European Journal of Operational Research 226 (2013) 551–559

Contents lists available at SciVerse ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

The implementor/adversary algorithm for the cyclic and robust scheduling problem in health-care

Matias Holte [a], Carlo Mannino [b,*]

[a] SINTEF ICT, Dept. of Applied Mathematics, Norway
[b] University of Rome La Sapienza, Dept. of Computer and System Sciences, Italy

ARTICLE INFO

Article history:
Received 26 October 2011
Accepted 7 October 2012
Available online 20 November 2012

Keywords:
Health-care optimization
Master surgery scheduling
Robust optimization
Mixed-integer programming

ABSTRACT

A general problem in health-care consists in allocating some scarce medical resource, such as operating rooms or medical staff, to medical specialties in order to keep the queue of patients as short as possible. A major difficulty stems from the fact that such an allocation must be established several months in advance, and the exact number of patients for each specialty is an uncertain parameter. Another problem arises for cyclic schedules, where the allocation is defined over a short period, e.g. a week, and then repeated during the time horizon. However, the demand typically varies from week to week: even if we know in advance the exact demand for each week, the weekly schedule cannot be adapted accordingly. We model both the uncertain and the cyclic allocation problem as adjustable robust scheduling problems. We develop a row and column generation algorithm to solve this problem and show that it corresponds to the implementor/adversary algorithm for robust optimization recently introduced by Bienstock for portfolio selection. We apply our general model to compute master surgery schedules for a real-life instance from a large hospital in Oslo.

1. Introduction

Mathematical optimization is playing an increasingly relevant role in health-care management [21]. Indeed, as observed in [4], “in the near future public resources for public health will become inadequate. Therefore, we need effective ways for planning, prioritization and decision making”. The major task of hospital administrations is that of efficiently allocating a number of medical resources for different purposes. A large variety of assignment and scheduling problems arise: we refer the reader to [10,21] for several examples and approaches.

The allocation of resources to different medical specialties is directly connected to the admission planning problem, which amounts to establishing the sequence of accepted patients. Typically, patients requiring some specific therapy are first placed in a waiting list and thereafter admitted to the hospital. A standard performance indicator for hospital efficiency is related to the length of such lists. Shorter lists are obviously preferred, but it is in general impossible to avoid a certain amount of queue, and maybe not even desirable. Indeed, the absence of queue for some specialties could actually reveal an inefficient allocation of some scarce resource.

It follows that the demand for medical care is met not only by scheduling the necessary resources but also by allowing a certain amount of queue. Albeit a short waiting list may be acceptable, long queues are to be avoided, as they represent a huge cost for health-care systems [13,20]. For each specialty, the cost of its queue is a design parameter which must be established by the hospital board. In the typical case, the cost will be represented by a convex function, with marginal costs increasing with the queue length.

So, a basic scheduling problem in health-care may be stated as the problem of allocating a number of resources to medical specialties so as to minimize the queue costs.

Clearly the allocation process must be undertaken long in advance and may involve some negotiation. So, resources are allocated at the beginning of a time horizon which may be quite long – from months to years. The number of patients for each specialty is thus estimated in advance and the actual number may differ substantially from the initial guess. In addition, schedules are often created with reference to the *planning horizon* (e.g. from one to four weeks) and then cyclically repeated along the much longer time horizon (*cyclic schedule*): obviously the actual demand may vary from period to period, even if deterministically known in advance. A safe (cyclic) schedule should then ensure that queues are as small as possible when the demand is the worst possible (with respect to the chosen schedule).

* Corresponding author. Tel.: +47 41588551.
E-mail addresses: kjetimh@student.matnat.uio.no (M. Holte), mannino@dis.uniroma1.it (C. Mannino).

