Scheduling surgical cases in a day-care environment: a branch-and-price approach

Brecht Cardoen, Erik Demeulemeester and Jeroen Beliën

KBI 0724

Scheduling surgical cases in a day-care environment: a branch-and-price approach

Brecht Cardoen, Erik Demeulemeester, Jeroen Beliën

Katholieke Universiteit Leuven, Faculty of Economics and Applied Economics, Department of Decision Sciences and Information Management, Naamsestraat 69, B-3000 Leuven, Belgium, brecht.cardoen@econ.kuleuven.be, erik.demeulemeester@econ.kuleuven.be, jeroen.belien@econ.kuleuven.be

Abstract

In this paper we will investigate how to sequence surgical cases in a day-care facility so that multiple objectives are simultaneously optimized. The limited availability of resources and the occurrence of medical precautions, such as an additional cleaning of the operating room after the surgery of an infected patient, are taken into account. A branch-and-price methodology will be introduced in order to develop both exact and heuristic algorithms. In this methodology, column generation is used to optimize the linear programming formulation of the scheduling problem. Both a dynamic programming approach and an integer programming approach will be specified in order to solve the pricing problem. The column generation procedure will be combined with various branching schemes in order to guarantee the integrality of the solutions. The resulting solution procedures will be thoroughly tested and evaluated using real-life data of the surgical day-care center at the university hospital Gasthuisberg in Leuven (Belgium). Computational results will be summarized and conclusions will eventually be formulated.

Keywords: health care operations, scheduling, column generation, branch-and-price

1 Introduction

In a hospital environment, the operating theater is often identified as a major cost driver. Although these costs typically point at financial issues, they could also be expressed as quality of care or stakeholder satisfaction. Unfortunately, the surgery scheduling process unites many stakeholders, like surgeons, patients or nurses, who may have conflicting preferences and priorities (Hamilton and Breslawski 1994). Due to this complexity, health managers can hardly manage to integrate individual surgeries into a coherent surgery schedule solely based on their experience. At this point, the field of operations research and operations management should assist in the development of an effective and efficient surgery schedule and consequently contribute to the performance of a hospital as a whole (Carter 2002).

In the literature, the surgery scheduling process for elective cases is often seen as a three stage process (Blake and Donald 2002, Beliën and Demeulemeester 2007). In a first stage, one has to determine how much operating room time is assigned to the different surgeons or surgical groups. This stage is often referred to as case mix planning and is situated on a strategic level (Blake and Carter 2002). The second stage, which is tactically oriented, concerns the development of a master surgery schedule. This schedule can be seen as a cyclic timetable that defines the number and type of operating rooms available, the hours that rooms will be open, and the surgeons or surgical groups to whom the operating room time is assigned (Blake, Dexter and Donald 2002). In the third and final stage, individual patients or cases can be scheduled on a daily base. It is on this operational level that our research should be situated.

Methodologies for scheduling individual surgical cases are often based on a two-step procedure. The first step is referred to as advance scheduling (assignment step) and describes the assignment of patients to surgery days. When surgeons are not allowed to switch between operating rooms on a specific surgery day, advance scheduling implicitly determines the operating room in which the surgery of interest will be performed. In the second step, which is referred to as allocation scheduling (sequencing step), the patient population for a specific surgery day has to be sequenced. Solution procedures that distinguish between these two phases can, for instance, be found in Jebali, Alouane and Ladet (2006), Guinet and Chaabane (2003), Marcon, Kharraja and Simonnet (2003), Sier, Tobin and McGurk (1997) or Hsu, de Matta and Lee (2003). A summary of these approaches can be found in Cardoen, Demeulemeester and Beliën (2006). Although we acknowledge the two-step approach, this research will only focus on allocation scheduling. Since we will allow for switches of surgeons between operating rooms, our second step comprises both the assignment of surgeries to operating rooms and the consecutive sequencing of the surgeries within each operating room. This is in correspondence with the surgical case scheduling problem (SCSP) that was examined in Cardoen, Demeulemeester and Beliën (2006). Since this NP-hard optimization problem will also constitute the focus of this paper, we will introduce a short description in Section 2. In this paper, however, we will develop a branch-and-price solution approach instead of generating pure integer programming models and dedicated branch-and-bound algorithms. To the best of our knowledge, the application of column generation and branch-and-price techniques to the scheduling of ambulatory surgical cases on the operational level has not yet been addressed in the literature.

In order to augment the applicability and relevance of the developed algorithms, we maintained a steady cooperation with the surgical day-care center of the university hospital Gasthuisberg in Leuven (Belgium). This medical facility has already been the subject of research in a case study of Beliën, Demeulemeester and Cardoen (2006) and yearly accounts for about 15000 hours of total net operating time and 25000 ambulatory surgeries. De Lathouwer and Poullier (2000) define an ambulatory surgery as a non-emergency procedure which is undertaken with all its constituent elements, i.e. admission, surgery and discharge. Furthermore, this procedure is performed during the time span of a normal working day, thus not exceeding 12 hours including the post-surgical recovery. Using a questionnaire, they revealed a rising trend in ambulatory surgery amongst the OECD members. Although the results are characterized by huge intercountry disparities, Belgium in particular exhibits an increase of 27.6% in the number of ambulatory surgery cases performed between 1995 and 1997. In other words, the increasing share of ambulatory treatments highlights the need for an efficient planning system of the day-care facility.

The remainder of this paper is structured as follows. In Section 2, we will briefly discuss the SCSP and capture its multiple objectives and constraints in a pattern-based integer programming (IP) formulation. Section 3 describes a column generation approach in order to optimize the subsequent linear relaxation. Since column generation cannot guarantee variables to be integer, we will extend this methodology to a broad branch-and-price framework in Section 4. Multiple branching schemes will be developed and combined with the column generation algorithm. In Section 5, a detailed computational experiment will be conducted using data from the surgical day-care center of Gasthuisberg. Finally, in Section 6, conclusions will be formulated and ideas for future research will be mentioned.

2 Problem statement

The current scheduling practice at the day-care center of Gasthuisberg implies that the final surgery schedule is made only one day in advance. Although patients know on which day their surgery will be performed, they are unaware of the time they should enter the hospital until the sequencing step of the scheduling process is finished. This sequencing step, in which the workload of one surgery day is handled, actually boils down to determining for each surgery both its surgery start time and the according operating room in which it will be performed.

Numerous types of decision variables can now be introduced to assist in formulating the multiple objectives and the constraints of the SCSP. The type of decision variables furthermore determines the strategy for constructing the surgery schedules and will hence influence the construction of solution procedures. One strategy, for instance, consists of treating surgeries individually. In particular, a binary decision variable x_{ips} could be introduced that would be equal to 1 if a surgery of type i starts on period p by surgeon s. A detailed overview of this modeling approach can be found in Cardoen, Demeulemeester and Beliën (2006) and will hence not be discussed. The strategy that will be highlighted in this research paper consists of assigning *patterns* to the operating theater. A pattern, which we will also refer to as a column, can be defined as a sequenced group in which all surgeries for one specific surgeon are represented. In particular, the choice whether a pattern will be assigned to the surgery schedule will be determined by the value of its binary decision variable z_{st}. This variable equals 1 when pattern t is chosen for surgeon s.

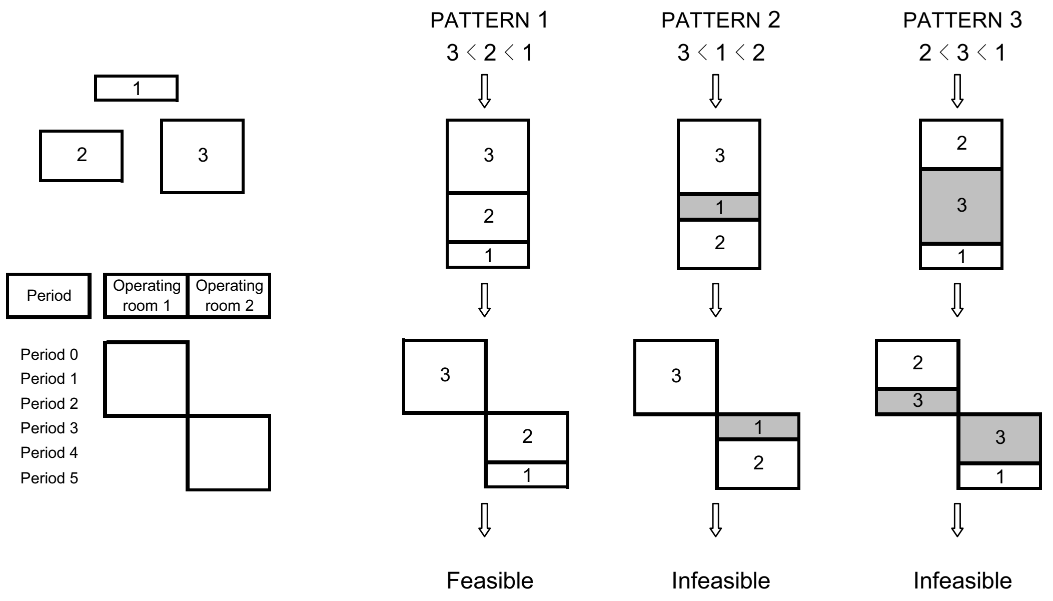


Figure 1: Visualizing patterns as decision variables for building surgery schedules.

