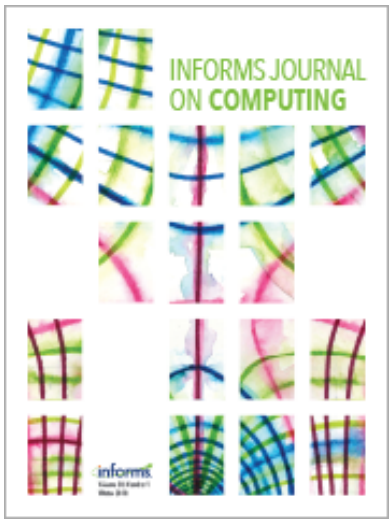

INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Constraint-Programming-Based Branch-and-Price-and-Cut Approach for Operating Room Planning and Scheduling

Seyed Hossein Hashemi Doulabi, Louis-Martin Rousseau, Gilles Pesant

To cite this article:
Seyed Hossein Hashemi Doulabi, Louis-Martin Rousseau, Gilles Pesant (2016) A Constraint-Programming-Based Branch-and-Price-and-Cut Approach for Operating Room Planning and Scheduling. INFORMS Journal on Computing 28(3):432-448. https://doi.org/10.1287/ijoc.2015.0686

INFORMS Journal on Computing
Vol. 28, No. 3, Summer 2016, pp. 432–448
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.2015.0686

A Constraint-Programming-Based Branch-and-Price-and-Cut Approach for Operating Room Planning and Scheduling

Seyed Hossein Hashemi Doulabi, Louis-Martin Rousseau
Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montreal, Canada H3C 3A7; and Interuniversity Research Center on Enterprise Networks, Logistics and Transportation (CIRRELT), Montreal, Canada H3C 3A7
{hashemi.doulabi@polymtl.ca, louis-martin.rousseau@polymtl.ca}

Gilles Pesant
Department of Computer and Software Engineering, Polytechnique Montréal, Montreal, Canada H3C 3A7; and Interuniversity Research Center on Enterprise Networks, Logistics and Transportation (CIRRELT), Montreal, Canada H3C 3A7,
gilles.pesant@polymtl.ca

This paper presents an efficient algorithm for an integrated operating room planning and scheduling problem. It combines the assignment of surgeries to operating rooms and scheduling over a short-term planning horizon. This integration results in more stable planning through consideration of the operational details at the scheduling level, and this increases the chance of successful implementation. We take into account the maximum daily working hours of surgeons, prevent the overlapping of surgeries performed by the same surgeon, allow time for the obligatory cleaning when switching from infectious to noninfectious cases, and respect the surgery deadlines. We formulate the problem using a mathematical programming model and develop a branch-and-price-and-cut algorithm based on a constraint programming model for the subproblem. We also develop dominance rules and a fast infeasibility-detection algorithm based on a multidimensional knapsack problem to improve the efficiency of the constraint programming model. The computational results show that our method has an average optimality gap of 2.81% and significantly outperforms a compact mathematical formulation in the literature.

Keywords: integrated operating room planning and scheduling; operations research in healthcare; branch-and-price-and-cut; constraint programming
History: Accepted by Michela Milano, Area Editor for Constraint Programming; received September 2014; revised June 2015; accepted November 2015. Published online May 11, 2016.

1. Introduction

Hospital managers must provide high-quality services by making efficient use of available medical resources. The operating theatre, which includes operating and recovery rooms, is particularly important: 60%–70% of patients are admitted for some forms of surgical intervention, accounting for more than 40% of the total cost (Guerriero and Guido 2011). Careful scheduling is necessary to decrease costs and improve resource utilization and patient flow.

Operating-theatre management has generally been divided into three stages (Guerriero and Guido 2011):

1. In the strategic stage, known as *case mix planning*, the operating-room availability is divided among the surgical departments or surgeons. The objective function may minimize the total deviation from a target allocation (Blake and Carter 2002) or maximize the weighted benefit of the scheduled surgeries (Baligh and Laughhunn 1969).

2. In the tactical stage, a *master surgery schedule* (a cyclic schedule over a medium-term planning horizon) is determined. It specifies the number of operating rooms available daily, the hours of availability, and the relative priorities of the surgeons or surgical departments.

3. In the operational stage, known as *elective case scheduling*, the daily scheduling is carried out. For each case we determine the surgeon, the starting time, the required equipment, etc. This stage is usually divided itself into the following two steps:

(a) *Operating-room planning*: Each surgery is assigned to a room and a day in the planning horizon (usually a week); this is also called *advance scheduling*.

(b) *Operating-room scheduling*: The surgeries are assigned to specific time intervals or the sequence of surgeries for each room is determined; this is also called *allocation scheduling*.

Two strategies can be used for the tactical and operational stages:

1. *Block-Scheduling Strategy*: A set of time blocks is assigned to surgeons or surgical departments. Each

time block is an interval on a specific day in a specific operating room.

2. *Open-Scheduling Strategy*: Surgeons from different departments can perform surgeries in the same time block. The surgeons must select the surgeries to perform the next day, and the operating theatre manager will schedule those surgeries in one or more rooms. This is a relaxation of the block-scheduling strategy, and it is expected to give a more efficient schedule. However, the management of such schedules is more difficult.

We focus on integrated operating room planning and scheduling (IORPS) at the operational level with an open-scheduling strategy. Many side constraints that are treated separately in the planning and scheduling phases must be considered simultaneously in the integrated problem. These constraints enforce the maximum daily hours of surgeons, prevent the overlapping of surgeries performed by the same surgeon, ensure sufficient cleaning time when switching from infectious to noninfectious cases, and enforce the surgery deadlines. The integrated approach leads to a more detailed planning problem, and this increases the chance of obtaining a stable schedule that can be successfully implemented. Our algorithm can also be applied in the context of the block-scheduling strategy when a block is shared among surgeons from the same department (Day et al. 2012).

IORPS has received little attention in the literature because of its intrinsic complexity; the planning and scheduling problems are usually solved sequentially that can lead to local optimal solutions. For example, Jebali et al. (2006) used a mixed integer programming (MIP) model to plan the surgeries for a given day and applied a second MIP model to assign surgeries to rooms. Guinet and Chaabane (2003) developed an assignment formulation for the planning problem and used a heuristic to assign surgeries to rooms. They then adjusted the daily assignments to satisfy other staff and material constraints. Fei et al. (2006) proposed a column generation (CG) model for a block-scheduling strategy assuming that just one surgeon works in a specific operating room on a specific day. This assumption makes the scheduling part of the studied problem less complex since surgeon overlapping does not happen. Fei et al. (2010) developed a two-stage approach for an open-scheduling strategy. The first stage maximizes the room utilization via CG, and the second stage uses a genetic algorithm to minimize the under- and overtime costs.

Only a limited numbers of papers in the literature have proposed unified approaches to do operating room planning and scheduling simultaneously. The exact approaches to IORPS use compact mathematical formulations based on four- or five-index binary variables. Roland et al. (2006, 2010) and Marques et al. (2012) defined binary variables x_{ikdt} that are 1 if surgery i starts at time t in room k on day d. To the best of our knowledge, Roland et al. (2006) were the first to investigate IORPS; their work was inspired by resource-constrained project scheduling. Roland et al. (2010) extended the model by considering various human resource constraints for the surgeons and nurses; they computationally showed that the proposed formulation is just capable of solving small-sized instances. Both studies applied genetic algorithms to solve large-sized instances. Marques et al. (2012) used the same type of four-index variable to formulate the integrated operating room planning and scheduling. Because of the large size of the MIP model, assignment and scheduling of emergency cases is first performed, then other patients are scheduled and finally a heuristic algorithm improves the solution. Vijayakumar et al. (2012) defined the binary variables x_{ikdts} that are 1 if surgery i is performed by surgeon s at time t in room k on day d. The model was able to solve small instances, and the authors also developed a heuristic approach.

In addition to the binary variables, the most recent model has continuous variables to model the start time of surgeries, and it includes the well-known scheduling constraints that require big-M values. Consequently, the linear programming relaxation of this model is expected to be weaker than the models previously discussed. Therefore, Roland et al. (2006, 2010) and Marques et al. (2012) can be considered the state-of-the-art exact algorithms for IORPS. Although these formulations address the IORPS problem as it is, they are inefficient in solving real-sized instances because they involve a huge number of binary variables. Our problem is not exactly the same as any of the problems investigated previously; the differences will be highlighted in Section 2 where assumptions are made.

We present a constraint-programming-based branch-and-price-and-cut algorithm for IORPS. A preliminary version, presented as a conference paper (Hashemi Doulabi et al. 2014) developed a column generation (CG) algorithm. CG is a well-known approach for linear programs that considers a large number of variables without including all of them explicitly in the model (Dantzig and Wolfe 1960, Gilmore and Gomory 1961, Desaulniers et al. 2005). The basic columns (variables) to define the optimal solution of such linear programs are generated iteratively by solving a subproblem. In the CG algorithm of Hashemi Doulabi et al. (2014) each subproblem generates schedules for a single room on a single day.

This paper extends the algorithm of Hashemi Doulabi et al. (2014) in two ways. First, we improve the efficiency of the constraint programming model in the subproblem. We add dominance rules to decrease the size of the subproblems, and we also present an infeasibility-detection algorithm based on a relaxation of the original subproblem that discovers infeasible

subproblems very quickly in many cases. Second, we extend the CG algorithm to a branch-and-price-and-cut approach by developing branching and cutting plane procedures. Our results show that the new approach significantly outperforms the original CG algorithm, decreasing the optimality gap from 9.90% to 2.81%. We also compare our approach with a state-of-the-art integer programming model for IORPS adapted from Marques et al. (2012) and also a pure constraint-programming model.

The remainder of this paper is organized as follows. Section 2 presents the problem definition and some notation, and Section 3 introduces the CG formulation. Section 4 discusses the dominance rules and a fast infeasibility-detection algorithm. Section 5 presents the general structure of the branch-and-price-and-cut algorithm and the branching and cutting-plane procedures. Section 6 presents extensive computational results, and Section 7 provides concluding remarks.

2. Problem Statement and Notation

In this section, we present the IORPS assumptions and some notation. An IORPS problem is defined over a planning horizon, usually a week. There are two types of surgeries: (1) mandatory surgeries whose deadlines fall within the current planning horizon and (2) optional surgeries that can be postponed to a later period. We consider an open scheduling environment, i.e., each room can be shared by different surgeons. The assumptions are as follows:

1. The durations of the surgeries are deterministic.
2. Each surgeon has determined his or her maximum surgery time for each day of the planning horizon.
3. Different types of infections can be present. An obligatory cleaning of the operating room is required when switching to a case with a different type of infection or to a noninfectious case. No cleaning is required between two cases with no infection or the same type of infection.
4. The surgeries have already been assigned to surgeons.
5. The recovery rooms are not a bottleneck: sufficient beds are available.
6. Eight hours are available in each operating room. No overtime is available.
7. The operating rooms are identical in terms of available time and equipment.