0377-2217/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2012.10.029

This kind of problems is tackled by *robust optimization* (see [6,15]). In the robust optimization paradigm, the parameters of the problem (i.e. the coefficients of the constraint matrix and the right hand sides) are not completely specified and they vary in the so called *uncertainty set*. One wants to find a value for (all of) the decision variables which is feasible for any possible realization (or *scenario*) of the parameters in the uncertainty set and is optimum according to some given criterium. A slightly different version of the robust optimization paradigm with major practical and theoretical consequences [6] is the *adjustable robust counterpart* of an optimization problem. This variation was introduced to take into account the presence in the model of *adjustable* variables, i.e. "variables which do not correspond to actual decisions and can tune themselves to varying data" [7]. In the literature this class of robust problems is also called *two stage robust problems* (see, e.g., [2]).

Different robustness optimality criteria can be adopted, according to the specific nature of the application (see [15]). Among the most adopted ones are the following two criteria:

- *Absolute Robustness* in which one wants to find the solution with minimum cost w.r.t. its worst scenario;
- *Robust Deviation*, also called *min–max regret* [3]. The *regret* for a given solution (say x) is defined as the difference between the cost of x, obtained with its worst input instance, say y, and the cost of the best possible solution when the input is y (a more formal definition will be given in Section 5): one wants to find a solution which ensures the minimum possible regret.

The absolute robustness criterium is clearly most conservative, and is typically adopted in situations where it is crucial to hedge against the worst possible happenings. This is the case when handling patients asking for medical care, and thus we have adopted this approach here. However, the regret is still an interesting measure of the quality of the robust solution and we have considered it in some of our assessing experiments (see Section 5).

We assume here that the demand for medical care in the different specialties is an uncertain parameter belonging to an uncertainty set Y. Also, in the optimization model we have variables associated with the schedule and variables associated with the queues. While the schedule variables represent "here and now" decisions, the queue variables "adapt" to meet the actual demand. In other words, the feasible schedules are those for which, for any possible demand realization, *there exists* a queue value so that the demand is met. We make explicit use of this definition in devising the row and column generation algorithm (*dynamic simplex algorithm*, see [1]) for the robust scheduling problem described in Section 4. The overall approach can be viewed as an implementation of the *implementor-adversary* algorithm for robust optimization introduced by Bienstock in [9]. In this paper, Bienstock also claims to expect that similar ideas would prove useful for other robust optimization models. With this work we confirm his forecast.

We apply our technique to instances of the Master Surgery Scheduling (MSS) Problem, a much addressed and studied problem in the class of scheduling problems in health-care. The literature on the deterministic version of this problem is quite huge; a comprehensive survey on the topic can be found in [10]. In contrast, few papers deal explicitly with uncertainty. In [5,13,19], the authors apply mathematical programming and local search techniques to cope with stochastic durations of surgical procedures. Two recent papers deal with uncertain demand, but with different assumptions and, consequently, different solution approaches. In [5], a probability (multinomial) distribution of the demand is given and the corresponding stochastic problem is modelled as a mixed integer linear program. In contrast, in [16] no probability distribution on the demand is known, which instead is assumed to vary in a given interval, and a light robustness model [12] is constructed to find suitable schedules.

In this paper we give several contributions to these research lines. First, we introduce a unifying model for cyclic scheduling and robust scheduling in health-care. Second, we devise a row and column generation approach to solve this model. Third, we show how this approach can be viewed as an implementation of the implementor/adversary algorithm introduced in [9]. Finally, we successfully apply our technique to solve real-life instances provided by SAB (*Sykehuset Asker og Bærum HF*), a major hospital in the city of Oslo.