We will clarify the concept of patterns by means of the illustrative example that is depicted in Figure 1. In this figure, three surgeries have to be scheduled during the available operating room time of a specific surgeon s. We will assume that the surgery of type 1 represents the surgery of an infected patient (e.g. carrier of the contagious hospital bacteria) so that the operating room needs additional cleaning afterwards. This cleaning, however, is not required when the next patient carries exactly the same infection. Moreover, when an infected patient is the last one to be treated in an operating room, no additional cleaning needs to be performed since the entire operating theater is thoroughly cleaned at closing time. However, when an infected patient is scheduled in a surgery block that is followed by a surgery block of a different surgeon, the cleaning is obligatory and should be entirely performed in the surgery block of the infected patient. In other words, we do assume that with regard to infections, surgeons work independently. As can be seen from the master surgery schedule in Figure 1, the available time for the specific surgeon is divided into two major operating room blocks. The first block in which surgeries can be scheduled is situated in operating room 1 (period 0 to 2), whereas the second major surgery block is situated in operating room 2 (period 3 to 5). Note that if a surgeon is active in multiple operating rooms, his or her operating room blocks are sequential, i.e. the surgeon does not perform multiple surgeries simultaneously. Moreover, if every surgeon is only assigned to one operating room, the sequencing step is restricted to deciding on the surgery start times. Recall that we balance the workload of a pattern with the available operating room time of the surgeon determined by the master surgery schedule. This implies that the patterns we will develop in the example will always contain three surgeries. In a first illustrative pattern, the surgery of type 3 precedes the surgery of type 2. This latter surgery, on its turn, precedes the surgery of type 1 ($3 \prec 2 \prec 1$). When we graphically represent this pattern as shown in the upper representation of the figure, the resemblance with a column becomes apparent. However, for ease of interpretation, we will also visualize the surgeon switch. The first pattern tends to be feasible since the surgeries are nicely spread over the major surgery blocks and the infected patient is scheduled as the last patient. The second pattern, in which $3 \prec 1 \prec 2$, fails to comply with the additional cleaning obligation and is hence infeasible. The third pattern, in which $2 \prec 3 \prec 1$, tends to be infeasible too since a switch in operating rooms takes place while a surgery is ongoing. In other words, there is a dissatisfying spread of the surgeries over the operating rooms. Although the number of restrictions discussed in Figure 1 is limited to 3, namely the incorporation of infections, the spread of surgeries over the operating rooms when surgeon switches occur and the obligation to schedule a surgeon's entire patient population in a pattern, additional restrictions are included in the SCSP. It is, for instance, essential that surgeries do not overlap. This means that surgeries cannot start when the operating room is occupied by any other surgery. Furthermore, it is possible that some patients still have to do some pre-surgical tests (e.g. X-ray) on the day of the surgery. It is, in other words, necessary to schedule the surgery start of these patients after a certain reference period in order to create time to do the required tests. In Section 3, we will return to the generation of patterns and discuss the restrictions that have to be taken into account in detail.

Now that we have an idea of how patterns look like, we can state the SCSP as the integer programming formulation described by Equations 1 to 8 . We refer to Appendix A for a complete overview of the symbols used throughout the equations in this paper.

$$MIN \left(\sum_{s \in S} \sum_{t \in T_s} c_{st} \cdot z_{st} \right) + \sum_{j \in J : j \geq 5} \frac{w_j \cdot \alpha_j}{worstvalue_j - bestvalue_j} - \sum_{j \in J} \frac{w_j \cdot bestvalue_j}{worstvalue_j - bestvalue_j} \tag{1}$$

$$S.T. \quad \left(\sum_{s \in S} \sum_{t \in T_s} a_{1pst} \cdot z_{st} \right) - \alpha_5 \leq 0 \qquad \forall p : min_{r,s} P^{lb}_{rs} \leq p \leq mrp \tag{2}$$

$$\left(\sum_{s \in S} \sum_{t \in T_s} a_{2pst} \cdot z_{st} \right) - \alpha_6 \leq 0 \qquad \forall p : min_{r,s} P^{lb}_{rs} \leq p \leq mrp \tag{3}$$

$$\sum_{s \in S} \sum_{t \in T_s} a_{(e+2)pst} \cdot z_{st} \leq cap_e \qquad \forall e \in E, \; \forall p : min_{r,s} P^{lb}_{rs} \leq p \leq max_{r,s} P^{ub}_{rs} \tag{4}$$

$$\alpha_5 \leq cap_l \tag{5}$$

$$\alpha_6 \leq cap_m \tag{6}$$

$$\sum_{t \in T_s} z_{st} = 1 \qquad \forall s \in S \tag{7}$$

$$z_{st} \in \{0, 1\} \qquad \forall s \in S, \forall t \in T_s \tag{8}$$

Under the assumption that only feasible patterns are added to the mathematical model, a surgery schedule is built by choosing for each surgeon exactly one pattern (Eq. 7). However, picking feasible patterns does not necessarily lead to the generation of a feasible surgery schedule due to the common use of the limited resources. In order to know the aggregate demand per period for a specific resource, we should know this demand for each pattern individually. Since we know the sequence of the surgeries in a pattern, these demands can easily be calculated and are represented by the data parameter a_{opst}. This parameter indicates how many units of resource type o are needed in period p when pattern t is chosen for surgeon s. Three types of resources are explicitly incorporated in the formulation. First, surgeries possibly require medical equipment or instruments during their execution. This implies that the demand per period for each instrument e over all the operating rooms cannot exceed cap_e, which is the total number of instruments of type e available (Eq. 4). We want to stress, however, that the demand for instruments does not solely depend on the simultaneous use of a type of instrument over the different operating rooms. After use, instruments possibly need to be sterilized for several periods and hence cannot be used for subsequent surgeries. This sterilization duration is incorporated during the calculation of the data parameter a_{opst}. Second, when a patient's surgery is finished, he or she is transferred to a first recovery room (recovery phase 1) to get through the critical awakening phase. Since the number of beds in this recovery room is limited, we have to construct a surgery schedule so that the peak demand for recovery beds in phase 1 (=α_5) does not exceed the available capacity cap_l (Eq. 5). A similar reasoning applies to Equation 6. When the patient is conscious and the awakening process in the first recovery phase tends to be normal, the patient is moved to a second recovery room (recovery phase 2) where the patient stays until the surgeon gives permission to leave the day-care hospital. The peak demand for recovery beds in recovery phase 2 (=α_6) cannot exceed its available capacity cap_m. Equations 2 and 3 will assist in determining the peak number of beds that are used in recovery phase 1 and recovery phase 2.

When a combination of patterns can be found that satisfies the above constraints, a feasible surgery schedule is constructed. We are, however, interested in finding the best surgery schedule with respect to multiple objectives. In particular, we want to generate a schedule in which surgeries of children ($\alpha_1 = \sum_{n\in N} \Theta^{child}_{type_n} \cdot v_n$) or prioritized patients ($\alpha_2 = \sum_{n\in N} \Theta^{prior}_{type_n} \cdot v_n$) are performed as early as possible. Furthermore, we want to incorporate the travel distance of patients. In particular, we will try to schedule the surgery start of travel patients after a certain reference period in order to provide sufficient time to get to the day-care center ($\alpha_3 = \sum_{n\in N: v_n < travelref} \Theta^{travel}_{type_n}$). We also want to minimize the number of periods that patients have to stay in recovery after the closure of the day-care center, since this would result in unplanned (and hence costly) hospitalizations or overtime for the nursing personnel ($\alpha_4 = \sum_{n\in N} max[0, v_n + k_{type_n} + l_{type_n} + m_{type_n} - 1 - P^{ub}]$). Finally, we are interested in minimizing the peak number of beds used in recovery phase 1 ($= \alpha_5$) and recovery phase 2 ($= \alpha_6$) in order to smoothen the bed occupancy and hence level the workload for the nursing personnel. Although $\sum_{j\in J} \alpha_j$ would be a straightforward function to optimize, we opt to combine the 6 objectives as represented in Equation 9.

$$\sum_{j\in J} w_j \cdot RFI_j = \sum_{j\in J} w_j \cdot \left(\frac{\alpha_j - bestvalue_j}{worstvalue_j - bestvalue_j} \right) \tag{9}$$

Since several objectives are expressed in different units, a trade-off has to be defined between the units. This decision, however, can be very subjective and hence difficult to argument. Therefore, we propose in Equation 9 a multi-objective function that is based on the *room for improvement* for each objective j (RFI_j). Since we know the population of surgeries that has to be scheduled (including the idle periods), we are able to calculate for each single objective the best and the worst possible value that could be obtained. These extreme values can consecutively be used as indicated in Equation 9 to generate a relative measure of quality. This implies that we do not have to struggle with units anymore: we get for each objective a value that is in the range [0,1] and is easy to interpret since the RFI_j indicates how much worse the value for objective j is with regard to its best value. If RFI_j equals 0, we cannot improve objective j any further. Calculation of the extreme values for α_j $(j \in J)$ is done during instance generation using the ILOG CPLEX 8.1 optimization library. Further information on this topic can be found in Cardoen, Demeulemeester and Beliën (2006). Using the transformation described above, all objectives will be gradually optimized to the same level and will somehow be comparable to each other. It is unlikely, however, that the objectives are of equal importance to the human planner. Thus, we should incorporate a possibility for the planner to express the relevance of the different objectives. This can easily be done by assigning a weight w_j to objective j. Note that the weights only indicate the preferences of the scheduler. Note that when the sum of the weights equals 1, the multiple objective function still has a value that is in the range $[0, 1]$. As can be seen below, Equation 1 can now easily be deducted from Equation 9. Since the last term in Equation 1 is a constant, it will be neglected during optimization. Note that only objective $j \in J : j \leq 4$ is incorporated in c_{st}. For these objectives, we have that $\alpha_j = \sum_{s\in S} \sum_{t\in T_s} (\tilde{c}_{jst} \cdot z_{st})$. This, though, does not apply to the bed leveling objectives. When a surgery schedule, for instance, consists of two columns and each column has a peak demand for beds in recovery phase 1 equal to 3, it is not guaranteed that $\alpha_5 = 3 + 3 = 6$. Actually, $\alpha_5 = 6$ only occurs when both peak demands are established simultaneously for at least one period.

$$
\begin{aligned}
&\sum_{j\in J} w_j \cdot \left(\frac{\alpha_j - bestvalue_j}{worstvalue_j - bestvalue_j} \right) \\
&= \sum_{j\in J: j\leq 4} w_j \cdot \frac{\left(\sum_{s\in S} \sum_{t\in T_s} \tilde{c}_{jst} \cdot z_{st}\right) - bestvalue_j}{worstvalue_j - bestvalue_j} + \sum_{j\in J: j\geq 5} w_j \cdot \frac{\alpha_j - bestvalue_j}{worstvalue_j - bestvalue_j} \\
&= \sum_{j\in J: j\leq 4} w_j \cdot \frac{\left(\sum_{s\in S} \sum_{t\in T_s} \tilde{c}_{jst} \cdot z_{st}\right)}{worstvalue_j - bestvalue_j} + \sum_{j\in J: j\geq 5} \frac{w_j \cdot \alpha_j}{worstvalue_j - bestvalue_j} - \sum_{j\in J} \frac{w_j \cdot bestvalue_j}{worstvalue_j - bestvalue_j} \\
&= \sum_{s\in S} \sum_{t\in T_s} \left(\sum_{j\in J: j\leq 4} \frac{w_j \cdot \tilde{c}_{jst}}{worstvalue_j - bestvalue_j} \cdot z_{st} \right) + \sum_{j\in J: j\geq 5} \frac{w_j \cdot \alpha_j}{worstvalue_j - bestvalue_j} - \sum_{j\in J} \frac{w_j \cdot bestvalue_j}{worstvalue_j - bestvalue_j} \\
&= \left(\sum_{s\in S} \sum_{t\in T_s} c_{st} \cdot z_{st} \right) + \sum_{j\in J: j\geq 5} \frac{w_j \cdot \alpha_j}{worstvalue_j - bestvalue_j} - \sum_{j\in J} \frac{w_j \cdot bestvalue_j}{worstvalue_j - bestvalue_j}
\end{aligned}
$$

In order to solve the integer programming formulation that is described by Equations 1 to 8, we could enumerate for each surgeon all the feasible columns, calculate for each column the corresponding a_{opst} and c_{st} and consecutively add the columns to the model. However, when the number of surgeries that have to be scheduled is substantial, the number of columns easily explodes. When the workload for one surgeon is, for instance, equal to 10 surgeries, we possibly have to enumerate $10! = 3628800$ columns. With up to 8 operating rooms, the number of variables is too large to be handled efficiently. A better approach would be to generate and add only those variables to the model that seem promising with respect to the objective function. This can be done using a column generation approach, as will be explained in the next section.