We maximize the total scheduled surgery time over the planning horizon subject to:

1. All mandatory surgeries are scheduled.
2. Surgeons do not exceed their maximum daily surgery times.
3. Each operating room and each surgeon can handle just one patient at a time.
4. Cleaning time is scheduled as previously explained.

One may criticize that Assumption 1 is not realistic as in the real world the durations of surgeries are stochastic. However it should be considered that the focus of this study is to have a planning problem with more details at the scheduling level to increase the chance of obtaining a stable schedule with more chance of successful implementation. Furthermore the problem is already very complicated so we leave the stochastic version to future research. Assumption 2 is realistic, but only Marques et al. (2012) have taken it into account. Assumption 3 is also very realistic and was first studied by Cardoen et al. (2009), but it has not been considered in the IORPS context. Assumption 4 is appropriate for hospitals where surgeons operate on the patients they have previously seen in their clinics. Vijayakumar et al. (2012) consider the assignment of surgeons to surgeries; this approach is suitable for teaching hospitals, where surgeries are assigned to assistant surgeons and residents as part of their training programs. Assumption 5 is common in the IORPS literature because the problem is already difficult without these additional resource constraints. Some researchers (Vijayakumar et al. 2012, Marques et al. 2012) apply Assumption 6 whereas others (Roland et al. 2006, 2010) allow overtime. Regarding Assumption 7, only Roland et al. (2010) consider rooms with different equipment. Our algorithm solves the problem without this assumption; see Appendix 1 (available as supplemental material at http://dx.doi.org/10.1287/ijoc.2015.0686) after reading Section 3. Our problem has features in common with the problems in the literature, but it is not exactly the same as any of them.

We now introduce the basic notation used throughout the paper. Some other notations may also be presented separately in each section.

Sets:

- D: Set of days in planning horizon.
- I: Set of surgeries; $I = I_1 \cup I_2$.
- I_1: Set of mandatory surgeries.
- I_2: Set of optional surgeries.
- I'_l: Set of surgeries for surgeon l.
- K_d: Set of operating rooms for day d.
- L: Set of surgeons.
- T_d: Set of time slots for day d; the available time is discretized into this set of time slots.

Parameters:

- t_i: Duration of surgery i ($i \in I$).
- s_i: Surgeon for surgery i.
- dd_i: Deadline of surgery i.
- A_{ld}: Maximum hours for surgeon l on day d.
- OCT: Duration of an obligatory cleaning that was explained in Assumption 3.
- $CL_{ii'}$: Equal to OCT if cleaning is required between surgeries i and i'; 0 otherwise.

Since the integer programming model developed in (Marques et al. 2012) is one of state-of-the-art exact algorithms for IORPS, an integer programming model adapted from (Marques et al. 2012) is presented in Appendix 2 of the online supplement. Moreover, a pure constraint-programming model formulating the previous problem is presented in Appendix 3 of the online supplement. We will compare our algorithm with these two approaches.

3. Formulation and Column Generation Algorithm

In this section we present an integer programming model and explain how we apply CG to this problem. The following two sections discuss the master problem and the subproblem.

3.1. Master Problem

For the master problem the sets, parameters, and variables are as follows.

Variable:

x_j: Equal to 1 if schedule j is accepted; 0 otherwise. Each schedule is a set of surgeries and cleaning activities with fixed start times for a single operating room on a given day; it must satisfy constraints 2–4 stated in Section 2.

Sets:

J: Set of all feasible schedules for all operating rooms during the planning period.
J_i: Set of feasible schedules that include surgery i.
J'_d: Set of feasible schedules for day d.

Parameters:

B_j: Total duration of all the surgeries in schedule j.
a_{ijt}: 1 if surgery i starts at time t in schedule j; 0 otherwise.
a'_{ij}: 1 if surgery i is included in schedule j; 0 otherwise.
b_{jd}: 1 if schedule j is scheduled on day d; 0 otherwise.

The master problem is as follows:

$$\max \sum_{j\in J} B_j x_j \tag{1}$$

$$\text{subject to: } \sum_{j\in J_i} x_j = 1, \quad \forall i \in I_1, \tag{2}$$

$$\sum_{j\in J_i} x_j \le 1, \quad \forall i \in I_2, \tag{3}$$

$$\sum_{j\in J'_d} x_j \le |K_d|, \quad \forall d \in D, \tag{4}$$

$$\sum_{j\in J'_d}\sum_{i\in I'_l} t_i a'_{ij} x_j \le A_{ld}, \quad \forall l \in L, \forall d \in D, \tag{5}$$

$$\sum_{j\in J'_d}\sum_{i\in I'_l}\sum_{t'\in T_d:\, t'\le t<t'+t_i} a_{ijt'} x_j \le 1, \quad \forall d \in D, \forall l \in L, \forall t \in T_d, \tag{6}$$

$$x_j \in \{0,1\}, \quad \forall j \in J. \tag{7}$$

Constraints (2)–(3) ensure that the mandatory surgeries are performed and allow the optional surgeries to be postponed. The deadlines of the mandatory surgeries will be respected in the subproblem. Constraint (4) ensures for each day that the number of schedules does not exceed the number of available operating rooms. Constraint (5) enforces the maximum working hours for each surgeon. Constraint (6) prevents overlapping surgeries for the same surgeon; in the literature it is referred to as the coloring constraint.

The model defined by (1)–(7) is inspired by the model presented by Fei et al. (2009). We have added the coloring constraint, and the objective function is also different. Moreover, our binary variables determine the daily schedule of an operating room, whereas the in Fei et al. (2009) the binary variables determine the assignments of surgeries to rooms but not the detailed schedules. The Fei model is intended just for planning, and it may assign more than one surgery to a surgeon at a given time. Therefore it is possible that the generated plans do not result in feasible schedules. The structure of our subproblem is totally different from that of the Fei subproblem.

Moreover, the other advantage of the proposed formulation to Fei model is that we have broken the symmetry of the identical operating rooms (see Section 3.2), and our model has only $|D|$ subproblems. In contrast, the Fei model must solve $\sum_{d\in D}|K_d|$ subproblems at each iteration of the CG, where $|D|$ is the number of days and $|K_d|$ is the number of operating rooms on day d. We could define other variables for the master problem; e.g., the variables could correspond to daily schedules for surgeons. The main advantage of the variables as defined is that the symmetry of the operating rooms can be broken in this case.

We embed model (1)–(7) in a branch-and-price algorithm. At each node k of the branch and bound tree, we generate a restricted number of columns ($J_k \subseteq J$); this problem is called the restricted master problem denoted RMP_k. Its linear programming relaxation, denoted $LR\text{-}RMP_k$, is solved by a CG algorithm, thereby generating more columns and obtaining a valid upper bound from the active nodes at the bottom of the tree. At the end of the branch-and-price algorithm we solve model (1)–(7) with all the generated columns to find a feasible solution and the corresponding lower bound.

3.2. Subproblem

After solving $LR\text{-}RMP_k$ we must solve a number of subproblems to generate new columns. These subproblems are defined using the dual values obtained by solving the relaxed master problem. If no column can be generated we terminate the CG process. We use a constraint programming model to formulate the subproblem. For each day $d \in D$ we solve a subproblem to generate a schedule for an operating room on that day.

Let $\pi_i^{(2)}$, $\pi_i^{(3)}$, $\pi_d^{(4)}$, $\pi_{ld}^{(5)}$, and $\pi_{ldt}^{(6)}$ be the dual variables corresponding to constraints (2)–(6) in the LR-RMP. We also define $\pi_i^{(2,3)}$ as

$$\pi_i^{(2,3)} = \begin{cases} \pi_i^{(2)}, & i \in I_1; \\ \pi_i^{(3)}, & i \in I_2. \end{cases}$$

The reduced cost of variable x_j is

$$\begin{aligned} \sigma_j = B_j &- \sum_{d\in D}\sum_{i\in I} b_{jd} a'_{ij} \pi_i^{(2,3)} - \sum_{d\in D} b_{jd}\pi_d^{(4)} \\ &- \sum_{d\in D}\sum_{l\in L}\sum_{i\in I'_l} t_i b_{jd} a'_{ij} \pi_{ld}^{(5)} \\ &- \sum_{d\in D}\sum_{l\in L}\sum_{i\in I'_l}\sum_{t\in T}\sum_{t'\in T_d:\, t'\le t<t'+t_i} a_{ijt'} b_{jd} \pi_{ldt}^{(6)}. \end{aligned}$$

In the subproblem we seek the column with the largest positive reduced cost. Because B_j is constant and all the other terms are summed over $d \in D$, decomposition of the subproblem over index $d \in D$ can be presented as follows:

$$\begin{aligned} \max\Bigg\{ B_j &- \sum_{i\in I} a'_{ij}\pi_i^{(2,3)} - \pi_d^{(4)} - \sum_{l\in L}\sum_{i\in I'_l} t_i a'_{ij}\pi_{ld}^{(5)} \\ &- \sum_{l\in L}\sum_{i\in I'_l}\sum_{t\in T}\sum_{t'\in T_d:\, t'\le t<t'+t_i} a_{ijt'}\pi_{ldt}^{(6)} \Bigg\}. \end{aligned}$$

Because only a small number of surgeries (n) can be performed in each operating room on a given day, we use a position-based model. The variables for the constraint programming model of the subproblem are as follows.

W_p: Index of surgery to be performed in position p on day d.

V_p: Start time of surgery assigned to position p on day d.

C_i: Number of W_p ($p \in \{1,\dots,n\}$) variables that take the value i ($i \in I \cup \{0\}$).

The subproblem constraints are:

$$W_p \in \{i \in I \mid dd_i \ge d\} \cup \{0\}, \quad \forall p \in \{1,\dots,n\}, \tag{8}$$

$$V_p \in \{0,1,\dots,|T_d|\}, \quad \forall p \in \{1,\dots,n\}, \tag{9}$$

$$\text{If } (W_p = 0) \text{ then } (W_{p+1} = 0), \quad \forall p \in \{2,\dots,n-1\}, \tag{10}$$

$$V_{p+1} \ge V_p + t_{[W_p]} + \mathrm{CL}_{[W_p][W_{p+1}]}, \quad \forall p \in \{1,\dots,n-1\}, \tag{11}$$

$$\sum_{p=1}^{n} (W_p == i) = C_i, \quad \forall i \in I, \tag{12}$$

$$0 \le C_i \le 1, \quad \forall i \in I, \tag{13}$$

$$\sum_{p=1}^{n} (s_{[W_p]} == l)\, t_{[W_p]} \le A_{ld}, \quad \forall l \in L, \tag{14}$$

$$\text{If } W_p = 0 \text{ then } V_p = V_{p-1} + t_{[W_{p-1}]}, \quad \forall p \in \{2,\dots,n\}. \tag{15}$$

Constraint (8) ensures that the positions are either assigned to a surgery in set I or left empty with value 0. The value n is an upper bound on the number of surgeries assigned to an operating room. Constraint (9) defines the domain of the start-time variables. Constraint (10) states that if a position is left empty then the next available position must also be empty, i.e., we do not allow empty positions between two surgeries. We observe that holes in the schedule are possible since the start times of the surgeries are determined by the V_p variables. The role of constraint (10) is to break the symmetry of the model by eliminating identical solutions with different assignments of the surgeries to the available positions. Constraint (11) ensures that the start time of a surgery is greater than the finish time of the previous surgery plus any cleaning time. In constraint (12), $(W_p == i)$ is a Boolean expression equal to 1 when W_p is equal to i and 0 otherwise. Constraint (12) together with constraint (13) ensures that each surgery appears at most once in a schedule. Constraint (14) enforces the maximum working hours of the surgeons. This restriction was present in the master problem (constraint (5)); we include it in the subproblem to prevent infeasible columns. Constraint (15) forces the start time of an empty position to be equal to the finish time of the previous position. This avoids similar columns with different start times for the empty positions.