2. The master scheduling problem

In this section we formalize the general scheduling problem with queue cost minimization in health-care. Let G be the set of specialties, such as orthopedics, gynecology, etc. We denote by D the set of days in our planning horizon, while R is the set of available medical resources, e.g. rooms or surgical teams, or even feasible combinations of medical resources. We assume here that medical resources can only be assigned by an integer number of time slots[1]. Depending on his/her specific treatment, each patient requires a set of contiguous time-slots of the medical resource, called *block*. In the deterministic model, the overall block demand in the time horizon, for each specialty $g \in G$ and each possible block length $l \in L$, is a non-negative integer b_{gl}. Observe that $\sum_l l \cdot b_{gl}$ is the total amount of slots required for specialty g during the time horizon. The number of time slots available in each day may vary from day to day. To simplify the notation let us introduce the set of *superslots* S, namely the set of feasible unordered triples $\{r,s,d\}$, where r is a medical resource, s a time-slot and d a day. That is, superslot $\{r,s,d\} \in S$ if resource r is available at slot s in day d. In this setting, a (*super*) block is simply a subset $B \subseteq S$ such that the superslots in B correspond to the same medical resource, to the same day and to one or more contiguous time-slots. Finally, let q_{gl} denote, for each specialty g and each block length l, the queue length measured as the number of unscheduled patients. A major task of the hospital board is to be able to keep short queues. Long queues have a social cost as wait times can impact health outcomes. We model such cost as a function c decomposable in the different specialties and block lengths, i.e. $c(q) = \sum_{g,l} c_{gl}(q_{gl})$, where c_{gl} is a non-decreasing, convex and piece-wise linear function. So the basic *Master Scheduling Problem* amounts to finding a feasible assignment of superslots to each specialty (*Master Schedule* or MS), so as to minimize the queue cost $c(q)$.

Let G, $P \subset N$. In the remainder of the paper, for any vector $x \in Z_+^{G\times P}$, we let $x_g = (x_{g1}, \ldots, x_{g|P|})^T$ for all $g \in G$ (and then $x = \left(x_1^T, \ldots, x_{|G|}^T\right)^T$).

Cyclic Schedules. The MS prepared by the hospital planning department is typically active for a long period, normally from one to a few years. Such long duration is mainly motivated by the fact that finding a MS is a difficult task, which involves specialized personnel for several working days. In addition, the hospital staff is affected by such timetable and the employees can put up a significant resistance to changes to the actual configuration.

In contrast, the actual planning horizon is much shorter, typically from one to eight weeks. The MS created for the planning horizon is then used as a pattern and repeated for the whole period of validity. So, the full MS which may hold for years is actually obtained by taking M copies of a shorter MS which refers to a few weeks. Again the reason to this is manyfold, but one major fact is

[1] At SAB the time slot is typically two hours.

that people prefer to have repeated working patterns along the year(s). Despite of the fact that the MS stays unchanged from period to period, the projected demand in each period may vary. Consequently, queue lengths and queue costs will also vary from period to period. Let us denote by $b^i \in Z_+^{G\times L}$ the (projected) demand in period i, for $i = 1, \ldots, M$, and let $Y = \{b^1, \ldots, b^M\}$ be the set of all demand vectors. An interesting goal is now that of finding a MS which is valid for all periods, and such that the largest queue cost is minimized. That is, denoting by $q^i \in Z_+^{G\times L}$ the queue vector in period $i = 1, \ldots, M$, the *cyclic* MS problem (c-MS) is the problem of finding a MS such that the quantity $\max_i c(q^i)$ is minimized.

Robust Schedules. The queue length depends both on the MS and on the demand b, which in turn is typically estimated from historical records. However, such estimate is inherently uncertain, and we assume that, for each $g \in G$ and $l \in L$, the quantity b_{gl} may vary between two possible values, namely

$$\beta_{gl} \leqslant b_{gl} \leqslant \gamma_{gl} \tag{1}$$

As observed in the seminal paper by Bertsimas and Sim [8], in most realistic situations it is very unlikely that all uncertain parameters assume their upper bounds simultaneously. In [8] this observation is exploited by assuming that only a given number of the uncertain coefficients may deviate simultaneously from the reference value. This restriction does not look appropriate in our case. Instead we assume here that, even if in principle all demands b_{gl} may vary within the corresponding feasible interval, still the overall number of slots required by all patients in the different specialties does not exceed a threshold K, i.e.

$$\sum_g \sum_l l \cdot b_{gl} \leqslant K \tag{2}$$

Again, we denote by $Y = \{b^1, \ldots, b^M\}$ the (finite) set of feasible demand vectors, namely $Y = \{b \in Z^{G\times L}: b \text{ satisfies (1) and (2)}\}$.