3 Linear relaxation through column generation

In order to solve the linear relaxation of the SCSP to optimality using a column generation approach, the scheduling problem is decomposed into a master problem and a subproblem (Barnhart et al. 1998, Desaulniers et al. 2005). This decomposition, which constitutes the spine of the column generation optimization loop, is depicted in Figure 2. The master problem corresponds with the formulation stated by Equations 1 to 8, though now we will relax the integrality constraint of Equation 8. Since this formulation will be solved using only a subset of the existing columns, we will refer to this problem as the *restricted master problem* (RMP).

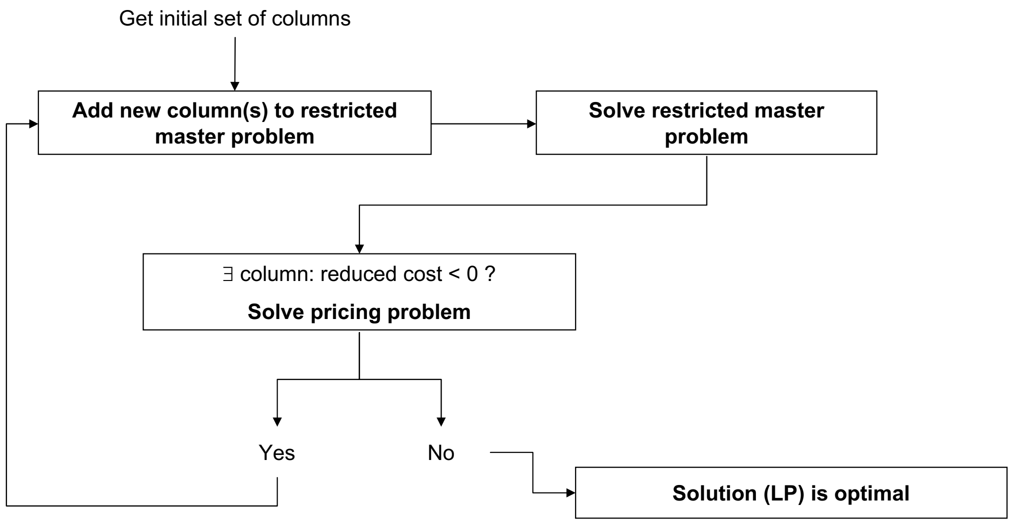

Get initial set of columns
Add new column(s) to restricted master problem
Solve restricted master problem
∃ column: reduced cost < 0 ?
Solve pricing problem
Yes
No
Solution (LP) is optimal

Figure 2: Visualizing a column generation optimization loop.

In order to verify whether the solution obtained by solving the RMP also optimizes the linear relaxation of the SCSP (i.e. when we would take all existing columns into account), a subproblem needs to be solved. We will refer to this subproblem as the *pricing problem.* In particular, we will try to generate a column that is characterized by a negative reduced cost. When we assume that ρ_p, σ_p, π_{ep} and λ_s respectively represent the dual prices of restrictions 2, 3, 4 and 7, the reduced cost of a column t for a surgeon s can be specified by Equation 10.

$$RC_{st} = c_{st} - \lambda_s - \sum_{p=min_{rs}P_{rs}^{lb}}^{mrp} (\rho_p \cdot a_{1pst} + \sigma_p \cdot a_{2pst}) - \sum_{e \in E} \sum_{p=min_{rs}P_{rs}^{lb}}^{max_{rs}P_{rs}^{ub}} \pi_{ep} \cdot a_{(e+2)pst} \tag{10}$$

When such a column can be found, it should be added to the subset of variables and the optimization loop should be repeated. However, when no column prices out (i.e. has a negative reduced cost), the solution to the RMP also optimizes the linear relaxation of the SCSP and the column generation loop terminates.

Since we can only obtain the dual prices of the restrictions when a feasible solution can be found for the RMP, this master problem will be initiated with a set of binary dummy variables or supercolumns. In particular, a supercolumn z_{s1} is added for each surgeon s and is similar to an ordinary column, except that $\forall p : min_{r,s}P_{rs}^{lb} \leq p \leq mrp$ and $\forall e \in E$: $a_{1ps1} = a_{2ps1} = a_{(e+2)ps1} = 0$. Furthermore, $\forall s \in S : c_{s1} = \infty$. In other words, since these columns do not consume any resources, their value can easily be adapted ($0 \leq z_{s1} \leq 1$) in order to satisfy the convexity constraints (Eq. 7) and hence to ensure feasible solutions. This, however, comes at a very high cost since a schedule built with supercolumns is unrealistic in nature, i.e. we will eventually have to find a schedule in which z_{s1} equals 0 for each surgeon.

In Figure 2, there is no need to solve the pricing problem for every $s \in S$. We could resolve the RMP whenever at least one column t for a certain surgeon s could be found with $RC_{st} < 0$. However, many variations are allowed in order to solve the linear relaxation to optimality. When multiple columns price out, one can choose on the number of columns to be added to the RMP. We could, for instance, try to solve the pricing problem for every $s \in S$ and consequently add for each surgeon s a column t to the RMP when $RC_{st} < 0$. Since it is even allowed to add multiple columns for one specific surgeon, it should be clear that the set of newly generated columns does not have to be limited to those exhibiting the most negative reduced cost. In Section 5.2, we will compare several pricing strategies on their computational efficiency. In the remainder of this section, we will elaborate on two algorithmic approaches to solve the pricing problem, namely a dynamic programming approach and an integer programming approach. Both algorithms generate, for a given surgeon, the column that results in the most negative reduced cost.

3.1 The pricing problem: dynamic programming

3.1.1 A recursive formulation

Dynamic programming (DP) is a solution methodology that decomposes a problem into a nested family of smaller and hence more tractable subproblems (Bellman 1957). In order to solve the pricing problem of a surgeon s, we will distinguish between $|N_s| + 1$ stages ($0 \leq g \leq |N_s|$). The numerical index of a stage represents the number of surgeries that are already scheduled at that point. Each stage has a number of states associated with it which indicate the types of surgeries that are already scheduled. This combinatorial problem can be regarded as a permutation. Since multiple patients can have the same surgery type (i.e. there are duplicates), describing states in terms of surgery types diminishes the symmetry and hence contributes to the computational efficiency of the algorithm since less permutations have to be considered (Skiena 1998). In Table 1, the enumeration of stages and states is illustrated by means of an example. In the example, 4 surgeries need to be scheduled for surgeon s ($|N_s|_{type_n=1} = 1$, $|N_s|_{type_n=2} = 2$ and $|N_s|_{type_n=3} = 1$), which results in 5 stages and 12 states. In general, stage 0 only contains the

Table 1: Enumerating stages and states of a dynamic programming example.

Stage	States
0	{}
1	{1}, {2}, {3}
2	{1,2}, {1,3}, {2,2}, {2,3}
3	{1,2,2}, {1,2,3}, {2,2,3}
4	{1,2,2,3}

empty state $h^{\emptyset}$: $H_0 = \{h^{\emptyset}\}$, whereas stage $|N_s|$ only consists of a state in which all surgeries are scheduled. It should be clear that the transition from a state h at stage g to a state h' at stage $g+1$ corresponds to adding one specific surgery of type i to the surgery schedule: $i \in I_s : |h'|_{value=i} - |h|_{value=i} = 1$. In Table 1, the transition $\{\} \rightarrow \{1\}$, for instance, corresponds with starting a surgery of type 1 at period $p = \min_r P^{lb}_{rs}$. Note that $h \rightarrow h'$ is only allowed when $\forall i \in I_s : |h|_{value=i} \leq |h'|_{value=i}$. In order to determine for surgeon s a column t that exhibits the most negative reduced cost RC^*_s, we can now formulate a recursive function:

$$RC^*_s = \min_{h \in H_1} \Big\{ C(h^{\emptyset}, h) + F_1(h) \Big\} - \lambda_s \tag{11}$$

$$F_g(h) = \min_{h' \in H_{g+1} : h \subset h'} \Big\{ C(h, h') + F_{g+1}(h') \Big\} \tag{12}$$

In general, $F_g(h)$ represents the minimum cost incurred for the completion of the surgery schedule with surgery types that still have to be scheduled during stages $g+1$ up to stage $|N_s|$, given that state h at stage g is realized. Note that for state $h \in H_{|N_s|} : F_{|N_s|}(h) = 0$. When we return to Table 1 and assume that state $\{1,2\}$ at stage 2 is realized, we could complete the surgery schedule by either adding the sequence $2 \prec 3$ (i.e. adding a surgery of type 2 at stage 3 and a surgery of type 3 at stage 4) or either adding the sequence $3 \prec 2$ (i.e. adding a surgery of type 3 at stage 3 and a surgery of type 2 at stage 4). Once both alternatives are evaluated, $F_2(\{1,2\})$ could be set equal to the cost of the alternative characterized by the lowest cost. Registering these minimum costs entails an important contribution to the computational efficiency of the algorithm. During the algorithmic search, a state h at stage g could possibly be visited multiple times. The state $\{1, 2\}$, for instance, could be reached from two states at stage 1, namely $\{1\}$ and $\{2\}$. Saving the values of $F_g(h)$ during the enrolment of the algorithm consequently avoids the need to recompute subproblems that already have been solved once. Michie (1968) referred to this concept as *memoization.*

The transition from state h to state h' comes at a cost equal to $C(h, h')$ and reflects the cost for starting a surgery of type $i \in I_s : |h'|_{value=i} - |h|_{value=i} = 1$ at period $p = \min_r P^{lb}_{rs} + \sum_{i \in h} |h|_{value=i} \cdot k_i$. Two major cost components can be distinguished in $C(h, h')$. On the one hand, there are costs related to the change in the objective function. Since only one patient n with a surgery type equal to i and $v_n = p$ is scheduled, these costs are calculated as $\sum_{j \in J: j \leq 4} (w_j \cdot \alpha_j)/(worstvalue_j - bestvalue_j)$. With respect to the calculations of α_j, we assume that $N = \{n\}$. On the other hand, $C(h, h')$ should also reflect costs associated with the non-positive dual prices of restrictions 2, 3 and 4, so that $C(h, h')$ is augmented by $- \sum_{p'=p+k_i}^{p+k_i+l_i-1} \rho_{p'}$ $- \sum_{p'=p+k_i+l_i}^{p+k_i+l_i+m_i-1} \sigma_{p'}$ $- \sum_{e \in E} \sum_{p'=p}^{p+k_i+stere_e} \pi_{ep'}$. However, when $i \in I_{presurg}$ and $p < presurgref$, we have that $C(h, h') = \infty$. This implies that we are not allowed to start a surgery before the reference period *presurglimit* when pre-surgical tests are incomplete on the day of surgery. Furthermore, $C(h, h') = \infty$ when surgeon s has to switch between operating rooms when the surgery of the patient is still ongoing.