We assume that $t_{[0]} = 0$ and $s_{[0]} = 0$. This ensures that when W_p is 0, $t_{[W_p]}$ and $s_{[W_p]}$ will be 0. This together with constraints (10) and (15) guarantees that when W_p is 0 the start time of this position and the next positions will be fixed to the start time of the last position with a surgery from I. We also assume that $\mathrm{CL}_{0i} = 0$, $\mathrm{CL}_{i0} = 0$, i.e., there is no cleaning before and after empty positions.

The objective function is

$$\begin{aligned} \max\Bigg\{ &\sum_{p=1}^{n} t_{[W_p]} - \sum_{p=1}^{n} \pi_{[W_p]}^{(2,3)} - \sum_{p=1}^{n} (t_{[W_p]}\pi_{d[s_{[W_p]}]}^{(5)}) \\ &- \sum_{p=1}^{n}\sum_{t\in T_d} ((t \ge V_p) \wedge (t < V_p + t_{[W_p]}))\pi_{d[s_{[W_p]}]t}^{(6)} - \pi_d^{(4)} \Bigg\}. \end{aligned} \tag{16}$$

Here, $\pi_{[W_p]}^{(2,3)}$, $\pi_{d[s_{[W_p]}]}^{(5)}$, and $\pi_{d[s_{[W_p]}]t}^{(6)}$ are element constraints that are 0 if W_p is 0. Otherwise, the corresponding dual values will be computed. The objective function can be simplified using precomputed matrices as follows:

$$\max\left\{ \sum_{p=1}^{n} G_{d[W_p]} - \sum_{p=1}^{n} \pi^*_{d[W_p][V_p]} - \pi_d^{(4)} \right\}, \tag{17}$$

where

$$G_{di} = \begin{cases} t_i - \pi_i^{(2,3)} - t_i \pi_{ds_i}^{(5)}, & i \ne 0; \\ 0, & i = 0; \end{cases} \quad \text{and}$$

$$\pi^*_{dit} = \begin{cases} \sum_{t' \in T_d : t \le t' < t + t_i} \pi^{(6)}_{ds_i t'}, & i \neq 0; \\ 0, & i = 0. \end{cases}$$

We observe that (17) is more efficient than (16) since $\sum_{p=1}^{n} \sum_{t \in T_d} ((t \ge V_p) \wedge (t < V_p + t_{[W_p]})) \pi^{(6)}_{d s_{[W_p]} t}$ has $n \times |T_d|$ element constraints whereas (17) has only n. As a result, it is much faster to create and delete (17).

In summary, the constraint programming model for the CG subproblem is

$\forall d \in D$:

$$\max \left\{ \sum_{p=1}^{n} G_{d[W_p]} - \sum_{p=1}^{n} \pi^*_{d[W_p][V_p]} - \pi^{(4)}_d \right\} \quad (18)$$

subject to: constraints (8)–(15)

$$\sum_{p=1}^{n} G_{d[W_p]} - \sum_{p=1}^{n} \pi^*_{d[W_p][V_p]} - \pi^{(4)}_d \ge 0. \quad (19)$$

Constraint (19) ensures that only columns with nonnegative reduced costs will be generated. During the constraint programming search, solutions with positive reduced costs will become new columns in the master problem. After the CG convergence, the objective value obtained by solving the linear programming relaxation of the master problem (LR-RMP) is an upper bound for the problem. We obtain a lower bound by solving an integer programming model for the restricted master problem.

4. Column Generation Enhancements

In this section we enhance the CG with four dominance rules and a fast infeasibility-detection algorithm. In constraint programming, different methods such as symmetry breaking and dominance rules were developed to reduce the search effort. Focacci and Milano (2001), and Fahle et al. (2001) independently developed symmetry breaking via dominance detection (SBDD). This algorithm breaks symmetry in the search tree by pruning a node if it is symmetrically equivalent to another node that is already explored. The idea of dominance rules is to specify some characteristics such that solutions with these characteristics dominate other solutions with respect to the objective function. Using dominance rules many low-quality solutions can be pruned off quickly resulting in significant speedups. Chu and Stuckey (2012) developed a generic method for generating dominance rules that are proven to be correct and compatible with each other. Many researchers have also applied problem-specific dominance rules in different combinatorial problems such as template design problem (Proll and Smith 1998), steel mill design problem (Prestwich and Beck 2004), rectangle packing problem (Korf 2004), open stacks problem (Chu and Stuckey 2009) and talent scheduling problem (De la Banda et al. 2011). Following in their footsteps we design problem-specific dominance rules applicable to the subproblem of our column generation algorithm that is a single machine scheduling problem with irregular starting time costs.

The first dominance rule characterizes the start time of the surgery in the first position. The second removes surgeries that will reduce the objective value of the subproblem if added to the schedule. For the third rule, we observe that if because of certain constraints a pair of surgeries cannot exist simultaneously in the same schedule, and one dominates the other in terms of the change in the objective value, then the latter does not appear in the optimal solution. For this dominance rule, if the pair of surgeries can appear simultaneously in the schedule then instead of removing the dominated surgery, we condition its existence in the solution on the scheduling of the dominant surgery. The fourth dominance rule states that if because of the constraints in the subproblem a set of surgeries that dominate another surgery cannot appear in an operating room schedule simultaneously, then the latter surgery does not appear in the optimal solution.

The fast infeasibility-detection algorithm solves a relaxation of the subproblem using a fast algorithm instead of the constraint programming solver. The latter can be time-consuming for subproblems that are infeasible with respect to constraint (19). If the objective value obtained by solving the relaxation of the subproblem is negative then the original subproblem is infeasible. Otherwise, the positive objective value found by the fast infeasibility-detection algorithm becomes an upper bound on the objective value of the constraint programming model.

4.1. Dominance Rules

The first dominance rule characterizes the start time of the surgery in the first position.

Dominance Rule 1. *The following constraint is valid for subproblem d*:

$$(V_1 == 0) \quad or \quad (V_1 \ge \mathrm{SD}_d),$$

where SD_d *is the shortest surgery duration for all the surgeries with deadlines on or after day d* (*i.e.*, $\mathrm{SD}_d = \min_{i \in I;\, dd_i \ge d}\{t_i\}$).

Proof. We prove that there always exists an optimal solution for the IORPS problem that satisfies the above condition in all the operating rooms. Assume that in an optimal solution, in one of the operating rooms on day d a surgery assigned to the first position starts at time V^* in the interval $[1, \mathrm{SD}_d - 1]$. The corresponding surgeon cannot have been in another operating room on this day in the interval $[0, \mathrm{SD}_d]$, because an overlap would then exist, which is a contradiction. Thus, it is possible to shift the start time of the surgery in the first position from V^* to 0. □

The other rules can be implemented dynamically. Note that for the variable selection we select the W_p variables in lexicographic order and then fix the V_p variables. The following dominance rules are applied when there is at least one W_p variable that is not yet fixed. For the remainder of this paper it is supposed that p^* the smallest index such that W_{p^*} is not yet fixed.

We define $\Delta_{id}^{\text{best}}$ to be the best improvement in the objective value of subproblem $d \in D$ when we add surgery i:

$$\Delta_{id}^{\text{best}} = t_i - \pi_i^{(2,3)} - t_i \pi_{[s_i]d}^{(5)} - \min_{t' \in \text{Dom}(V_p^*)} \left\{ \sum_{t=t'}^{t'+t_i-1} \pi_{[s_i]dt}^{(6)} \right\}, \quad (20)$$

where $\text{Dom}(V_{p^*})$ denotes the domain of variable V_{p^*}. Here t_i is the direct improvement from the increase in B_j when surgery i is added. The terms $-\pi_i^{(2,3)}$ and $-t_i \pi_{[s_i]d}^{(5)}$ are the changes in the objective function of the subproblem due to the dual values of constraints (2), (3), and (5). Because the dual values of constraint (6) are nonnegative, $-\min_{t' \in \text{Dom}(V_p^*)} \{\sum_{t=t'}^{t'+t_i-1} \pi_{[s_i]dt}^{(6)}\}$ is the best possible change in the objective function of the subproblem regarding the dual values of constraint (6). This term considers the best possible start time for surgery i. Note that the interval $[1, \text{SD}_d - 1]$ has been removed from $\text{Dom}(V_{p^*})$ by Dominance Rule 1. Equation (20) considers the possibility of performing surgery i in all available positions $p \geq p^*$ because in (20) we can replace $\text{Dom}(V_{p^*})$ with $\bigcup_{p \geq p^*} \text{Dom}(V_p)$ since $\text{Dom}(V_{p+1}) \subseteq \text{Dom}(V_p)$ is valid with respect to constraint (12).

In (16) and (18), $\pi_d^{(4)}$ is the nonnegative dual variable of constraint (4), which is fixed for each subproblem $d \in D$; adding surgery i does not change this part of the objective function. We similarly define $\Delta_{id}^{\text{worst}}$ to be the largest deterioration in the objective function of subproblem $d \in D$ when we add surgery i:

$$\Delta_{id}^{\text{worst}} = t_i - \pi_i^{(2,3)} - t_i \pi_{[s_i]d}^{(5)} - \max_{t' \in \text{Dom}(V_p^*)} \left\{ \sum_{t=t'}^{t'+t_i-1} \pi_{[s_i]dt}^{(6)} \right\}. \quad (21)$$

The next dominance rule eliminates surgeries that deteriorate the objective value of the subproblem.

Dominance Rule 2. *If $\Delta_{id}^{\text{best}} < 0$ then surgery i does not appear in the optimal solution of subproblem d.*

Proof. Assume that surgery i with $\Delta_{id}^{\text{best}} < 0$ appears in an optimal solution. When we remove surgery i, the objective function is improved by at least $|\Delta_{id}^{\text{best}}|$, which is a contradiction. □

Dominance Rules 3(a)–3(c) remove or condition a surgery by finding a dominant surgery. We introduce the following notation.

RAS_l: The maximum remaining time for surgeon l given the surgeries scheduled in positions 1 to $p^* - 1$. This is $A_{ld} - \sum_{p=1}^{p^*-1} (s_{[W_p]} == l) t_{[W_p]}$ for $p^* > 1$ and A_{ld} for $p^* = 1$.