Now, as in the c-MS problem, we want to find a MS which minimizes the largest queue cost. If we denote by $q^i \in Z_+^{G\times L}$ the queue vector with respect to $b^i \in Y$, then the adjustable robust MS problem (*aR-MS problem*) amounts to finding a MS such that the quantity $\max_i c(q^i)$ is minimized. It should be apparent that the c-MS problem and the aR-MS problem only differ in the way the set Y is defined, and from the next section we unify the discussion. By suitably defining the set Y, we can solve the actual and more general problem which is both cyclic and stochastic and thus the solution provided will be a *robust cyclic schedule*. The following discussion and algorithms apply to the general case.

3. An adjustable-robust model for the MS problem

There are several alternative ways to represent the MS problem by mathematical optimization models, and in particular by mixed-integer linear programs (MILPs) (see [4] for various examples). Here we adopt the so called *pattern formulation* introduced in [16] for the master surgery scheduling problem. Such formulation is based on the concept of *l-pattern*, which is simply a set of l contiguous superslots corresponding to the same day and the same medical resource. For example, if the slot length is $2h$ and a working day is $8h$, then we have 4 time-slots. Correspondingly, we have four different 1-patterns (patterns of one slot or 2 h), three distinct 2-patterns (4 h), 2 distinct 3-patterns (6 h) and one 4-pattern (8 h).

When a specialty is assigned a block of length l, it is actually assigned a specific l-pattern. Observe that each pattern corresponds to a precise time interval (i.e. from 10:00 to 16:00). Each specialty $g \in G$ "asks" for a number b_{gl} of patterns with block length l, for each possible block length l. Let us denote by P the set of all possible patterns, and let $E = \{\{u, v\}: u, v \in P \text{ and } u \cap v \neq \emptyset\}$, i.e. E is the set of non-disjoint pair of patterns. In other words, $\{u, v\} \in E$ if and only if pattern u and pattern v (which are both collections of time contiguous superslots) share a common superslot. Thus, two patterns intersect if they correspond to the same medical resource and they overlap in time. For all $g \in G$ and $p \in P$ we introduce a binary variable x_{gp} which is 1 if and only if pattern p is assigned to specialty g. To simplify the notation in the basic model we assume that every pattern $p \in P$ can be assigned to any specialty $g \in G$. It is not difficult to extend the model to cope with infeasible assignments by fixing suitable variables to 0. This may be the case, e.g., when a specialty needs some specific equipments not available in all operation rooms.

Let $p,q \in P$ be two (not necessarily distinct) patterns. If p and q intersect (i.e. $\{p,q\} \in E$) then they cannot be assigned simultaneously. This is represented by the following *packing constraint*:

$$x_{fp} + x_{gq} \leqslant 1, \quad g,f \in G, \{p,q\} \in E, \quad g \neq f \bigvee p \neq q \tag{3}$$

Similar packing constraints may be included to represent other incompatible assignments. For instance, if only one surgery team is available for specialty g, then we have $x_{gp} + x_{gq} \leqslant 1$, for all $p,q \in E, p \neq q$. A vector $x \in \{0,1\}^{G\times P}$ satisfying all above inequalities is a *Master Scheduling* (MS) (or *Master Schedule*). The set of all feasible MS is denoted by X.

Actually, as shown in [16], the above constraints can be strengthened by considering any set C of mutually intersecting patterns. Then we can replace (3) with the following family of inequalities:

$$\sum_{g\in G} \sum_{p\in C} x_{gp} \leqslant 1, \qquad C \in \mathcal{C} \tag{4}$$

where $\mathcal{C}$ is the family of the maximal sets of mutually intersecting patterns. Such a family can be identified efficiently [16].

Now, let $P(l)$ be the set of patterns of length l. The quantity $\sum_{p\in P(l)} x_{gp}$ is the number of blocks of length l assigned to specialty g: in principle this quantity may exceed b_{gl}. Then the queue q_{gl} of patients of group g with block length requirement l is defined as:

$$q_{gl} = \max\left(0, b_{gl} - \sum_{p\in P(l)} x_{gp}\right) \quad g \in G, l \in L \tag{5}$$

Thus, the queue $q \in Z_+^{G\times L}$ is a non-negative function $q(x,b)$ of the MS x and of the demand b.