The dynamic programming formulation of Equations 11 and 12 is accurate for optimizing pricing problems in which infections do not occur. This formulation, however, does not hold when infections come into play. One problem, for instance, arises with the application of the memoization feature. We can illustrate this using the example introduced in Table 1 once again. Suppose we arrived in state $\{1, 2\}$ at stage 2 and $I_{bact} = \emptyset$. Furthermore $F_2(\{1{,}2\})$ is already calculated and equals the cost associated with the sequence $3 \prec 2$. No matter how we arrived in state $\{1, 2\}$ (i.e. by $\{\} \rightarrow \{1\} \rightarrow \{1, 2\}$ or $\{\} \rightarrow \{2\} \rightarrow \{1, 2\}$), $F_2(\{1{,}2\})$ can always be used as the optimal value of the subsequent subproblem. However, when $I_{bact} = \{1\}$, the additional cleaning time equals 1 period and the surgeries of type 2 represent idle periods, it would be incorrect to use $F_2(\{1{,}2\})$ as the optimal value of the subproblem according to the path $\{\} \rightarrow \{2\} \rightarrow \{1, 2\}$, since the infected surgery would immediately precede a surgery of type 3 instead of the obliged idle period. One other problem of scheduling infected patients would be that these decisions do not only restrict the states that can be reached in the next stage, but also those in further stages (i.e. as long as the obliged cleaning session is not finished). In order to incorporate infections accurately, Equations 11 and 12 should hence be thoroughly modified. A formulation of this generalized dynamic programming formulation can be found in Appendix B.

3.1.2 Time complexity

In order to determine the efficiency of the DP algorithm, we will identify the appropriate complexity class. In this section, we assume that no patients $n \in N_s$ can be identified for which the surgery type is equal. It should be noted that this assumption leads to a worst-case analysis, since duplicate surgeries would further reduce the calculation effort needed to solve the pricing problem.

We will focus on the number of recursive calls, which clearly depends on the problem size $|N_s|$, in order to determine the running time of the algorithm. Since a recursive call can be interpreted as a visit to a state at a further stage, defining the running time boils down to determining how much progressive state visits are executed during the algorithmic search. In particular, two components need to be calculated. On the one hand, we have to determine the number of states present at stage g $(= |H_g|)$. On the other hand, we have to determine the number of visits to a state h at stage g. We assume that no visit is required to the empty state $h^{\emptyset}$ at stage 0, since this is the starting point of the algorithm.

With respect to the first component, the number of states at stage g is calculated as $|N_s|!/[(|N_s| - g)! \cdot g!]$. Since the state space is divided into $|N_s| + 1$ stages, the total number of states needed to solve the entire pricing problem is equal to $\sum_{g=0}^{|N_s|} |N_s|!/[(|N_s| - g)! \cdot g!]$. Since this number actually represents the enumeration of all possible subsets of surgeries, it is equal to $2^{|N_s|}$. With respect to the second component, the number of visits to a state at stage g equals g. This is a consequence of saving intermediate results, i.e. a consequence of the memoization feature. Note that when this feature is turned off, the number of visits to a state at stage g increases to $g!$. This latter approach would result in a complete enumeration of all the paths and could hence be seen as a brute-force approach with a running time equal to $O(n!)$. Since we do apply the memoization feature, the number of computational steps needed to solve a pricing problem of size $|N_s|$ can be determined through Equation 13.

$$\sum_{g=0}^{|N_s|} g \cdot \frac{|N_s|!}{(|N_s|-g)! \cdot g!} = |N_s| \cdot \frac{1}{2} \cdot 2^{|N_s|} = |N_s| \cdot 2^{|N_s|-1} \qquad (13)$$

From Equation 13, we may conclude that the dynamic programming algorithm runs in exponential time. Although algorithms with an exponential running time are globally considered to be inefficient, it should be noted that if the problem size is small, the complexity class might not matter very much (Standish 1994). This implies that satisfying computational results can still be obtained for the pricing problem since $|N_s|$ is limited to 15 surgeries (see Section 5.2). Moreover, although the running time is exponential, it is far more efficient than the factorial approach. Solving a pricing problem of size $|N_s| = 15$ results in 245760 computational steps in the former case, whereas about 3.5×10^{12} steps are needed in the latter case.

3.2 The pricing problem: integer programming

Alternatively, we can also state the pricing problem as an integer programming formulation and solve it using a commercial IP solver. The IP formulation is actually a restricted version of the preprocessed IP model introduced in Cardoen, Demeulemeester and Beliën (2006). Note that the binary decision variable x_{ips}, which equals 1 if a surgery of type i starts on period p by surgeon s, also exhibits the advantage of symmetry tackling since it is based on surgery types and not on patients.

There are two main reasons why we developed, next to the dynamic programming approach, this second pricing algorithm. First, we want to use it as a benchmark in order to investigate the efficiency of the dynamic programming formulation (see Section 5.2). Second, when the IP pricing formulation itself turns out to perform well, we could use it in the branch-and-price algorithms in order to branch on the column variables z_{st}. When we can prevent columns to price out multiple times only by making slight modifications (e.g. changing cost coefficients), the structure of the pricing problem remains stable. One major inconvenience that is related to branching on the column variables, though, is that the pricing structure has to be adapted. This, however, is a complicated task with respect to the dynamic programming formulation, whereas it can easily be done for the IP pricing algorithm. When we prohibit a column z_{st} (i.e. $\forall n \in N_s : \upsilon_n$ is known) to be generated during the column generation optimization loop, we only have to add Equation 14 to the mathematical formulation of the according pricing problem.

$$\sum_{n \in N_s} x_{type_n, v_n, s} \leq |N_s| - 1 \tag{14}$$

4 A branch-and-price approach

Since the column generation loop is optimizing the linear relaxation of the SCSP, the optimal values for the column variables do not necessarily equal 0 or 1. In order to get integer values for these column variables, we will embed the column generation optimization loop within an enumerative branch-and-bound framework. This methodology is referred to as branch-and-price (Desaulniers et al. 2005).

The solution of the initial column generation optimization loop corresponds to solving the root node. When the solution values of the column variables are integer and $\forall s \in S : z_{s1} = 0$ (i.e. no supercolumns are chosen), the SCSP is immediately solved to optimality. Any other solution that does not incorporate the presence of supercolumns, on the contrary, leads to the introduction of new nodes on a next level (two nodes when the decision scheme is binary). The initiation of a new node coincides with the introduction of a new branching restriction. Column variables z_{st} that do not comply with the appropriate restrictions (i.e. restrictions introduced in the path from the root node to the current node) will have a solution value equal to 0. In the current node, a new column generation optimization loop is executed. Columns that price out must satisfy the set of restrictions. When the LP solution surpasses the objective value of the provisional best solution, the algorithm backtracks. Note that backtracking also occurs whenever supercolumns are present in the optimal LP solution of a node. When both backtracking rules do not apply, the algorithm will check the variables for integrality. Integrality will lead to the registration of the surgery schedule and is followed by backtracking. When the variables are fractional, new branching restrictions are added, leading to the introduction of new nodes. In the remainder of this section, specific features of the branch-and-price methodology, related to the branching strategy, branching schemes and speed-up techniques, will be discussed.

4.1 Branching strategy

Based on the selection of the node to branch from next, two general strategies can be defined, namely a depth-first or backtracking strategy and a best-first or skiptracking strategy. Since both approaches tend to have their strengths and their weaknesses (Demeulemeester and Herroelen 2002), we will implement the depth-first as well as the best-first strategy during the computational experiment (see Section 5.3). However, since the occurrence of at least one feasible solution is crucial on the operational surgery level, we can question whether a best-first strategy will be effective. When the discrepancy between the optimal linear relaxation of the root node and the optimal solution of the SCSP is large, there is an increased probability that time will be too short to search the entire tree and consequently even find a feasible solution. In Table 2, we compare the absolute gap between the linear relaxation of the SCSP and its optimal solution with respect to 3 different mathematical formulations. Both the basic IP and the preprocessed IP were introduced in Cardoen, Demeulemeester and Beliën (2006), whereas the column IP corresponds to the formulation described in this paper. The test set used for this experiment will be described in Section 5.1. Both the average and the median gap are the smallest when the SCSP is formulated in terms of patterns or columns. Moreover, the optimal linear relaxation of the column IP tends to be more stable. These findings are in line with Barnhart et al. (1998), who state that the relaxation of a MIP can often be tightened by a reformulation that involves a huge number of variables. In short, when the relaxation is tight, the optimal solution could be near and the search of a best-first algorithm could potentially be finished in time. Note that feasible solutions could be encountered even when a best-first branch-and-price algorithm ends prematurely. This is the case when the intermediate solution of the RMP during the column generation optimization loop is constituted by integer variables.

4.2 Branching schemes

In Section 3.2, we proposed to branch on the column variables in order to partition the solution space and eliminate the occurrence of fractional column variables. This would imply that we

Table 2: Absolute gap between the linear relaxation of the SCSP and the optimal solutions.

	average	median	standard deviation
Basic IP	0.056	0.042	0.051
Preprocessed IP	0.043	0.035	0.039
Column IP	0.027	0.018	0.031

need to identify a column $z_{st} : 0 < z_{st} < 1$ and fix z_{st} either to 1 (left branch) or to 0 (right branch). However, thorough testing of the IP pricing algorithm revealed a weak performance (see Section 5.2), so that we will limit the focus on branching schemes in which the pricing problem is only slightly modified using the cost coefficients and thus the same algorithmic solution approach can be applied. In particular, four binary branching schemes will be presented and implemented in which branching restrictions will be formulated in terms of the individual surgeries. It can be shown that these schemes are complete.