RAO: The remaining time in the operating room given the surgeries scheduled in positions 1 to $p^* - 1$. This is $|T_d| - \min\{\text{Dom}(V_{p^*})\}$ where $\min\{\text{Dom}(V_{p^*})\}$ is the minimum value in the domain of V_{p^*}.

Dominance Rule 3(a). *In subproblem d, for a given surgery i with $dd_i \geq d$, if there is a surgery i' with $dd_{i'} \geq d$ and $s_i = s_{i'}$ such that*

(1) $\Delta_{i'd}^{\text{worst}} > \Delta_{id}^{\text{best}}$;
(2) $t_i \geq t_{i'} + \lambda$; and
(3) $t_i + t_{i'} > RAS_{s_i}$;

then surgery i is dominated by surgery i' and can be removed from the domain of W_p variables without losing the optimal solution. Here, λ is a constant upper bound on the additional pre-/post-cleaning time needed when surgery i is replaced by surgery i'.

Proof. The right-hand side of condition (3) is the maximum remaining time for surgeon $s_i = s_{i'}$. Given this condition, surgeries i and i' cannot both appear in a schedule. Thus, assume that surgery i is included in the schedule. Then, by condition (2), removing surgery i gives sufficient time to perform surgery i'. Condition (1) ensures that replacing surgery i by surgery i' improves the objective value. Therefore, surgery i does not appear in the optimal solution of subproblem d. □

Table 1 presents the values of λ for different combinations of infectious and noninfectious situations: f_i, $f_{i'}$, and f'_{p^*-1} denote the infection status of surgeries i, i', and the surgery in position $p^* - 1$. Appendix 4 of the online supplement derives the values of λ for the nine cases.

Dominance Rule 3(b). *If condition* (3) *of Dominance Rule* 3(a) *does not hold, then the following constraint is valid for the subproblem*:

$$C_i \leq C_{i'}. \quad (22)$$

This constraint states that if surgery i appears in the schedule then surgery i' must also appear; otherwise the objective value can be improved by replacing surgery i by surgery i'.

Dominance Rule 3(c). *In Dominance Rule* 3(a) *if $s_i \neq s_{i'}$ but conditions* (1) *and* (2) *hold and we have*

$$RAS_{s_{i'}} \geq RAO, \quad (23)$$

then constraint (22) *can be added to the subproblem without losing the optimal solution because the objective value can be improved by replacing surgery i by surgery i'. Condition* (23) *guarantees that the constraint on the maximum remaining time for surgeon $s_{i'}$ is not violated by the replacement.*

The next dominance rule considers a set of surgeries that conditions another surgery through constraint (22). If it is not possible to schedule all the entries in the set, then the latter surgery does not appear in the optimal solution.

Table 1 **Values of λ for Different Infectious and Noninfectious Combinations for Surgeries i, i', and the Surgery in Position p^*-1**

Case	Surgery i	Surgery i'	Surgery in position p^*-1	λ
1	Noninfectious	Noninfectious	No condition	0
2	Noninfectious	Infectious	Noninfectious or infectious with $f'_{p^*-1} \neq f_{i'}$	*OCT*
3	Noninfectious	Infectious	Infectious with $f'_{p^*-1} = f_{i'}$	0
4	Infectious	Noninfectious	Noninfectious or infectious with $f'_{p^*-1} = f_i$	0
5	Infectious	Noninfectious	Infectious with $f'_{p^*-1} \neq f_i$	*OCT*
6	Infectious	Infectious with $f_{i'} = f_i$	No condition	0
7	Infectious	Infectious with $f_{i'} \neq f_i$	Noninfectious or infectious with $f'_{p^*-1} \neq f_i$ and $f'_{p^*-1} \neq f_{i'}$	*OCT*
8	Infectious	Infectious with $f_{i'} \neq f_i$	Infectious with $f'_{p^*-1} = f_i$	2 *OCT*
9	Infectious	Infectious with $f_{i'} \neq f_i$	Infectious with $f'_{p^*-1} = f_{i'}$	0

Dominance Rule 4. *Let P_i be a set of surgeries that, with respect to constraint (22), must be scheduled if surgery i is scheduled. Then surgery i can be removed from the domain of W_p if $\sum_{r\in P_i\cup\{i\}} t_r > |T_d|$ or $\sum_{r\in(P_i\cap I'_{s_i})\cup\{i\}} t_r > A_{s_i d}$.*

Proof. If we include surgery i in the schedule, we must also include all the surgeries in P_i. However, $\sum_{r\in P_i\cup\{i\}} t_r > |T_d|$ indicates that the sum of the durations of surgery i and the preceding surgeries exceeds the available time in the operating room in subproblem d. Moreover, $\sum_{r\in(P_i\cap I'_{s_i})\cup\{i\}} t_r > A_{s_i d}$ indicates that the sum of the duration of surgery i and the durations of surgeon s_i's previous surgeries exceeds the available time for s_i on day d. Therefore, surgery i can be removed from the domain of W_{p^*}. □

4.2. Fast Infeasibility-Detection Algorithm

Sometimes the dual value of $\pi_d^{(4)}$ is so large that the best possible schedule in the subproblem has a negative objective value. No column with a positive reduced cost can be generated, and the subproblem is infeasible with respect to constraint (19). The rapid detection of such infeasibilities is very critical, especially in the last CG iterations. We use a knapsack relaxation to find an upper bound on the optimal value of subproblem d. We consider a multidimensional knapsack problem in which each surgery i from the subproblem corresponds to an item with the value $\Delta_{id}^{\text{best}}$. The integer programming formulation of this multidimensional knapsack problem reads as follows:

Sets:

I''_d: Set of surgeries with deadlines on or after day d that remain in the domain of W_p after that dominance rules have been applied.

Variables:

y_i: Equals to 1 if item i (corresponding to surgery i) is included in the knapsack.

$$\max \sum_{i\in I''_d} \Delta_{id}^{\text{best}} y_i \tag{24}$$

$$\text{subject to: } \sum_{i\in I''_d} t_i y_i \leq |T_d|, \tag{25}$$

$$\sum_{i\in I''_d\cap I'_l} t_i y_i \leq A_{ld}, \quad \forall\, l \in L, \tag{26}$$

$$y_i \in \{0,1\}, \quad \forall\, i \in I''_d. \tag{27}$$

Constraint (25) ensures that the sum of the durations of scheduled surgeries does not exceed the total available time in the operating room. Constraint (26) states that the total scheduled time for each surgeon does not exceed the maximum time for that surgeon. Obviously this multidimensional knapsack problem is a relaxation of the scheduling problem in the subproblem and an upper bound for this problem is an upper bound for the subproblem if we neglect the fixed value $\pi_d^{(4)}$. We use a well-known greedy algorithm with a value/weight selection criterion to find an upper bound for the problem defined by (24)–(27). As before, p^* is the smallest index such that W_{p^*} is not yet fixed. This fast infeasibility-detection algorithm is implemented dynamically, i.e., it called once at the beginning when no variable W_p is fixed ($p^*=1$) and also whenever a variable W_p is fixed ($p^*>1$). Before presenting the algorithm we define $\Delta_{idp}^{\text{best}}$ as follows:

$$\Delta_{idp}^{\text{best}} = t_i - \pi_i^{(2,3)} - t_i\pi_{[s_i]d}^{(5)} - \min_{t'\in\text{Dom}(V_p)} \left\{ \sum_{t=t'}^{t'+t_i-1} \pi_{[s_i]dt}^{(6)} \right\}. \tag{28}$$

This value is the best improvement in the objective value of subproblem d obtained by setting the start time of surgery i that is already scheduled at position p. This start time must take a value from $\text{Dom}(V_p)$. The difference between $\Delta_{idp}^{\text{best}}$ and $\Delta_{id}^{\text{best}}$ is that V_{p^*} is replaced by V_p.

The fast infeasibility-detection algorithm is as follows:

1. Set $UpperBound_d = \sum_{p=1}^{p^*-1} \Delta_{[W_p]dp}^{\text{best}}$. For $p^*=1$ this value is defined as zero.

2. Sort the unscheduled surgeries by $(\Delta_{id}^{\text{best}}/t_i)$ and set $i^* = \arg\max_{i \in I;\, dd_i \geq d}\{\Delta_{id}^{\text{best}}/t_i\}$.

3. Assign surgery i^* to the operating room if the remaining available time in the operating room is sufficient and the constraint on the maximum time of surgeon s_{i^*} is not violated. In this case, set $UpperBound_d := UpperBound_d + \Delta_{i^*d}^{\text{best}}$, $RAS_{s_{i^*}} := RAS_{s_{i^*}} - t_{i^*}$, and $RAO := RAO - t_{i^*}$ and return to step 1. Otherwise go to step 4 or 5.

4. If $t_{i^*} > RAS_{s_{i^*}}$ and $RAS_{s_{i^*}} < RAO$ then divide i^* into two surgeries i_1^* and i_2^* with durations $RAS_{s_{i^*}}$ and $t_{i^*} - RAS_{s_{i^*}}$ where s_{i^*} denotes the surgeon of surgery i^*. Assign surgery i_1^* to the schedule and disregard surgery i_2^*. Set $UpperBound_d := UpperBound_d + RAS_{s_{i^*}}(\Delta_{i^*d}^{\text{best}}/t_{i^*})$, $RAS_{s_{i^*}} := 0$, and $RAO := RAO - (t_{i^*} - RAS_{s_{i^*}})$ and return to step 2. Here $RAO > RAS_{s_{i^*}}$ guarantees that there is sufficient time in the operating room for the assignment of surgery i_1^*. Surgery i_2^* is eliminated for the rest of the algorithm as there is not any available time for surgeon s_{i^*} to perform it.

5. If $t_{i^*} > RAO$ and $RAO \leq RAS_{s_{i^*}}$ then divide surgery i^* into two surgeries i_1^* and i_2^* with durations RAO and $t_{i^*} - RAO$. Assign surgery i_1^* to the schedule and set $UpperBound_d := UpperBound_d + RAO(\Delta_{i^*d}^{\text{best}}/t_{i^*})$. Then stop because no operating-room time remains after the assignment of surgery i_1^*. Here condition $RAO \leq RAS_{s_{i^*}}$ ensures that surgeon s_{i^*} has sufficient time to perform surgery i_1^*.

This fast infeasibility-detection algorithm is evaluated dynamically as the W_p variables are fixed and p^* increases. When $p^* = 1$, if $UpperBound_d - \pi_d^{(4)} < 0$ at the end of the algorithm then the subproblem on day d is infeasible and can be skipped. Otherwise, the following constraint can be added to the subproblem:

$$\begin{aligned} &\textit{Objective Function of subproblem } d \\ &\quad \leq UpperBound_d - \pi_d^{(4)}. \end{aligned} \tag{29}$$

Constraint (29) ensures that the constraint programming solver stops when it finds a feasible solution with an objective value of $UpperBound_d - \pi_d^{(4)}$.