The convex piece-wise linear cost associated with q can be easily transformed into a linear one by introducing suitable variables and constraints [16].

For sake of simplicity we discuss here the simpler case of a linear cost function. In particular, we consider the cost $c(q)$ of the queue q to be proportional to the queue total length, i.e.

$$c(q) = \sum_{g\in G} \sum_{l\in L} c_{gl} \cdot q_{gl} \tag{6}$$

where c_{gl} is a suitable, strictly positive constant (see Section 5 for the exact definition of the cost vector c in our test cases). The extension to the more general convex, piece-wise linear case is straightforward.

We linearize (5) by associating a non-negative real variable s_{gl} with q_{gl}, for $g \in G$, $l \in L$; then the (deterministic) MS problem can be formulated as following Mixed Integer Linear Program (MILP):

$$\begin{aligned} \min \quad & \xi \\ \text{s.t.} \quad & \text{(i)} \quad \sum_{g\in G} \sum_{p\in C} x_{gp} \leqslant 1, \quad C \in \mathcal{C} \\ & \text{(ii)} \quad s_{gl} + \sum_{p\in P(l)} x_{gp} \geqslant b_{gl}, \quad g \in G, l \in L \\ & \text{(iii)} \quad \xi \geqslant \sum_{g\in G} \sum_{l\in L} c_{gl} s_{gl} \\ & \xi \in \mathbb{R}, s \in \mathbb{R}_+^{G\times L}, \quad x \in \{0,1\}^{G\times P} \end{aligned} \tag{7}$$

Constraint (7ii) along with the non-negativity of s ensure that, for any x and b, we have $s \geqslant q(x,b)$. Moreover, thanks to (7iii) and the structure of the cost coefficients which are strictly positive, we trivially have $s^* = q(x^*,b)$ for every optimal solution (x^*,s^*) to (7).

Cyclic and Robust Schedules. We assume now that the demand b can actually range in the set Y previously defined, that we rewrite for the reader:

$$Y := \left\{ b \in Z^{G\times L} : \beta \leqslant b \leqslant \gamma, \sum_g \sum_l l \cdot b_{gl} \leqslant K \right\} = \{b^1, \ldots, b^{|M|}\} \tag{8}$$

where $\beta, \gamma \in Z_+^{G\times L}$. Remark that Y is a finite set with $|M|$ elements, where $M = \{1, \ldots, |M|\}$, and $|M|$ can grow exponentially in G and L. Then recalling the aR-MS problem, which consists in finding the MS x which minimizes the cost of the queue corresponding to the worst possible demand in the uncertainty set Y, can be stated as the following min–max problem:

$$\min_{x\in X} \max_{b\in Y} c(q(x,b)). \tag{9}$$

Following Bienstock [9] we can interpret this problem as a particular *implementor/adversary* game, where the implementor establishes a MS $\bar{x} \in X$ trying to minimize the queue cost while the adversary fixes a demand $b \in Y$, in turn *maximizing* the queue cost w.r.t. the schedule $\bar{x}$. We come back on this interpretation later on.

For every $x \in X$, let us denote by $q^i = q(x,b^i)$ the queue when the demand vector is b^i, for $b^i \in Y$. As for the deterministic case, we associate a non-negative real variable s^i_{gl} with q^i_{gl}, for every $g \in G$, $l \in L$ and $i \in M$. Then the aR-MS problem can be immediately represented by the following MILP:

$$\begin{aligned}
\min\quad & \xi \\
\text{s.t.}\quad & \text{(i)} \quad \sum_{g\in G}\sum_{p\in C} x_{gp} \leqslant 1, \quad C \in \mathcal{C} \\
& \text{(ii)} \quad s^i_{gl} + \sum_{p\in P(l)} x_{gp} \geqslant b^i_{gl}, \quad g \in G, l \in L, i \in M \\
& \text{(iii)} \quad \xi \geqslant \sum_{g\in G}\sum_{l\in L} c_{gl} s^i_{gl}, \quad i \in M \\
& \text{(iv)} \quad \xi \in \mathbb{R}, s \in \mathbb{R}_+^{G\times L\times M}, \quad x \in \{0,1\}^{G\times P}
\end{aligned} \tag{10}$$