In a first scheme, we will fix a surgery of type i to start on a period p for a surgeon s ($x_{ips} = 1$) in the left branch, whereas $x_{ips} = 0$ in the right branch. Since there are far more columns for which the proposition of the right branch holds, this branching scheme will result in a highly unbalanced tree. Although this restricts the ability to prove the optimality of a solution, it should allow for a quick detection and improvement of feasible solutions. The second branching scheme is similar to the first, except that we do not oblige a surgery to start on a specific period but that a surgery of type i for surgeon s should be in process on period p (left branch). The contrary obviously holds for the right branch. Although the freedom to slightly shift the starting period of the surgery type should favor the balancing of the tree, this branching scheme is still unbalanced. Therefore, we thought of a third branching scheme in which a surgery of type i for surgeon s has to be started before or on period p (left branch), whereas it should start after period p in the right branch. In this branching scheme, however, a problem occurs when multiple patients have the same type of surgery since the branching restriction would apply to all patients with a surgery type equal to i. In order to overcome this problem, we specified the branching restrictions on the patient level instead of on the surgery type level. This implies, for instance, that the surgery of patient n for surgeon s has to be started before or on period p (left branch). Moreover, precedence relations between patients with a common surgery type were introduced in order to avoid symmetry. Next to the time-based branching schemes, we will also introduce a sequence-based branching scheme. In particular, we will oblige a surgery type to be scheduled in a specific position, e.g. a surgery of type 10 has to be scheduled as the fifth surgery (left branch). In the right branch, only columns are taken into account in which the surgery type does not appear on the specific position. We should mention that many other branching schemes, for instance hybrid or non-binary schemes, could be developed and tested. This, however, constitutes an area for future research.

The information needed in order to specify the appropriate branching restriction is determined through the comparison of two fractional columns for a common surgeon. We do have to decide, though, which columns will be picked and which $i - p$ combination will be chosen when the surgery sequences of the fractional columns differ on multiple places. Intuitively, it seems reasonable to select columns characterized by the most fractional value (i.e. close to 0.5 and thus balanced) or highest value (i.e. close to 1 and thus an extremely valuable column). Preliminary computational results indicated that selecting the columns with the highest fractional value and choosing the $i - p$ information that corresponds with the earliest conflict, performed better than a branch-and-price approach in which choices (most fractional value, highest fractional value, earliest conflict and latest conflict) were made randomly.

4.3 Speed-up techniques

In order to upgrade the performance of the branch-and-price algorithms, several speed-up techniques could be developed and implemented. The techniques introduced in this paper consist of an initial solution, a lower bound in the column generation optimization loop and the elimination of columns along the branching tree.

4.3.1 Initial solution

The expected contribution of introducing an initial solution is twofold. On the one hand, it should enable the algorithm to fathom branches that lead to a solution value that is larger than the initial solution value (depth-first). On the other hand, it should augment the probability of finding at least one feasible solution to the SCSP. This last feature is especially complementary with the best-first branching strategies.

Different initial solution algorithms can be conceived. One could, for instance, introduce the iterated branch-and-bound procedure described in Cardoen, Demeulemeester and Beliën (2006). In this paper, however, we opt for a column based heuristic, i.e. we will generate a limited set of columns and try to combine them into a feasible schedule using a commercial IP solver. The computation time for generating an initial solution is limited to 60 seconds and should be divided over the column generation part and the schedule generation part. The impact of this time allocation on the initial solution quality will be examined in Section 5.3.1. Moreover, attention will be paid to the number of columns to be generated. Often, a feasible schedule cannot be determined solely based on the columns generated during the column generation optimization loop of the root node. In Section 5.3.2, we will examine the performance of this initial solution heuristic when the allowed computation time is increased to 300 seconds, which is comparable to all other exact procedures introduced in this paper.

4.3.2 Lower bound calculation

One of the well-known difficulties with column generation is that a large number of iterations is required in order to prove that the RMP is solved to optimality. In the literature, one often refers to this phenomenon as the tailing-off effect (e.g. Vanderbeck and Wolsey 1996). A lower bound, though, can be specified in order to prematurely stop the column generation optimization loop without any risk of missing the LP optimum (e.g. Hans 2001 or Beliën 2006). The calculation of the lower bound, which is also referred to as the Lagrangian lower bound, starts with solving the RMP. Next, for each surgeon $s \in S$, a pricing problem needs to be solved in which the column with the most negative reduced cost has to be determined. When the pricing problem of surgeon s results in a column t: $RC_{st} < 0$, its value is added to the solution value of the RMP. Remark that this summation decreases the LP solution value and that at most $|S|$ summations will be performed. If the modified LP solution value is larger than the best integer solution value that was already obtained along the tree, the column generation optimization loop can be safely terminated.

4.3.3 Column elimination

When we focus on the depth-first branching strategy, columns that have been generated along the tree may become superfluous at a certain point in time. In other words, no better surgery schedule will be detected in which one of the superfluous columns is chosen. This can easily be shown by the branching tree of Figure 3. In this figure, the numbers of the nodes indicate the tree generation process. Arriving in node 5, for instance, implies that all column variables z_{st} that are generated by transition to grey nodes are set to 0 since they do not comply with the active branching restriction. Moreover, we will never need them in the further development of the tree. Loading the RMP with a considerable amount of variables (even when they are set equal to 0) increases the required solution time. It should hence be beneficial to remove the superfluous columns from the list instead of fixing the values to 0.

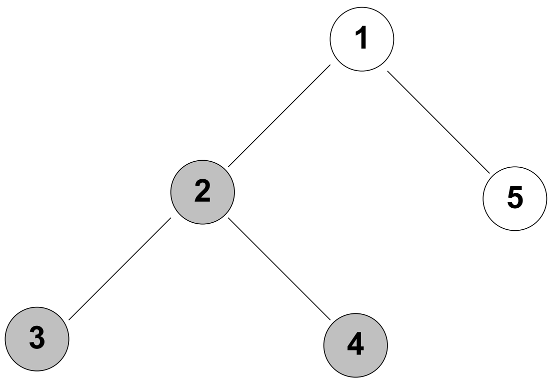


Figure 3: Branching tree for visualizing superfluous columns.

With respect to the best-first branching strategy, the elimination of columns is less interesting at first sight. A trade-off will exist between the time gained by solving a RMP with few columns and the time needed to regenerate columns that were already deleted during the tree generation process. In Section 5, though, we will indicate that the column elimination feature could be beneficial for intertwining branch-and-price algorithms, especially when they are best-first oriented, with an intermediate and recurrent heuristic procedure without losing the exact nature of the algorithms.

5 Computational experiment

A detailed computational study of the column generation optimization loop and the branch-and-price procedures will constitute the focus of this section. All algorithms are written in MS Visual C++.NET and are linked with the ILOG CPLEX 8.1 optimization library when needed (ILOG, 2002). The computational experiment was executed on a 2.8 GHz Pentium 4 PC with the Windows XP operating system.

5.1 Test set

In total, 8 design patterns · 2 instances · 14 sizes = 224 test instances were generated using patient-related data of the surgical day-care center of the university hospital Gasthuisberg in Leuven. The size of an instance indicates the number of surgeries that has to be scheduled (i.e. 20, 25,... up to 85 surgeries), whereas the design pattern adds a specific structure to the instance. This structure may differ on 3 levels ($2^3 = 8$ design patterns). First, a distinction is made in the assignment of surgeries over the operating theater. On the one hand, the workload can be unequally divided over the operating rooms or the surgeons. Note that the number of surgeries that can be performed by a single surgeon s is limited to 15. On the other hand, surgeries can be nicely spread over the entire operating room complex. Second, we distinguish between master surgery schedules with frequent switches of surgeons in the operating rooms and schedules in which a switch is rare. Third, the objectives can be equally weighted or not. The sum of the weights, however, always equals 1.

We will use patient-related data gathered in 2005 and make a distinction between 17 of the most important medical disciplines or entities (e.g. orthopaedics, gynaecology, dermatology,...). For each medical discipline and their surgery types we can calculate the probability of occurrence. Furthermore, for each surgery type, the planned surgical duration (including anaesthesia, skin-to-skin time, after care and cleaning), the planned time in recovery phase 1 and phase 2, the required bottleneck instruments and the corresponding sterilization time is known. All time-related data are expressed in five-minute periods. In contrast with the recovery length, the duration of a surgery is at least equal to one period. Probabilities concerning children, priority, travel distance, pre-surgical tests and infections, however, are only occasionally registered up to now and hence suggested by the health manager, based on his experience. Further information on the instance generation process can be found in Cardoen, Demeulemeester and Beliën (2006).

5.2 Column generation evaluation

In Section 3, we proposed either a dynamic programming approach (DP) or an integer programming approach (IP) in order to solve the pricing problems. We also indicated that there are many ways to combine the pricing problems and the RMP into a column generation optimization loop. In this section, multiple combinations will be tested on their computational efficiency, though most will be dynamic programming oriented. A summary of the diverse versions can be found in the list below:

- DP1: Within each iteration of the column generation optimization loop, a pricing problem is solved for each $s \in S$ in which the column with the most negative reduced cost is generated. Columns z_{st}: $RC_{st} < 0$ are added to the RMP, which is consecutively resolved.

- DP2: For a surgeon s, the column z_{st} with the most negative reduced cost is generated and added to the RMP (when $RC_{st} < 0$) which is instantly resolved. New dual prices are introduced for generating a column for a next surgeon.
- DP3: For a surgeon s, the column z_{st} with the most negative reduced cost is generated and added to the RMP (when $RC_{st} < 0$) which is instantly resolved. New dual prices are introduced for generating a new column for the same surgeon s. This sequence is repeated until no further columns price out for this surgeon. Next, columns are generated for a subsequent surgeon.
- DP4: A pricing problem is solved for each $s \in S$ in which a set of columns with negative reduced costs is generated. During the generation phase, only columns will be added to the set with a reduced cost that is smaller than the minimum of the reduced costs related to columns that are already present in the set. This implies that the column with the most negative reduced cost will always be included in the set. Next, the columns in the set are added to the RMP which is then resolved.
- DP5: A pricing problem is solved for each $s \in S$ in which a set of columns with negative reduced costs is generated. The column with the most negative reduced cost is included in this set. In contrast with DP4, no restriction applies to the magnitude of the reduced costs. Next, the columns in the set are added to the RMP which is then resolved.
- IP: Within each iteration of the column generation optimization loop, a pricing problem is solved for each $s \in S$ in which the column with the most negative reduced cost is generated. Columns z_{st}: $RC_{st} < 0$ are added to the RMP, which is consecutively resolved.

In Table 3, some descriptive statistics can be found to compare the approaches on their computational efficiency. In particular, interest is given to the solution time for solving the column generation optimization loop of the root node, i.e. when no branching has yet occurred. It is clear that all dynamic programming algorithms outperform the integer programming approach. Since the solution times for the IP procedure are in general quite high, it cannot be used to build efficient branch-and-price algorithms and will hence be omitted for further analysis. With respect to the dynamic programming procedures, DP1 outperforms the other algorithms and will consequently be used in the branch-and-price approaches of Section 5.3.2. It also allows

Table 3: Comparing the computational efficiency of the column generation optimization loop (seconds).

