (2020) The vehicle routing problem with time windows, flexible service locations and time-dependent location capacity. Technical report, Technical University of Munich, URL https://mediatum.ub.tum.de/doc/1427720/1427720.pdf.

Keskin M, Laporte G, Çatay B (2019) Electric vehicle routing problem with time-dependent waiting times at recharging stations. Computers & Operations Research 107:77–94.

Kontoravdis G, Bard J (1995) A GRASP for the vehicle routing problem with time windows. Journal on Computing 7(1):10–23.

Kramer R, Cordeau JF, Iori M (2019) Rich vehicle routing with auxiliary depots and anticipated deliveries: An application to pharmaceutical distribution. Transportation Research Part E: Logistics and Transportation Review 129:162–174.

Lam E, Hentenryck PV (2016) A branch-and-price-and-check model for the vehicle routing problem with location congestion. Constraints 21:394–412.

Lim A, Zhang Z, Qin H (2017) Pickup and delivery service with manpower planning in hong kong public hospitals. Transportation Science 51(2):688–705.

Nemati S, Shylo OV, Prokopyev OA, Schaefer AJ (2016) The surgical patient routing problem: A central planner approach. INFORMS Journal on Computing 28(4):657–673.

Ozbaygin G, Karasan OE, Savelsbergh M, Yaman H (2017) A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations. Transportation Research Part B: Methodological 100:115–137.

Pecin D, Pessoa A, Poggi M, Uchoa E (2017) Improved branch-cut-and-price for capacitated vehicle routing. Mathematical Programming Computation 9(1):61–100.

Reyes D, Savelsbergh M, Toriello A (2017) Vehicle routing with roaming delivery locations. Transportation Research Part C: Emerging Technologies 80:71–91.

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

Righini G, Salani M (2008) New dynamic programming algorithms for the resource constrained elementary shortest path problem. Networks 51(3):155–170.

Rix G, Rousseau LM, Pesant G (2015) A column generation algorithm for tactical timber transportation planning. Journal of the Operational Research Society 66:278–287.

Schmid V, Doerner KF (2014) Examination and operating room scheduling including optimization of intra-hospital routing. Transportation Science 48(1):59–77.

Toth P, Vigo D (2002) Chapter 1: An overview of vehicle routing problems. Toth P, Vigo D, eds., The vehicle routing problem, chapter An overview of vehicle routing problems, 1–26 (Philadelphia: Society for Industrial and Applied Mathematics).

Toth P, Vigo D, eds. (2014) Vehicle Routing (Philadelphia, PA: Society for Industrial and Applied Mathematics).

Vanderbeck F, Wolsey L (1996) An exact algorithm for IP column generation. Operations Research Letters 19(4):151–159.

Vidal T, Laporte G, Matl P (2020) A concise guide to existing and emerging vehicle routing problem variants. European Journal of Operational Research 286(2):401–416.

WHO (2016) Global strategy on human resources for health: Workforce 2030. URL https://apps.who.int/iris/bitstream/handle/10665/250368/9789241511131-eng.pdf.