When $p^* > 1$, if $UpperBound_d - \pi_d^{(4)} < 0$, then the current partial solution will not lead to a complete solution with a positive objective value, and the constraint programming solver backtracks to another partial solution in the search tree. However, if $UpperBound_d - \pi_d^{(4)} \geq 0$, constraint (29) can be added locally to the search tree. In this case, constraint (29) reflects that the objective value of the current partial solution cannot be better than $UpperBound_d - \pi_d^{(4)}$ and the constraint programming solver then backtracks whenever it finds a local complete solution with an objective value of $UpperBound_d - \pi_d^{(4)}$.

It should be noted that the effect of cleaning times are not considered in the previous upper-bound calculation since this simplification results in a relaxed knapsack problem and is necessary to get a valid upper bound. Note that for the value selection of W_p variables the surgeries are sorted by descending order of $\Delta_{id}^{\text{best}}/t_i$ and for the V_p variables a lexicographic ordering is applied.

5. Branch-and-Price-and-Cut Algorithm

It is well-known that to improve the quality of solutions obtained from a CG algorithm, it can be extended to a branch-and-price algorithm where in each node of a branch-and-bound tree, variables with positive reduced cost (in a maximization problem) are generated by CG. We now describe the general structure of the branch-and-price-and-cut algorithm, and then we develop branching and cutting-plane procedures.

5.1. Structure of the Branch-and-Price-and-Cut Algorithm

Algorithms 1 and 2 describe the price-and-cut and branch-and-price-and-cut procedures. In Algorithm 1, two search phases are possible. If *CheckOptimalityPhase* is 0, it means that in solving the subproblems, the constraint programming solver will stop after *CPtimelimit* seconds. When *CheckOptimalityPhase* is set to 1, the constraint programming solver is allowed to solve the subproblems without any time limit. The algorithm switches from the first phase to the second only if no column is generated in an iteration (line 16). It is also possible that the algorithm switches to the first phase if some columns are generated in the second phase (line 21). If no column is generated when *CheckOptimalityPhase* is 1, *StoppingCriterion* will be set to 1 at line 18, terminating the while loop. In this case, the objective value of the relaxed master problem will be a valid upper bound. This careful use of a time limit is vital for the efficiency of the algorithm. The cuts added at line 6 will be explained in Section 5.3. At line 10 of Algorithm 1, when the constraint programming solver finds a solution with a positive objective value a new column is generated and the constraint programming algorithm continues its search to generate other columns with larger positive reduced costs. At line 12, we check the columns generated in this iteration to see if they also have positive reduced costs on other days. If they do, similar schedule patterns will be generated for the other days if they respect the surgery deadlines. Finally, we solve the master problem restricted to generated columns with binary variables to find a lower bound (line 24). The initial columns for the restricted master problem are generated by a constructive heuristic; see Section 6.1.

Algorithm 1 (Price-and-cut algorithm)

1: *StoppingCriterion* = 0; *CheckOptimalityPhase* = 0; *CPtimelimit* = 10 s
2: **while** (*StoppingCriterion* == 0) **do**
3: Solve the relaxed master problem.
4: Check for violated minimal cover cuts.
5: **while** (there are violated minimal cover cuts) **do**
6: Add a lifted minimal cover inequality to the master problem.
7: Solve the relaxed master problem.
8: **end while**
9: **for** (all subproblems $d \in D$) **do**
10: Solve the subproblem using constraint programming.
11: **end for**
12: Check if the generated columns can be copied to other days.
13: Add all columns generated in this iteration to the master problem.
14: **if** (no columns are generated) **then**
15: **if**(*CheckOptimalityPhase* == 0) **then**
16: *CheckOptimalityPhase* = 1; *CPtimelimit* = inf;
17: **Else**
18: *StoppingCriterion* = 1;
19: **end if**
20: **else if** (*CheckOptimalityPhase* == 1)
21: *CheckOptimalityPhase* = 0; *CPtimelimit* = 10 s;
22: **end if**
23: **end while**
24: Solve the restricted master problem with integer variables.

In the branch-and-price-and-cut algorithm presented by Algorithm 2, lines 3, 6, and 9 implement branches (1)–(3); see Section 5.2. The algorithm stops when a predetermined time limit is reached or the lower and upper bounds are the same, confirming the optimality of the current integer solution. The node selection (line 5) is based on a best-first strategy. If no branching is possible, the restricted master problem with integer variables is solved (line 11) to find a lower bound. If a predetermined time limit is reached (line 13), the algorithm computes a valid upper bound based on the upper bounds for those nodes for which the LR-RMP has been solved to optimality. The algorithm subsequently switches to fast branching, i.e., *CheckOptimalityPhase* is set to 1 in Algorithm 1. This allows the algorithm to rapidly generate many columns by visiting many nodes in the tree instead of spending time proving the optimality of the generated nodes. In fact, generating many columns using the prior strategy increases the chance of getting a good integer solution at line 16.

Algorithm 2 (Branch-and-price-and-cut algorithm)

1: UpperBound = $+\infty$; LowerBound = $-\infty$;
2: Apply *Price-and-Cut algorithm* at the root node.
3: Check and save possible branching at the root node;
4: **while** (stopping criteria are not activated) **do**
5: Select a node from the list of feasible nodes;
6: **if** (branching is possible) **then**
7: Branch and generate child nodes;
8: Apply *Price-and-Cut algorithm* to the generated nodes.
9: Check and save possible branching at generated nodes;
10: **else**
11: Solve the restricted master problem in the current node with integer variables.
12: **end if**
13: Check for switch to fast branching;
14: Check stopping criteria;
15: **end while**
16: Solve the restricted master problem with all generated columns and integrality constraints.

5.2. Branching Procedure

We use three types of branching:

(1) Branching on the number of total scheduled surgeries in the planning horizon.

(2) Branching on the number of scheduled surgeries for a particular surgeon in the planning horizon.

(3) Branching on whether two particular surgeries are scheduled in the same operating room (Ryan and Foster 1981).

Branches (1) and (2) require a new constraint in the master problem. For branch (3) some columns must be removed from the master problem, and some constraints must be added to the subproblem. Branches (1) and (2) are implemented as traditional dichotomic branches generating two new nodes. For branch (1), we add one of the following constraints to the master problem:

$$\sum_{j\in J}\sum_{i\in I} a'_{ij}x_j \geq \lceil m \rceil, \tag{30}$$

$$\sum_{j\in J}\sum_{i\in I} a'_{ij}x_j \leq \lfloor m \rfloor, \tag{31}$$

where m is the fractional number of scheduled surgeries in LR-RMP. For branch (2) for surgeon l, we add one of the following constraints to the master problem:

$$\sum_{j\in J}\sum_{i\in I'_l} a'_{ij}x_j \geq \lceil m'_l \rceil, \tag{32}$$

$$\sum_{j\in J}\sum_{i\in I'_l} a'_{ij}x_j \leq \lfloor m'_l, \rfloor, \tag{33}$$

where m'_l is the fractional number of scheduled surgeries for surgeon l in LR-RMP. In the computation of matrix G_{di} in (18), $\Delta_{id}^{\text{best}}$, $\Delta_{id}^{\text{worst}}$, and $\Delta_{idp}^{\text{best}}$ must be replaced by $g_i(G_{di})$, $g_i(\Delta_{id}^{\text{best}})$, $g_i(\Delta_{id}^{\text{worst}})$, and $g_i(\Delta_{idp}^{\text{best}})$, where

$$g_i(h) := \begin{cases} h - \pi^{(30)} - \pi^{(31)}, & i \neq 0, s_i \neq l, \\ h - \pi^{(30)} - \pi^{(31)} - \pi_{s_i}^{(32)} - \pi_{s_i}^{(33)}, & i \neq 0, s_i = l, \\ h, & i = 0, \end{cases}$$

and $\pi^{(30)}$, $\pi^{(31)}$, $\pi^{(32)}$, and $\pi^{(33)}$ are the dual variables of constraints (30)–(33).

Branch (3), introduced by Ryan and Foster (1981), gives better integer solutions than the other branches give. In one of the child nodes, two surgeries i and i' must be scheduled in the same operating room, and we remove from the master problem all the columns in which this is not the case. In the other child node, surgeries i and i' cannot be scheduled in the same operating room, and we remove some columns accordingly. We also add constraint (34) to the first child node and (35) to the second:

$$C_i = C_{i'}, \tag{34}$$

$$C_i + C_{i'} \leq 1. \tag{35}$$

As computationally shown in Appendix 5 of the online supplement, branches (1) and (2) usually improve the upper bound, whereas branch (3) is usually more effective in generating good columns, thus increasing the likelihood of finding good integer solutions. Therefore, to improve lower and upper bounds with the proposed branching rules, they are applied in the following order: We apply branch (1) first. For branch (2), we select the surgeon for which the fractional part of the number of scheduled surgeries is closest to 0.5. Finally, in branch (3), we select the pair of surgeries for which the fractional number of times that they appear together is closest to 0.5.

5.3. Cutting Planes

To improve the quality of the upper bound obtained by CG algorithm, a class of cutting planes is added to the master problem iteratively. We use the lifted minimal cover inequality (Gu et al. 1998, Atamtürk 2005) based on a reformulation of the knapsack capacity constraints (5), which are the constraints on the maximum daily hours of the surgeons. Barnhart et al. (2000) used the same idea to improve the quality of a branch-and-price algorithm for a multicommodity flow problem. We introduce an auxiliary binary variable z_{id} that is 1 if surgery i is scheduled on day d and 0 otherwise. Constraint (5) can now be reformulated as follows:

$$\sum_{i \in I'_l} t_i z_{id} \leq A_{ld}, \quad \forall d \in D, \forall l \in L. \tag{36}$$

The corresponding lifted minimal cover inequalities are

$$\sum_{i \in I'_l \setminus C} \alpha_i z_{id} + \sum_{i \in C} z_{id} \leq |C| - 1, \quad \forall d \in D, \forall l \in L, \forall C \in \Psi_{ld}, \tag{37}$$

where C is a minimal cover set for surgeon l on day d. This is the minimal subset of surgeries for surgeon l that cannot be scheduled together on day d because of the limited knapsack capacity (A_{ld}). The set Ψ_{ld} contains all minimal cover sets on day d for surgeon l. To impose these cutting planes we express z_{id} in terms of x_j via $z_{id} = \sum_{j \in J'_d \cap J_i} x_j$. The cuts to be added to the master problem are as follows:

$$\sum_{i \in I'_l \setminus C} \sum_{j \in J'_d \cap J_i} \alpha_i x_j + \sum_{i \in C} \sum_{j \in J'_d \cap J_i} x_j \leq |C| - 1, \quad \forall d \in D, \forall l \in L, \forall C \in \Psi_{ld}. \tag{38}$$

During the CG procedure, whenever we detect a violated cut, it is added to the master problem. To detect violated cuts, we use the default lifted cover inequality (LCI) algorithm of Gu et al. (1998). However, because of time efficiency instead of the exact lifting algorithm, a very fast lifting procedure from Atamtürk (2005) to generate LCI. This procedure is as follows:

$$\alpha_i = \begin{cases} r, & \theta_r \leq t_i \leq \theta_{r+1} - 1, \\ & \quad \forall r \in \{1, 2, \ldots, |C| - 1\}, \\ |C|, & t_i \geq \theta_{|C|}, \end{cases} \quad \forall i \in I'_l \setminus C, \tag{39}$$

where t_i is the duration of surgery i. Here $\theta_r = \sum_{k=1}^{r} t_{c_k}$, $\forall r \in \{1, 2, \ldots, |C|\}$, and c_k is the kth surgery in the minimal cover C when the surgeries in C are sorted in nondecreasing order of durations.