	DP1	DP2	DP3	DP4	DP5	IP
average	0.393	0.597	1.057	0.403	0.471	21.505
Q1	0.031	0.062	0.063	0.047	0.031	0.297
median	0.125	0.203	0.274	0.125	0.110	2.298
Q3	0.391	0.640	0.907	0.407	0.516	10.641
minimum	0.000	0.015	0.015	0.000	0.000	0.015
maximum	5.281	7.500	18.297	5.297	6.140	679.885

for an easy incorporation of the lower bound of Section 4.3.2 since this algorithmic approach returns for each surgeon the column with the most negative reduced cost. It should be noted that DP1 and IP represent the conventional way to price out columns and to structure the column generation optimization loop.

In about 17% of the instances, solving the linear relaxation of the SCSP results in the optimal solution for the SCSP, i.e. all column variables have an integer value. We refer to Table 2 to have some statistics on the absolute gap between the linear relaxation value of the SCSP and the optimal solutions.

5.3 Branch-and-price evaluation

5.3.1 Speed-up techniques

Before we discuss the diverse branch-and-price procedures in detail, we need to examine whether the speed-up techniques of Section 4.3 contribute to the solution quality. With respect to the initial solution heuristic, though, we still have to decide on its configuration. In Figure 4, 36 configurations are tested and evaluated. All configurations have in common that the set of columns, needed by the commercial IP solver to develop surgery schedules, is generated by building a limited tree based on a branch-and-price methodology. Further configuration, though, may differ on three levels. First, the depth of the tree is limited and varies from level 1 to level 6. In the former case, the tree is equal to the root node, whereas the tree maximally consists of 63 nodes in the latter case. Second, the branching strategy of the branch-and-price algorithm may be depth-first or best-first. Third, the allocation of the allowed computation time may differ. In

Figure 4, 15/45 denotes that maximally 15 seconds are devoted to the generation of columns, whereas minimally 45 seconds are allocated to generating surgery schedules by the commercial IP solver.

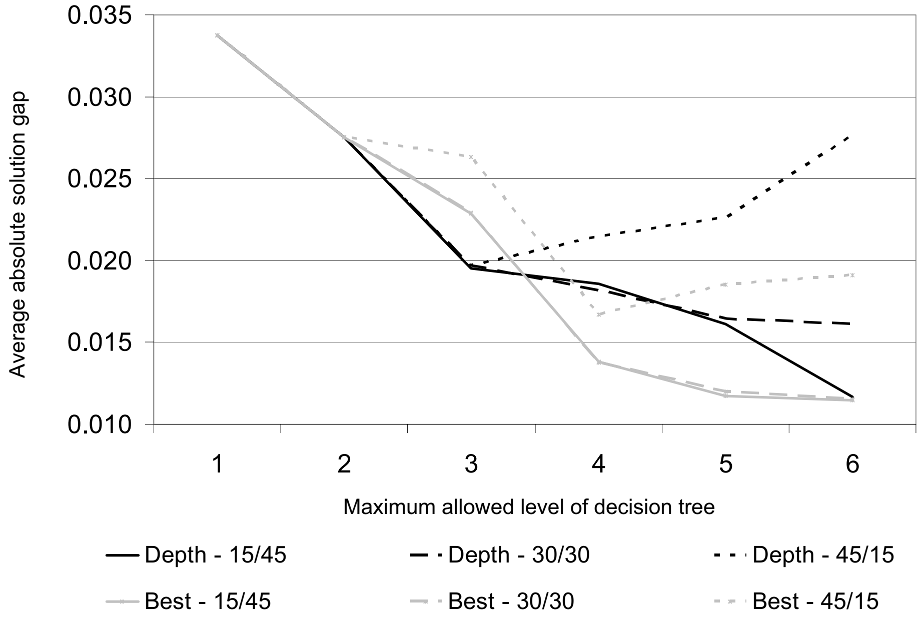


Figure 4: Visualizing the contribution of the diverse initial level-constrained heuristics on the average absolute solution gap.

When the branching tree does not exceed level 2, Figure 4 indicates that all configurations have an equal performance. Since the depth-first and best-first branching strategy only differ from level 2 on, and since the nodes could be explored in less than 15 seconds, this result could be expected. In general, increasing the depth of the tree results in an increased variety in columns and hence leads to smaller solution gaps. This trend, however, does not apply for the algorithms in which the time setting is equal to 45/15. These configurations are from a certain level on characterized by decreasing performance. When more columns are allowed, but schedule generation time is quite limited, the commercial IP solver encounters difficulties in finding good solutions. Within each time allocation pattern, the best-first approach tends to perform better than the depth-first approach when the tree has considerable depth. Moreover, along with the depth of the tree, it seems better to allocate time according to a 15/45 pattern. This implies that it is more important to save time for schedule generation than for column generation.

In the experiment, we only considered the first branching scheme (see Section 4.2), though we believe that the main conclusions will hold for the other branching schemes, i.e. better solutions can be obtained by increasing the number of columns on the one hand and saving enough computation time for the schedule generation phase on the other hand. The initial solution heuristics applied in the remainder of this paper will follow a 15/45 time allocation pattern and the depth of the initial column tree will be limited to level 6. The branching strategy will be in correspondence with the branching strategy of the main branch-and price procedure that will be tested in order to avoid miscellaneous effects.

In Table 4, the impact of an initial solution heuristic (a), the Lagrangian lower bound (b) and the elimination of columns (c) can be found for a depth-first branch-and-price algorithm that incorporates branching scheme 1. The results indicate that incorporating the lower bound only leads to minor improvements, whereas the initial solution heuristic has a major beneficial impact. The contribution of the column elimination feature is somehow moderate. Since all speed-up techniques seem to be profitable, we will integrate them all in the branch-and-price approaches with a depth-first branching strategy.

When we turn our interest to the best-first procedures, the same properties hold for the use of the initial heuristic and the lower bound calculation. In Section 4.3, though, we mentioned that eliminating columns in a best-first environment may end up in regenerating columns that were discarded from the column pool. One advantage of eliminating the set of columns from time to time is that this set remains small. This implies that the set is suited for generating surgery schedules using a commercial IP solver. In other words, without losing the exact nature of the branch-and-price algorithms, column elimination leads to the ability to intertwine the tree generation process with an easy heuristic and should hence improve the quality of the surgery schedules. Moreover, the best-first nature implies that nodes from different regions of the tree are explored and should increase the variety of columns in the set. In the next section, we will notice from Table 5 that the exact branch-and-price algorithms in which a depth-first branching strategy was applied generate on average less columns than the best-first procedures. This is a consequence of resetting the column pool of the best-first procedures after a 5-node exploration. The allowed computation time of the commercial IP solver, which immediately precedes the deletion of the columns, is limited to 15 seconds. The column pool will only be reset when at least one feasible solution is already encountered during the tree search.

Table 4: Evaluation of speed-up techniques with respect to the absolute solution gap (depth-first, branching scheme 1).

	average	standard deviation
/	0.02741	0.08950
(a)	0.00945	0.02323
(b)	0.02703	0.08935
(c)	0.02398	0.08915
(a)+(b)	0.00914	0.02289
(a)+(c)	0.00895	0.02272
(b)+(c)	0.02397	0.08894
(a)+(b)+(c)	0.00890	0.02272

5.3.2 Comparing the algorithmic procedures

The computational results of the branch-and-price algorithms of this paper are summarized in Table 5. Note that the heuristic algorithm *Best - Scheme 1 - MIP* reflects the generation of a set of columns (allowed level is 10, time allocation pattern equals 50/250) which is consecutively solved using the ILOG IP solver.

From the table, we can see that none of the branch-and-price algorithms succeeds in proving the optimality of the solution in at least 50% of the instances. This is a rather poor result which indicates that the applied branching schemes may not be very restrictive and may not result in a well-balanced tree. However, for about 70% of the instances, the branch-and price methodology leads to a solution value that equals the optimal solution (zero solution gap), which is a steady performance. Except for Depth - Scheme 4, at least one feasible surgery schedule can be found for each instance. This is a valuable result when it comes to the practical implementation of the algorithms.

The smallest average solution gap is encountered for the *Best - Scheme 1* algorithm and equals 0.007. This result equals the average solution gap of the iterated IP, which was the best performing algorithm that was introduced in Cardoen, Demeulemeester and Beliën (2006). This implies that a similar result can now be obtained through the application of an exact algorithm instead of a heuristic. Moreover, when we compare the standard deviation of the absolute solution gap, the branch-and-price approach tends to be more stable in delivering qualitative

Table 5: Computational results for the branch-and-price procedures.

		solution	time	abs gap solution	nr columns
Depth - Scheme 1	average	0.100	166.280	0.009	11616
49% opt - 0% no sol	median	0.079	300.000	0.000	9138
69% zero sol gap	st. dev.	0.086	143.399	0.023	12427
Depth - Scheme 2	average	0.100	171.655	0.010	12082
47% opt - 0% no sol	median	0.080	300.000	0.000	11008
67% zero sol gap	st. dev.	0.086	143.177	0.026	12736
Depth - Scheme 3	average	0.099	180.819	0.009	9183
49% opt - 0% no sol	median	0.080	300.000	0.000	6145
67% zero sol gap	st. dev.	0.085	215.145	0.023	10518
Depth - Scheme 4	average	0.109	176.676	0.019	19699
48% opt - 1% no sol	median	0.081	300.000	0.000	16886
65% zero sol gap	st. dev.	0.122	165.367	0.083	19919
Best - Scheme 1	average	0.098	171.377	0.007	22890
47% opt - 0% no sol	median	0.079	300.000	0.000	7887
71% zero sol gap	st. dev.	0.085	142.286	0.021	31834
Best - Scheme 2	average	0.098	176.690	0.008	24906
46% opt - 0% no sol	median	0.079	300.000	0.000	7925
69% zero sol gap	st. dev.	0.084	142.831	0.024	35158
Best - Scheme 3	average	0.098	176.177	0.008	20212
45% opt - 0% no sol	median	0.079	300.000	0.000	6555
69% zero sol gap	st. dev.	0.084	142.993	0.024	30946
Best - Scheme 4	average	0.098	178.403	0.008	24951
44% opt - 0% no sol	median	0.079	300.000	0.000	8101
69% zero sol gap	st. dev.	0.086	143.143	0.022	34707
Best - Scheme 1 - MIP	average	0.099	/	0.009	741
0% no sol	median	0.079	/	0.000	794
70% zero sol gap	st. dev.	0.085	/	0.026	640

schedules. Recall that the absolute solution gap is within the range $[0, 1]$, so that values varying from 0 to 0.010 are good. The development of an iterated branch-and-price approach, similar to the iterated IP, constitutes an area for future research.

We can summarize the computational results by stating that the branch-and-price procedures definitively guarantee a qualitative surgery schedule. The algorithms tend to have a comparable performance and steadily outperform most of the approaches introduced in Cardoen, Demeulemeester and Beliën (2006) when the goal is to minimize the absolute solution gap. With respect to the proof of optimality, however, the performance should still be upgraded. In a daily operational setting, though, priority is given to stable algorithms that result in qualitative, but not necessarily optimal, surgery schedules.

6 Conclusions and future research

In this paper, a surgical case scheduling problem was introduced and solved using a branch-and-price methodology. The problem was formulated using patterns or columns that represent groups of sequenced surgeries. A column generation optimization loop was specified in which the restricted master problem was solved to optimality by pricing out profitable columns. Two pricing algorithms were developed, yet only the dynamic programming procedure succeeded in generating favorable columns within a minimum of computation time. Since column generation cannot guarantee that variables have integer values, the column generation approach was intertwined with branching schemes and upgraded through speed-up techniques. This resulted in both exact and heuristic algorithms for which the performance was evaluated through a realistic test set. Although many satisfying results could be obtained, solving instances to optimality remains difficult for this NP-hard optimization problem. Furthermore, parameter settings should be closely examined and intelligent branching schemes should be introduced and further tested. Since the application of even a simple heuristic during the tree generation process seemed to be beneficial, we should further explore this opportunity and check whether, for instance, local branching (Fischetti and Lodi 2003), guided dives or relaxation induced neighborhood search (Chabrier et al. 2004) could be embedded in the branch-and-price setting of the SCSP.

The cooperation with the surgical day-care center of the university hospital Gasthuisberg in Leuven (Belgium) added a realistic and practical dimension to the problem. The development of effective algorithms obviously contributes to the progress towards a health care sector that is focused on quality and profitability. The applicability of the algorithms and thus the valorization of these practical scheduling instruments, though, could be significantly increased through the introduction of a graphical user interface (GUI) that assists the health manager in his or her scheduling process. The design of such a GUI for scheduling surgical cases in a day-care environment constitutes an area for future development.

Next to the sequencing of individual patients on a surgery day, attention should be paid to the assignment of patients to a particular surgery day. It should be clear that the quality of the sequencing step clearly depends on these assignment policies. Therefore, we want to enlarge the surgical case scheduling problem and try to develop simple online assignment policies that lead to a daily patient population for which a qualitative surgery schedule can be found.

Acknowledgements

We are grateful to Pierre Luysmans of the surgical day-care center at Gasthuisberg for introducing this practical scheduling problem and providing the data in order to build a structured test set. His suggestions and experience contributed to the practical value of this research. We acknowledge the support given to this project by the Bijzonder Onderzoeksfonds of the Katholieke Universiteit Leuven.

Appendix A

In this Appendix we will state the symbols used throughout the equations of the paper. We will distinguish between indices, sets, decision variables, help variables and data parameters.

Indices:

i:	surgery type	e:	instrument type	r:	operating room
n:	patient	j:	objective	b:	infection
p, p':	period	t:	column or pattern	s:	surgeon
h, h', $h^{\emptyset}$:	state	q, q', $q^{\emptyset}$:	sequenced set	g:	stage
o:	resource				

Note that a state h and a sequenced set q actually represent sets of surgery types and that $h^{\emptyset}$ and $q^{\emptyset}$ are equal to the empty set.

Sets:

I:	set of surgery types
I_s:	set of surgery types for surgeon s
I_r:	set of surgery types that can be performed in operating room r
I_e:	set of surgery types for which instrument e is needed
I_{bact}:	set of surgery types with bacterial infection ($b \neq 0$)
$I_{presurg}$:	set of surgery types with pre-surgical tests
R:	set of operating rooms
R_s:	set of operating rooms for surgeon s
S:	set of surgeons
E:	set of instrument types
T_s:	set of columns for surgeon s
J:	set of objectives: $\forall j \in J : worstvalue_j \neq bestvalue_j$
N:	set of patients to be scheduled
N_s:	set of patients to be scheduled for surgeon s

H_g:	set of states at stage g
Q_b:	set of sequenced surgery sets for bacteria b
B_s:	set of infections represented in patient population of surgeon s

Decision variables:

$$x_{ips} = \begin{cases} 1 & \text{if a surgery of type } i \text{ starts on period } p \text{ by surgeon } s \\ 0 & \text{otherwise} \end{cases}$$

$$z_{st} = \begin{cases} 1 & \text{if column } t \text{ is chosen for surgeon } s \\ 0 & \text{otherwise} \end{cases}$$

Help variables and functions:

α_1:	$\sum_{n \in N} \Theta^{child}_{type_n} \cdot \upsilon_n$ (=periods)
α_2:	$\sum_{n \in N} \Theta^{prior}_{type_n} \cdot \upsilon_n$ (=periods)
α_3:	$\sum_{n \in N: \upsilon_n < travelref} \Theta^{travel}_{type_n}$ (=patients)
α_4:	$\sum_{n \in N} max[0, \upsilon_n + k_{type_n} + l_{type_n} + m_{type_n} - 1 - P^{ub}]$ (=periods)
α_5:	peak number of beds used in recovery phase 1 (=beds)
α_6:	peak number of beds used in recovery phase 2 (=beds)
υ_n:	starting period of surgery of patient n
$\lambda_s, \rho_p, \sigma_p, \pi_{ep}$:	dual prices
RFI_j:	room for improvement of objective j
RC_{st}:	reduced cost of column t for surgeon s
RC^*_s:	most negative reduced cost for surgeon s
$\Phi(h,q)$:	value determining feasibility of adding sequence q to state h
$\Psi(h,q)$:	stage reached after addition sequence q to state h
$\beta_g(h)$:	infection introduced at stage g
$F_g(h)$:	minimum cost of surgery completion given that state h in stage g is realized
$C(h,h')$:	state transition cost function
$\|Set\|$:	number of elements in Set
$\|Set\|_{condition}$:	number of elements in Set for which *condition* is true

Data parameters:

$$\theta_i^{child} = \begin{cases} 1 & \text{if } i \in I_{child} \\ 0 & \text{otherwise} \end{cases} \qquad \theta_i^{prior} = \begin{cases} 1 & \text{if } i \in I_{prior} \\ 0 & \text{otherwise} \end{cases} \qquad \theta_i^{travel} = \begin{cases} 1 & \text{if } i \in I_{travel} \\ 0 & \text{otherwise} \end{cases}$$

P^{lb}:	opening period of the day-care center
P^{ub}:	closing period of the day-care center
P^{lb}_{rs}:	starting period for surgeon s in operating room r
P^{ub}_{rs}:	closing period for surgeon s in operating room r
k_i:	length of surgery of type i (periods)
l_i:	length of recovery phase 1 for surgery type i (periods)
m_i:	length of recovery phase 2 for surgery type i (periods)
$clean$:	cleaning type (= merger of k_{clean} idle types)
$ster_e$:	periods needed to sterilize instrument of type e
cap_l:	capacity of recovery phase 1 (patients)
cap_m:	capacity of recovery phase 2 (patients)
cap_e:	number of instruments of type e available
$idle$:	idle type (k_{idle}=1 period)
mrp:	latest period on which any patient can possibly be in recovery
$bestvalue_j$:	best possible value for α_j
$worstvalue_j$:	worst possible value for α_j
a_{opst}:	units of resource o needed in period p when column t is chosen for surgeon s
$type_n$:	surgery type of patient n
w_j:	weight of objective j
c_{st}:	$\sum_{j \in J: j \leq 4} (w_j \cdot \tilde{c}_{jst})/(worstvalue_j - bestvalue_j)$
$\tilde{c}_{1st}$:	$\sum_{n \in N_s} \Theta^{child}_{type_n} \cdot \upsilon_n$ when column t is chosen for surgeon s
$\tilde{c}_{2st}$:	$\sum_{n \in N_s} \Theta^{prior}_{type_n} \cdot \upsilon_n$ when column t is chosen for surgeon s
$\tilde{c}_{3st}$:	$\sum_{n \in N_s : \upsilon_n < travelref} \Theta^{travel}_{type_n}$ when column t is chosen for surgeon s
$\tilde{c}_{4st}$:	$\sum_{n \in N_s} max[0, \upsilon_n + k_{type_n} + l_{type_n} + m_{type_n} - 1 - P^{ub}]$ when t is chosen for s
$bact_i$:	infection for surgery type i, if $bact_i = 0$: no infection occurs
$presurgref$:	reference period for pre-surgical tests

Appendix B

In this Appendix we will elaborate on the dynamic pricing problem introduced in Section 3.1 in order to handle the occurrence of infections. The modified recursive dynamic programming formulation is stated as follows:

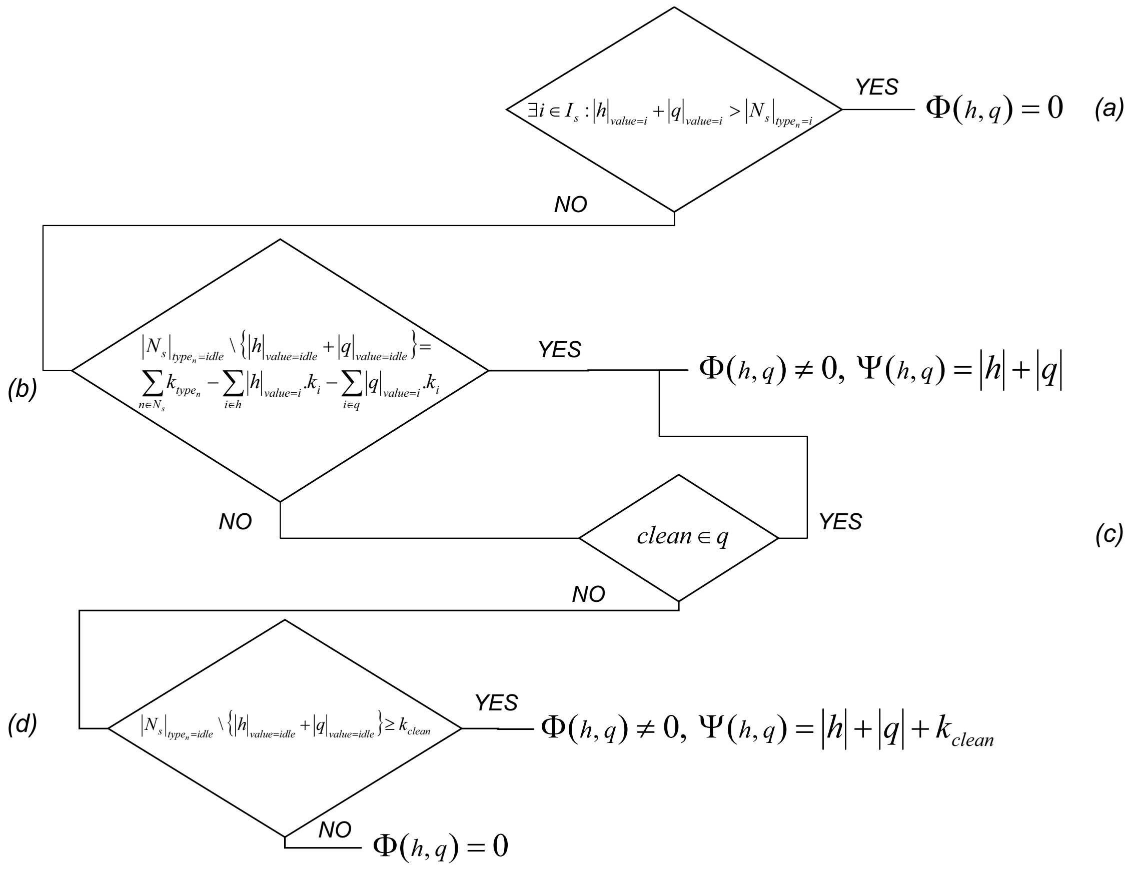
$$RC_s^* = \min_{h \in H_1} \left\{ \min_{q \in Q_{\beta_1(h)} : \Phi(h,q) \neq 0} \left\{ C(h^{\emptyset}, q^{\emptyset}, h, q) + F_{\Psi(h,q)}(h,q) \right\} \right\} - \lambda_s \tag{15}$$