The main concern when adding cuts to a CG algorithm is how to consider the effect of the dual values of the added cuts in the objective function of the subproblems. To address this, we define Γ_{di} to be the set of all added cuts that include surgery i on day d with a nonzero coefficient for z_{id}. In each subproblem $d \in D$, whenever a new column that includes surgery i is examined, the effect of the dual values of all the cuts in Γ_{di} must be taken into account in the reduced-cost computation. We define λ_{di} to be the contribution of the dual values of the added cuts in the objective function of subproblem d due to the inclusion of surgery i on day d:

$$\lambda_{di} = \sum_{v \in \Gamma_{di}} \pi_v \alpha'_{iv}, \quad \forall i \in I, \forall d \in D, \tag{40}$$

where π_v is the dual value of cut $v \in \Gamma_{di}$ and α'_{iv} is the coefficient of z_{id} in this cut. After we solve the master problem, we compute the matrix $\lambda = [\lambda_{di}]$ and replace (18) by the following objective function:

$$\max \left\{ \sum_{p=1}^{n} G_{d[W_p]} - \sum_{p=1}^{n} \pi^*_{d[W_p][V_p]} - \sum_{p=1}^{n} \lambda_{d[W_p]} - \pi_d^{(4)} \right\}.$$

This is equivalent to the aggregation of matrix λ in matrix A in (18) as follows: $G_{di} = t_i - \pi_i^{(2,3)} - t_i \pi_{ds_i}^{(5)} - \lambda_{di}$ (for $i \neq 0$). Further, $\Delta_{id}^{\text{best}}$, $\Delta_{id}^{\text{worst}}$, and $\Delta_{idp}^{\text{best}}$ must be updated via $\Delta_{id}^{\text{best}} := \Delta_{id}^{\text{best}} - \lambda_{di}$, $\Delta_{id}^{\text{worst}} := \Delta_{id}^{\text{worst}} - \lambda_{di}$, and $\Delta_{idp}^{\text{best}} := \Delta_{idp}^{\text{best}} - \lambda_{di}$.

We define the lifted minimal cover inequalities on constraint (36) and not constraint (5) because in the former the effect of the dual values of cuts can be considered in the objective function of the subproblems, whereas in the latter there is no effective way to do this. We also tested clique cuts and odd-cycle inequalities, but because of the difficulty in efficiently considering the dual values of the variables in these cuts, they were not very effective and we do not discuss them further.

6. Computational Experiments

We implemented the algorithm in IBM ILOG CPLEX Optimization Studio V12.4, which includes CPLEX and CP Optimizer to solve linear, integer, and constraint programming models. Experiments were run on a computer with two Intel Xeon X5675 processors, 3.07 GHz, and a total of 12 cores. We ran different instances using different cores.

6.1. Instances

We generated three sets of instances with different specifications. In all the instances, the planning horizon is five days of a week and the maximum hours for the surgeons (A_{ld}) are taken from Fei et al. (2009, Table 1) with some modifications. Fei et al. (2009) allowed some surgeons to work overtime beyond eight hours but we do not, and the corresponding entries are decreased to eight hours. In our instances, the average maximum time for surgeons is about 320 minutes with a variance of roughly 20 minutes (i.e., $\text{Var}[(\sum_{d\in D} A_{ld}/|D|)_{l\in L}] =$ 20 min). The surgery deadlines are uniformly generated in the interval [1, 14] as suggested by Fei et al. (2009). Surgeries with deadlines after the fifth day are optional. The surgeries are randomly assigned to eight surgeons, as suggested by Fei et al. (2009). We randomly assigned half of the surgeries in each instance to the infectious category, and we set the cleaning time (OCT) to 30 minutes.

In the first set of instances, denoted A, the surgery durations are uniformly generated in the interval [2 hours, 4 hours] and time is discretized into five-minute units. Six operating rooms are available for eight hours each day. Usually the number of rooms is large enough that this resource does not happen to be the bottleneck of the system and the availability of the surgeons is more restrictive.

Instance set B is the same as A except that the number of operating rooms is set using a heuristic (see Appendix 6 in the online supplement). The heuristic computes an approximate average of the daily working hours of each surgeon, assuming that the surgeon is going to perform all the surgeries and the workload is distributed evenly over the planning horizon. It then uses these values to compute the operating-room hours required daily. From this we derive the number of operating rooms required. The main characteristic of instance set B is that the number of operating rooms can be restrictive.

In instance set C, the surgery durations are generated according to the Pearson III distribution, as explained by Fei et al. (2009). The daily numbers of operating rooms are again set using the heuristic. The average durations of the surgeries in this set is considerably lower than those of the two previous sets.

The three instance sets cover a wide range of operating room planning and scheduling problems in different surgical departments. Sets A and B are good representatives of cases that surgeries take more than two hours, such as cardiac surgery, bile duct and liver surgery, and various types of vascular surgery (Leong et al. 2006). Set C is representative of cases with shorter surgery durations such as hemiarthroplasty, limb amputation, and abdominal hysterectomy which commonly take less than two hours (Leong et al. 2006). We set the number of surgeries in A and B to $\{40, 60, 80, 100, 120\}$ and the number in set C to $\{60, 80, 100\}$. We generate five instances for each problem setting for a total of 65 instances. We do not consider 40 or 120 surgeries in set C because in the former case, due to short surgery durations, only one operating room is required based on the heuristic presented in Appendix 6 of the online supplement and in the latter neither the proposed algorithm nor the integer programming and constraint programming models presented in Appendices 1 and 2 are very efficient.

To ensure feasibility of generated instances considering surgery deadlines, we apply a constructive heuristic to generate an initial solution for each instance. In this heuristic whenever a surgery deadline cannot be respected its deadline is postponed to the next day. The heuristic fills the operating rooms consecutively starting from the first operating room on the first day to the last one on the last day. At each step it assigns a single surgery to an operating room. For scheduling surgeries on each day the surgeons are considered in a random order, and the surgeries are sorted by deadline. Two criteria break ties: (1) Surgeries performed by more important surgeons according to a random importance assignment are considered before other surgeries for the assignment. (2) If ties remain, the tied cases are sorted by decreasing order of processing time. In each operating room earlier time slots are favored as the start time of surgeries. The selected start time must lead to a feasible solution for the other surgeries of the same surgeon, i.e., all coloring constraints and

maximum working hours must be respected. The resulting schedules define the initial columns of the master problem.

6.2. Parameters

The computational results are presented in Tables 2–8. We set the CG time limit to two hours, and we set *CPtimelimit*, one of the parameters of the Algorithm 1 explained in Section 5.1, to 10 seconds. If the CG does not converge within the time limit we report a trivial upper bound, $\min(\sum_{d\in D}|K_d||T_d|, \sum_{d\in D}\sum_{l\in L}A_{ld}, \sum_{i\in I}t_i)$. We then solve the integer programming model of the restricted master problem with a time limit of one hour to find a lower bound (line 24 of Algorithm 1). In the case of a branch-and-price or a branch-and-price-and-cut, we allow an additional two hours for the branching procedure. In the first 15 minutes, we try to prove the optimality of the nodes. We then compute an upper bound of the tree and the optimality of generated nodes will not be proved anymore and child nodes of a given node are generated as the CG algorithm switches to $CheckOptimalityPhase = 1$. After the branching procedure, we apply the integer programming model solver for one hour with all the generated columns (line 16 of Algorithm 2).

6.3. Results

In Tables 2–8, we give the computational times in seconds; each row presents the average over five instances. In Table 2, we evaluate the dominance rules and the fast infeasibility-detection algorithm. Under columns "*Original CG*" and "*Enhanced CG*" the computational results of the CG algorithm without applying any of enhancements and with all of them are presented respectively. In the enhanced CG, dominance rules and the fast infeasibility-detection algorithm are implemented dynamically and are called whenever a W_p variable is fixed. We give the number of CG iterations (*No. of iterations*) and the number of instances for which the CG has not converged within the time limit (*No. of unconverged*). The results in columns 5–12 are obtained by averaging over all the subproblems and all the CG iterations at the root node of the constraint programming search tree, (i.e., when no W_p variable is fixed yet and $p^* = 1$) although the dominance rules and the fast infeasibility-detection algorithm are implemented dynamically. This provides a good sense of the effectiveness of these features.

"*DR* 1" gives the percentage of values in the domains of V_1 removed by Dominance rule 1. "*DR* 2," "*DR* 3," and "*DR* 4" give the percentages of surgeries in the subproblems removed by dominance rules 2–4. When a rule removes a surgery from the domain of W_{p^*}, that surgery is not considered by the subsequent rules. The dominance rules are applied sequentially as previously mentioned. This is reasonable because Dominance Rule 2 is the least time-consuming: it is executed in $O(|I|)$. Dominance Rule 2 removes some of the surgeries from the domain of W_{p^*} before Dominance Rule 3 (which runs in $O(|I|^2)$) examines all pairs of available surgeries. Dominance Rules 3(b) and 3(c) add some constraints in the form of constraint (22) that increases the number of surgeries in the set P_i, thus making Dominance Rule 4 more effective. "*Conditioned values*" gives the percentage of surgeries in the subproblems that are conditioned by constraint (22) but not removed by any of the dominance rules. "*FID* 1" gives the percentage of infeasible subproblems detected by the fast infeasibility-detection algorithm. "*FID* 2" gives the percentage of infeasible subproblems over all the CG

Table 2 Evaluation of Dominance Rules and the Fast Infeasibility-Detection Algorithm

		Original CG		Enhanced CG									
Instance set	No. of *surgeries*	No. of iterations	No. of unconverged	DR 1 (%)	DR 2 (%)	DR 3 (%)	DR 4 (%)	Conditioned values (%)	FID 1 (%)	FID 2 (%)	Stopped by knapsack bound (%)	No. of iterations	No. of unconverged
A	40	13	0	25.21	67.41	3.95	0.88	3.44	9.61	28.32	11.72	16	0
	60	26	0	25.00	61.64	4.25	0.84	3.77	29.96	13.61	7.83	30	0
	80	57	0	25.00	43.14	5.18	1.16	4.77	20.96	3.42	6.77	41	0
	100	122	5	25.21	38.86	5.15	0.82	4.64	27.72	0.15	3.58	76	0
	120	125	5	25.00	34.97	6.19	1.45	6.56	24.28	0.00	4.65	62	0
B	40	13	0	25.17	4.96	5.88	1.99	12.11	0.00	0.00	11.08	17	0
	60	22	0	25.21	3.22	8.61	2.79	10.37	0.00	0.00	8.15	25	0
	80	57	1	25.00	2.72	7.49	1.84	9.05	0.00	0.00	5.55	39	0
	100	97	5	25.00	4.59	8.39	1.13	6.20	7.79	0.00	2.70	68	0
	120	95	5	25.00	3.65	9.45	2.12	6.45	3.74	0.00	2.55	89	3
C	60	61	5	10.50	4.65	5.50	7.71	48.93	2.27	0.00	23.80	25	1
	80	43	5	9.44	4.18	6.76	9.02	47.10	7.93	0.00	12.50	55	1
	100	69	5	9.17	4.08	6.00	11.99	47.51	4.35	0.00	10.65	50	2
Average:	—	61	—	21.53	21.39	6.37	3.36	16.22	10.66	3.50	8.58	46	—
Total:	—	—	36	—	—	—	—	—	—	—	—	—	7

iterations that are detected because all the surgeries are removed from the domains of W_p by the application of Dominance Rules 2–4 at the root node of the constraint programming search tree (i.e., when no W_p variable is fixed yet and $p^* = 1$). Column "*Stopped by knapsack bound*" gives the percentage of subproblems for which the constraint programming solver stopped because it found a solution with an objective value equal to the upper bound imposed by constraint (29) for the case where $p^* = 1$. Table 2 shows that

(1) 31% of the surgeries are removed by the dominance rules;

(2) 16.22% of the surgeries are conditioned by constraint (22);

(3) about 14% of subproblems are detected to be infeasible before letting the constraint programming solver do any propagation;

(4) 8.58% of the subproblems also stopped as they reach the knapsack bound of constraint (29); and

(5) the dominance rules and fast infeasibility-detection algorithm reduce the number of unconverged instances from 36 to 7.

We observe that the effects of the dominance rules and fast infeasibility-detection algorithm vary; this is especially noticeable for "*FID* 2."

Tables 3–6 present results for CG algorithms, price-and-cut, branch-and-price, and branch-and-price-and-cut. We apply the dominance rules and ID algorithm unless otherwise stated. The columns are: (1) "*No. of columns*": the total number of columns generated. (2) "*Alg. time*": the computational time, excluding the time to find a lower bound at the end of the CG and the end of the branching. (3) "*LB time*": the time to solve the restricted master problem to obtain a lower bound. (4) "*Total time*": the sum of "*Alg. Time*" and "*LB time*", (5) "*UB*": the upper bound. (6) "*LB*": the lower bound. (7) "*Gap*": the optimality gap, i.e., $100(UB - LB)/UB$. (8) "*No. of cuts*": the number of cuts

Table 3 Evaluation of the CG Algorithms

		Original CG							Enhanced CG						
Instance set	No. of surgeries	No. of columns	Alg. time	LB time	Total time	UB	LB	Gap (%)	No. of columns	Alg. time	LB time	Total time	UB	LB	Gap (%)
A	40	627	48	0	48	1,431	1,395	2.44	357	25	0	25	1,431	1,384	3.20
	60	1,396	532	46	578	2,020	1,941	3.92	877	108	2	110	2,020	1,865	7.72
	80	2,204	2,966	11	2,977	2,307	2,123	7.94	1,386	407	7	414	2,307	2,077	9.98
	100	3,004	7,200	12	7,212	2,880	2,195	23.79	1,619	1,960	14	1,974	2,401	2,159	10.04
	120	2,606	7,200	5	7,205	2,880	2,183	24.19	1,532	1,337	8	1,345	2,423	2,183	9.91
B	40	692	79	1	80	1,371	1,352	1.35	448	35	1	36	1,371	1,355	1.12
	60	1,243	605	9	614	1,866	1,807	3.17	812	137	5	142	1,866	1,799	3.59
	80	2,041	3,569	176	3,745	2,207	2,083	5.58	1,220	736	23	759	2,205	2,067	6.22
	100	2,894	7,200	523	7,723	2,438	2,276	6.67	1,921	2,349	373	2,722	2,416	2,271	6.01
	120	2,861	7,200	54	7,254	2,496	2,206	11.61	1,878	4,807	155	4,962	2,488	2,221	10.71
C	60	571	7,200	0	7,200	576	507	11.98	741	1,964	4	1,968	576	532	7.71
	80	412	7,200	0	7,200	792	688	13.11	1,307	3,508	4	3,512	792	692	12.58
	100	480	7,200	0	7,200	979	853	12.91	1,878	4,916	7	4,923	979	853	12.89
Average			4,477					9.90		1,715					7.82

Table 4 Evaluation of the Price-and-Cut Algorithm

		Price-and-cut							
Instance set	No. of surgeries	No. of columns	No. of cuts	Alg. time	LB time	Total time	UB	LB	Gap (%)
A	40	474	31	35	0	35	1411	1402	0.61
	60	1,093	72	129	10	139	1,975	1,897	3.90
	80	1,833	118	424	10	434	2,205	2,141	2.88
	100	2,498	134	2,158	47	2,205	2,297	2,218	3.44
	120	2,591	199	1,055	10	1,065	2,301	2,231	3.01
B	40	546	29	42	1	43	1,367	1,355	0.88
	60	1,117	73	187	18	205	1,860	1,819	2.13
	80	2,206	105	1,662	97	1,759	2,173	2,111	2.85
	100	3,036	144	4,633	1,139	5,772	2,402	2,320	3.38
	120	3,173	178	5,002	315	5,317	2,457	2,306	6.14
C	60	754	0	2,329	9	2,338	576	532	7.57
	80	1,230	0	6,034	3	6,037	792	693	12.53
	100	1,787	0	4,729	11	4,740	978	855	12.56
Average									4.76

Table 5 Evaluation of the Branch-and-Price Algorithm

		Branch-and-price-and-cut								
Instance set	No. of surgeries	No. of columns	No. of nodes (optimality proved)	No. of nodes (all nodes)	Alg. time	LB time	Total time	UB	LB	Gap (%)
A	40	631	397	821	182	1	183	1,421	1,400	1.49
	60	6,399	20	507	7,323	1,441	8,764	2,017	1,957	2.99
	80	4,660	21	966	6,590	288	6,878	2,308	2,151	6.73
	100	4,211	8	135	9,170	79	9,249	2,401	2,217	7.64
	120	3,557	12	152	8,552	30	8,582	2,471	2,234	9.58
B	40	2,776	64	781	7,238	6	7,244	1,366	1,361	0.39
	60	3,036	32	388	7,344	323	7,667	1,862	1,841	1.17
	80	2,523	13	416	7,945	88	8,033	2,202	2,114	3.97
	100	2,778	4	455	9,532	1,068	10,600	2,416	2,303	4.68
	120	2,853	1	65	12,033	306	12,339	2,488	2,296	7.72
C	60	3,964	10	356	9,175	1,101	10,276	576	570	1.08
	80	4,030	5	102	10,739	2,568	13,307	792	712	10.08
	100	5,348	2	84	12,141	2,957	15,098	977	858	12.20
Average										5.36

Table 6 Evaluation of the Branch-and-Price-and-Cut Algorithm

			Branch-and-price-and-cut									
Instance set	No. of surgeries	Initial objective value (heuristic)	No. of columns	No. of cuts	No. of nodes (optimality proved)	No. of nodes (all nodes)	Alg. time	LB time	Total time	UB	LB	Gap (%)
A	40	1,321	785	34	3	39	83	1	84	1,403	1,403	0.00
	60	1,814	9,379	132	19	300	7,353	1,306	8,659	1,971	1,960	0.55
	80	1,894	7,015	152	15	137	7,639	864	8,503	2,203	2,174	1.30
	100	1,968	5,413	178	6	80	9,381	222	9,603	2,297	2,255	1.81
	120	2,014	5,487	230	6	66	8,273	22	8,295	2,298	2,266	1.40
B	40	1,193	1,781	63	41	225	2,262	207	2,469	1,361	1,361	0.00
	60	1,601	3,475	131	21	191	7,398	779	8,177	1,853	1,843	0.55
	80	1,824	4,796	141	11	89	8,878	906	9,784	2,161	2,132	1.32
	100	1,993	4,819	168	2	46	11,862	2,814	14,676	2,401	2,337	2.65
	120	2,004	5,120	215	2	43	12,233	1,157	13,390	2,457	2,355	4.13
C	60	507	4,216	0	5	141	9,547	1,786	11,333	576	571	0.94
	80	688	3,745	2	2	85	13,261	2,319	15,580	792	713	9.92
	100	853	4,929	0	2	105	11,956	3,589	15,545	978	860	12.01
Average												2.81

added. (9) "*No. of nodes* (*optimality proved*)": the number of nodes for which optimality is proved. (10) "*No. of nodes* (*all nodes*)": the total number of nodes generated.

Table 3 shows that the dominance rules and the fast infeasibility-detection algorithm decrease the computational time from 4,477 seconds to 1,715 seconds, but for most instances in sets A and B the quality of integer solutions obtained from the original CG is better than that of integer solutions resulted from the enhanced CG. This is because of the larger number of columns generated in the earlier algorithm which let the integer programming model called at the end of the algorithm have more choices to get good integer solutions, whereas the second algorithm converges faster with fewer number of columns as a result of applying dominance rules. However, in 21 instances of sets A and B the original CG has not converged and trivial upper bounds are reported. For set C, the original CG struggles to solve the subproblems; the enhanced CG generates more columns and finds a better lower bound.

As can be seen in Tables 3–6, the average optimality gaps for original CG, enhanced CG, price-and-cut, branch-and-price, and branch-and-price-and-cut are 9.90%, 7.82%, 4.76%, 5.36%, and 2.81% respectively. In Table 6, the "*initial objective value*" is obtained from the constructive heuristic previously explained in this section; our algorithm significantly improves these values. Depending on the time limits in practical cases, one may decide to use the branch-and-price-and-cut or the price-and-cut algorithms that require two to four hours for large instances. If less time is available, the practitioner could develop a heuristic and validate it by comparing it with our algorithms.