$$F_{\Psi(h,q)}(h,q) = \min_{\substack{h' \in H_{\Psi(h,q)+1} : \\ (h \cup q) \subset h'}} \left\{ \min_{\substack{q' \in Q_{\beta_{\Psi(h,q)+1}(h')} : \\ \Phi(h',q') \neq 0}} \left\{ C(h,q,h',q') + F_{\Psi(h',q')}(h',q') \right\} \right\} \tag{16}$$

When the transition from state h at stage g to state h' at stage $g+1$ coincides with the decision to schedule the surgery of a patient with infection b, we will try to add an eligible and sequenced set of surgeries $q \in Q_b$ to the schedule so that the supplementary restrictions introduced by the infection cannot influence future decisions anymore. However, when such a transition denotes the scheduling of a regular patient (i.e. without infection so that $b = 0$), we will apply the logic of the original DP formulation of Equations 11 and 12. It will become clear that this is actually equivalent with the addition of the only sequenced set in Q_0, namely $q^{\emptyset}$. Note that, due to the addition of the sequenced sets, decisions possibly will not have to be made in every stage now.

Before we can initiate the recursive function, some calculations need to be done. First, we will try to aggregate idle periods into a cleaning type (*clean*) since this would reduce the number of surgeries to be scheduled. Such aggregation is only allowed, though, when no other surgery schedules can be developed in which the maximum number of successive idle periods is smaller than the duration of the cleaning type (k_{clean}). When, for instance, 4 surgeries need to be scheduled for surgeon s ($|N_s|_{type_n=1} = 1$, $|N_s|_{type_n=2} = 2$ and $|N_s|_{type_n=3} = 1$) and we assume that $I_{bact} = \{1\}$, the additional cleaning time equals 2 periods and the surgeries of type 2 represent idle periods, we can only merge the idle periods into a cleaning type when surgeon s is not the last surgeon who will perform surgeries in the specific operating room. Second, for each infection b, represented in the patient population of surgeon s, we have to enumerate the sequenced sets of surgeries $q \in Q_b$. Except for $q^{\emptyset} \in Q_b : 0 \leq b \leq |B_s|$, each sequenced set q

consists of a combination of idle periods, infected surgery types and cleaning types. Although we will not discuss the enumeration algorithm in detail, it is important to know that these sequenced sets either end on an infected surgery type or on a cleaning type. Moreover, multiple types of infections do not occur in a particular sequenced set. Formulating the DP using the sequenced sets implies that the occurrence of infections is mainly handled by choosing eligible paths through the stages, instead of assigning costs for infeasibilities.



$\exists i \in I_s : |h|_{value=i} + |q|_{value=i} > |N_s|_{type_n=i}$
YES
$\Phi(h,q)=0$
(a)
NO
$|N_s|_{type_n=idle} \setminus \{|h|_{value=idle} + |q|_{value=idle}\} = \sum_{n\in N_s} k_{type_n} - \sum_{i\in h} |h|_{value=i}.k_i - \sum_{i\in q} |q|_{value=i}.k_i$
(b)
YES
$\Phi(h,q)\neq 0,\ \Psi(h,q)=|h|+|q|$
NO
$clean \in q$
YES
(c)
NO
$|N_s|_{type_n=idle} \setminus \{|h|_{value=idle} + |q|_{value=idle}\} \geq k_{clean}$
(d)
YES
$\Phi(h,q)\neq 0,\ \Psi(h,q)=|h|+|q|+k_{clean}$
NO
$\Phi(h,q)=0$

Figure 5: Decision scheme for determining value of $\Phi(h, q)$ and $\Psi(h, q)$.

When state h is realized by scheduling a surgery type for which $b \neq 0$, we can use the decision scheme depicted in Figure 5 in order to determine whether the choice of a sequenced set q after realization of state h would lead to a feasible path ($\Phi(h, q) \neq 0$). Otherwise, i.e. when a surgery is scheduled for which b equals 0, we add the empty sequenced set $q^{\emptyset}$, set $\Phi(h, q) = 0$ and take the next decision at stage $\Psi(h, q) + 1 = |h| + 1$. From Figure 5 (a), we can see that it is not allowed to schedule a sequenced set q when this would lead to a schedule in which more patients of a certain surgery type are scheduled than represented in the patient population of the surgeon. Next, three situations can occur so that the infection is neutralized. First, it is possible that no more surgeries have to be scheduled or that the remaining surgery types, i.e. those that still have to be scheduled, only consist of idle types (Figure 5 b). This implies that we could reach the end of the surgery session for that surgeon. Second, it might be possible that the sequenced set q ends on a cleaning type (Figure 5 c). Finally, when $clean \notin I$, we allow the consequences of the infection to be fully neutralized by idle periods, though now without reaching the end of the session (Figure 5 d). When none of the above situations apply, the sequenced set q cannot be added to the surgery schedule and a next sequenced set should be consulted.

The cost function $C(h, q, h', q')$ is constructed in a similar way as in the original DP formulation of Section 3.1, except that we possibly have to incorporate objective costs, duality costs and infeasibility costs of multiple surgeries. The number of surgeries introduced to the schedule during a stage transition is equal to $\Psi(h', q')$-$\Psi(h, q)$. Since q' is sequenced, we can easily associate a start time with each surgery. With respect to the infeasibility costs, we want to stress that $C(h, q, h', q')$ could be equal to ∞ not only due to pre-surgical tests or a switch in operating rooms when a surgery is performed, but also due to infections when the additional cleaning of the operating room affects the surgical block of the subsequent surgeon, given that there is a surgeon switch in the operating room.

References

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

Beliën, J. 2006. *Exact and Heuristic Methodologies for Scheduling in Hospitals: Problems, Formulations and Algorithms*, Ph.D. dissertation, Katholieke Universiteit Leuven, Belgium.

Beliën, J. and Demeulemeester, E. 2007. Building cyclic master surgery schedules with leveled resulting bed occupancy, *European Journal of Operational Research* **176**: 1185–1204.

Beliën, J., Demeulemeester, E. and Cardoen, B. 2006. Visualizing the demand for various resources as a function of the master surgery schedule: A case study, *Journal of Medical Systems* **30**: 343–350.

Bellman, R. 1957. *Dynamic Programming*, Princeton University Press, Princeton, NJ.

Blake, J. T. and Carter, M. W. 2002. A goal programming approach to strategic resource allocation in acute care hospitals, *European Journal of Operational Research* **140**: 541–561.

Blake, J. T., Dexter, F. and Donald, J. 2002. Operating room manager's use of integer programming for assigning block time to surgical groups: A case study, *Anesthesia and Analgesia* **94**: 143–148.

Blake, J. T. and Donald, J. 2002. Mount Sinai hospital uses integer programming to allocate operating room time, *Interfaces* **32**: 63–73.

Cardoen, B., Demeulemeester, E. and Beliën, J. 2006. Optimizing a multiple objective surgical case scheduling problem, *Research Report KBI_0625*, Katholieke Universiteit Leuven, Department of Decision Sciences and Information Management.

Carter, M. 2002. Health care: Mismanagement of resources, *ORMS Today* **19**: 26–32.

Chabrier, A., Danna, E., Le Pape, C. and Perron, L. 2004. Solving a network design problem, *Annals of Operations Research* **130**: 217–239.

De Lathouwer, C. and Poullier, J. 2000. How much ambulatory surgery in the world in 1996-1997 and trends, *Ambulatory Surgery* **8**: 191–210.

Demeulemeester, E. and Herroelen, W. S. 2002. *Project scheduling - A research handbook*, Kluwer Academic Publishers, Boston, MA.

Desaulniers, G., Desrosiers, J. and Solomon, M. 2005. *Column generation*, Springer, New York.

Fischetti, M. and Lodi, A. 2003. Local branching, *Mathematical Programming* **98**: 23–47.

Guinet, A. and Chaabane, S. 2003. Operating theatre planning, *International Journal of Production Economics* **85**: 69–81.

Hamilton, D. M. and Breslawski, S. 1994. Operating room scheduling: Factors to consider, *Association of Operating Room Nurses Journal* **59**: 665–680.

Hans, E. W. 2001. *Resource loading by branch-and-price techniques*, Ph.D. dissertation, Twente University Press, Enschede, The Netherlands.

Hsu, V., de Matta, R. and Lee, C.-Y. 2003. Scheduling patients in an ambulatory surgical center, *Naval Research Logistics* **50**: 218–238.

ILOG 2002. *ILOG CPLEX 8.1 User's Manual.*

Jebali, A., Alouane, A. B. H. and Ladet, P. 2006. Operating rooms scheduling, *International Journal of Production Economics* **99**: 52–62.

Marcon, E., Kharraja, S. and Simonnet, G. 2003. The operating theatre planning by the follow-up of the risk of no realization, *International Journal of Production Economics* **85**: 83–90.

Michie, D. 1968. Memo functions and machine learning, *Nature* **218**: 19–22.

Sier, D., Tobin, P. and McGurk, C. 1997. Scheduling surgical procedures, *Journal of the Operational Research Society* **48**: 884–891.

Skiena, S. 1998. *The Algorithm Design Manual*, Springer-Veralg, New York.

Standish, T. 1994. *Data Structures, Algorithms & Software Principles in C*, Addison Wesley.

Vanderbeck, F. and Wolsey, L. 1996. An exact algorithm for ip column generation, *Operations Research Letters* **19**: 151–159.