Table 7 Evaluation of the Integer Programming Model, Constraint Programming Model, and Mixed Algorithm

		IP				CP						CP-IP			
Instance set	No. of surgeries	Time	UB	LB	Gap (%)	No. of branches (millions)	No. of fails (millions)	Time	UB	LB	Gap (%)	Time	UB	LB	Gap (%)
A	40	300	1,415	1,112	20.97	2.71	1.12	300	1,446	1,366	5.51	257	1,411	1,399	0.86
	60	11,000	1,966	1,941	1.20	70.29	28.24	11,000	2,150	1,693	21.23	10,401	1,974	1,879	4.64
	80	11,100	2,220	1,670	24.76	58.74	23.25	11,100	2,436	1,947	20.07	11,100	2,225	1,995	10.31
	100	11,600	2,302	851	63.15	44.74	16.79	11,600	2,436	2,058	15.50	11,600	2,316	2,101	9.20
	120	8,800	2,338	421	81.10	29.13	10.58	8,800	2,436	2,041	16.22	8,800	2,351	2,056	12.52
B	40	4,400	1,361	1,304	4.18	46.17	20.56	4,400	1,402	1,285	8.27	4,117	1,365	1,323	3.09
	60	11,000	1,851	1,700	8.11	92.89	40.00	11,000	1,920	1,673	12.85	11,000	1,856	1,739	6.21
	80	12,800	2,180	1,439	33.90	70.61	28.58	12,800	2,224	1,841	17.12	12,800	2,189	1,898	13.26
	100	18,200	2,400	0	100.00	77.18	28.95	18,200	2,433	2,007	17.50	18,200	2,408	2,035	15.51
	120	17,800	2,430	0	100.00	67.26	23.87	17,800	2,484	2,140	13.86	17,800	2,434	2,133	12.39
C	60	14,800	576	538	6.63	125.40	54.27	14,800	576	534	7.22	14,800	576	547	4.97
	80	18,200	792	267	66.24	161.85	68.96	18,200	792	611	22.80	18,200	792	718	9.29
	100	18,200	978	0	100.00	112.69	60.74	18,200	979	487	51.65	18,200	978	487	51.62
Average					46.94						17.68				11.84

Table 8 Improvement in Optimality Gap from Cuts and Branching

Instance set	No. of surgeries	Improvement from cuts (%)	Improvement from branching (%)	Improvement from branching and cuts (%)
A	40	2.59	1.71	3.20
	60	3.82	4.74	7.18
	80	7.10	3.24	8.67
	100	6.61	2.41	8.23
	120	6.90	0.33	8.51
B	40	0.24	0.73	1.12
	60	1.45	2.42	3.03
	80	3.37	2.24	4.89
	100	2.63	1.33	3.36
	120	4.57	2.98	6.58
C	60	0.14	6.63	6.77
	80	0.05	2.50	2.65
	100	0.33	0.69	0.88
Average		3.06	2.46	5.01

Table 7 gives the results for the integer programming model (*IP*) and the constraint programming model (*CP*) developed in Appendices 1 and 2. For each row we set the time limit to the maximum time for the same instances in Table 6. As the constraint programming model does not provide upper bounds, the trivial upper bounds of the instances are reported for this algorithm. Column *CP-IP* gives the results for a mixed algorithm that uses CP for half of the available time and then switches to IP. The IP solver can prune more quickly by starting with the CP solution, and the heuristics in the CPLEX solver use it to find better solutions. The average optimality gaps are large: 46.94%, 17.68%, and 11.84% for IP, CP, and CP-IP respectively which are weaker than the average optimality gap 2.81% obtained by the proposed branch-and-price-and-cut algorithm. Before running IP, we tuned its search parameters using CPLEX's automatic tuning tool. We randomly chose three small instances from each of sets A, B, and C, and CPLEX examined different parameter settings to find the most reliable setting over the nine instances; the time limit was three hours for each instance.

Table 8 presents the improvement in the optimality gap achieved by adding cuts, branching, or both to the enhanced CG. Adding both cuts and branching improves the gap by 5.01% on average. In Appendix 5 of the online supplement we evaluate the different types of branches.

7. Conclusion and Future Research

Integrated operating room planning and scheduling (IORPS) can improve the efficient use of operating theatres by synchronizing the assignment and scheduling of surgeries. Operational details can be considered at the scheduling level, increasing the chance of obtaining a stable schedule that can be successfully implemented. Since patients, surgeons, nurses, anesthesiologists, and operating-room managers all play a role, there are many constraints to take into account. We have developed a branch-and-price-and-cut algorithm for the IORPS which is capable of addressing various kinds

of problem settings. To improve the efficiency of the algorithm, a constraint programming model proposed to solve subproblems is enhanced by some dominance rules and a fast infeasibility-detection algorithm. Our model reduces the number of subproblems at each CG iteration, so we can solve large instances. Extensive computational results demonstrate the superiority of the developed method to a compact mathematical programming model adapted from the literature and show that our algorithm can obtain solutions with an average optimality gap of 2.81% for instances with up to 120 surgeries. Future research could explore the following two issues:

- Two sources of uncertainty could be considered: the durations of the surgeries and the arrival times of emergency cases. Similar problems have been studied under a block-scheduling strategy. However, to the best of our knowledge, only Batun et al. (2011) have studied the problem under an open-scheduling strategy, and they solve instances with at most 11 surgeries. More powerful algorithms are needed for stochastic IORPS under an open-scheduling strategy.
- Integration of the tactical and the operational levels can be also another interesting opportunity where the number of blocks assigned to each surgical department and the start and finish times of these blocks could be determined concurrently with the IORPS.

Supplemental Material

Supplemental material to this paper is available at http://dx.doi.org/10.1287/ijoc.2015.0686.

Acknowledgments

The authors thank the associate editor and two anonymous referees for their meticulous reviews and constructive comments that highly improved the quality of this work. They also gratefully acknowledge financial support from the Fonds de recherche du Québec—Nature et technologies (FRQNT).

References

Atamtürk A (2005) Cover and pack inequalities for (mixed) integer programming. *Ann. Oper. Res.* 139(1):21–38.

Baligh HH, Laughhunn DJ (1969) An economic and linear model of the hospital. *Health Services Res.* 4(4):293–303.

Barnhart C, Hane CA, Vance PH (2000) Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Oper. Res.* 48(2):318–326.

Batun S, Denton BT, Huschka TR, Schaefer AJ (2011) Operating room pooling and parallel surgery processing under uncertainty. *INFORMS J. Comput.* 23(2):220–237.

Blake JT, Carter MW (2002) A goal programming approach to strategic resource allocation in acute care hospitals. *Eur. J. Oper. Res.* 140(3):541–561.

Cardoen B, Demeulemeester E, Beliën J (2009) Sequencing surgical cases in a day-care environment: An exact branch-and-price approach. *Comput. Oper. Res.* 36(9):2660–2669.

Chu G, Stuckey PJ (2009) Minimizing the maximum number of open stacks by customer search. Gent IP, ed. *Principles and Practice of Constraint Programming—CP 2009, Lisbon, Portugal*, 242–257.

Chu G, Stuckey PJ (2012) A generic method for identifying and exploiting dominance relations. Milano M, ed. *Principles and Practice of Constraint Programming—CP 2012, Lecture Notes in Computer Science*, Vol. 7514 (Springer, Berlin), 6–22.

Dantzig GB, Wolfe P (1960) Decomposition principle for linear programs. *Oper. Res.* 8(1):101–111.

Day R, Garfinkel R, Thompson S (2012) Integrated block sharing: A win–win strategy for hospitals and surgeons. *Manufacturing Service Oper. Management* 14(4):567–583.

De la Banda MG, Stuckey PJ, Chu G (2011) Solving talent scheduling with dynamic programming. *INFORMS J. Comput.* 23(1):120–137.

Desaulniers G, Desrosiers J, Solomon MM, eds. (2005) *Column Generation* (Springer Science & Business Media, New York).

Fahle T, Schamberger S, Sellmann M (2001) Symmetry breaking. Walsh T, ed. *Principles and Practice of Constraint Programming—CP 2001*, Lecture Notes in Computer Science, Vol. 2239 (Springer, Berlin), 93–107.

Fei H, Chu C, Meskens N (2009) Solving a tactical operating room planning problem by a column-generation-based heuristic procedure with four criteria. *Ann. Oper. Res.* 166(1):91–108.

Fei H, Meskens N, Chu C (2006) An operating theatre planning and scheduling problem in the case of a block scheduling strategy. *Internat. Conf. Service Systems Service Management* (IEEE, Piscataway, NJ), 422–428.

Fei H, Meskens N, Chu C (2010) A planning and scheduling problem for an operating theatre using an open scheduling strategy. *Comput. Indust. Engrg.* 58(2):221–230.

Focacci F, Milano M (2001) Global cut framework for removing symmetries. Walsh T, ed. *Principles and Practice of Constraint Programming—CP 2001*, Lecture Notes in Computer Science, Vol. 2239 (Springer, Berlin), 77–92.

Gilmore PC, Gomory RE (1961) A linear programming approach to the cutting-stock problem. *Oper. Res.* 9(6):849–859.

Gu Z, Nemhauser GL, Savelsbergh MWP (1998) Lifted cover inequalities for 0–1 integer programs: Computation. *INFORMS J. Comput.* 10(4):427–437.

Guerriero F, Guido R (2011) Operational research in the management of the operating theatre: A survey. *Health Care Management Sci.* 14(1):89–114.

Guinet A, Chaabane S (2003) Operating theatre planning. *Internat. J. Production Econom.* 85(1):69–81.

Hashemi Doulabi SH, Rousseau LM, Pesant G (2014) A constraint programming-based column generation approach for operating room planning and scheduling. Simonis H, ed. *Integration AI OR Techniques Constraint Programming—CPAIOR, Cork, Ireland*, 455–463.

Jebali A, Hadj Alouane AB, Ladet P (2006) Operating rooms scheduling. *Internat. J. Production Econom.* 99(1):52–62.

Korf RE (2004) Optimal rectangle packing: New results. *Internat. Conf. Automated Planning Scheduling* (AAAI, Palo Alto, CA), 142–149.

Leong G, Wilson J, Charlett A (2006) Duration of operation as a risk factor for surgical site infection: Comparison of English and US data. *J. Hospital Infection* 63(3):255–262.

Marques I, Captivo ME, Pato MV (2012) An integer programming approach to elective surgery scheduling. *OR Spectrum* 34(2):407–427.

Prestwich S, Beck JC (2004) Exploiting dominance in three symmetric problems. *Fourth Internat. Workshop Symmetry Constraint Satisfaction Problems, Toronto*, 63–70. http://zeynep.web.cs.unibo.it/SymCon04/SymCon04.pdf.

Proll LG, Smith B (1998) Integer linear programming and constraint programming approaches to a template design problem. *INFORMS J. Comput.* 10(3):265–275.

Roland B, Di Martinelly C, Riane F (2006) Operating theatre optimization: A resource-constrained based solving approach. *Internat. Conf. Service Systems Service Management* (IEEE, Piscataway, NJ), 443–448.

Roland B, Di Martinelly C, Riane F, Pochet Y (2010) Scheduling an operating theatre under human resource constraints. *Comput. Indust. Engrg.* 58(2):212–220.

Ryan DM, Foster BA (1981) An integer programming approach to scheduling. Wren A, ed. *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling* (North-Holland, Amsterdam), 269–280.

Vijayakumar B, Parikh PJ, Scott R, Barnes A, Gallimore J (2012) A dual bin-packing approach to scheduling surgical cases at a publicly-funded hospital. *Eur. J. Oper. Res.* 224(3):583–591.