In fact, constraints (10ii) and non-negativity ensure that $s^i \geqslant q^i$, for $i = 1, \ldots, |M|$, whereas constraints (10iii) ensure that $\xi \geqslant \max_{i\in M} c^T s^i \geqslant \max_{i\in M} c(q^i)$ with equality holding for the optimal solution. So, the original min–max problem (9) has been reduced to a standard MILP. A major difficulty with the above formulation is the large number (possibly exponential in $|G| \cdot |L|$) of inequalities (10ii) and (10iii) and variables s^i_{gl}. A classical technique to cope with it is to initially consider a small subset of such inequalities and variables and generate new ones only if necessary. This is the topic of the next section.

4. The implementor/adversary algorithm for the adjustable-robust MS Problem

We begin this section by recalling the classical delayed row generation approach (see, e.g., [1,18]).

Consider the linear program $(P) : \min\{c^t x : Ax \leqslant b, x \in \mathbb{R}^n\}$, where $A \in \mathbb{R}^{m,n}$, $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. To simplify the following discussion, we assume that (P) has a (finite) optimal solution x^*. In the sequel, we denote by A_i the ith row of A (written as a row vector). When A contains a large number of rows, or it is not given explicitly, it may be convenient to apply the so called *dynamic simplex algorithm* [1], which resolves (P) by solving a (hopefully short) sequence of smaller problems, obtained from (P) by dropping original constraints. More precisely, it starts by considering a subset of original constraints, solves the corresponding problem and, if necessary, adds one of the missing constraints and iterates. In order to choose the constraints to be included, the method exploits an algorithm called *separation oracle*. Given $\bar{x} \in \mathbb{R}^n$, the separation oracle either establishes that $A\bar{x} \leqslant b$, or returns a row A_j of A such that $A_j\bar{x} > b_j$. In other words, when $\bar{x} \in \mathbb{R}^n$ is not feasible for (P), the separation oracle returns one of the linear inequalities defining the feasible region of (P) which is violated by $\bar{x}$. More formally, the dynamic simplex algorithm proceeds as follows. Let $M = \{1, \ldots, m\}$ be the set of row indices, and let $M^0 \subseteq M$ be an initial subset of indices. Once again for sake of simplicity, we assume that the problem $(P^0) : \min\{c^t x : A_i x \leqslant b_i (i \in M^0), x \in \mathbb{R}^n\}$ has a (finite) optimal solution x^0. At the qth iteration of the method, we solve the problem $(P^q) : \min\{c^t x : A_i x \leqslant b_i (i \in M^q), x \in \mathbb{R}^n\}$, with $M^0 \subseteq M^q \subseteq M$. Since both (P^0) and (P) have optimal solutions, so does (P^q). Let x^q be the optimal solution to (P^q). Apply the separation oracle to x^q. If x^q is feasible for (P), then x^q is also optimal for (P) (as (P^q) is a relaxation of (P)) and we have solved the original problem. Otherwise, the oracle returns a row A_j of A such that $A_j x^q > b_j$. We let $M^{q+1} = M^q \cup \{j\}$ and iterate. It is not difficult that the method terminates after at most m iterations. Also, since for all $q > 0$, (P^{q-1}) is a relaxation of (P^q), we have $c^T x^{q-1} \leqslant c^T x^q \leqslant c^T x^*$. Observe that the iterative row generation can be carried out even when x is restricted to a subset of $\mathbb{R}^n$, e.g. the set of binary vectors in $\mathbb{R}^n$. Finally, the dual version of row generation, called *column generation* is also much applied to solve complex linear and combinatorial optimization problems (see, e.g., [11]).

We adapt the above approach to solve the MILP (10). Namely, we start by considering a subset $\widetilde{Y} \subseteq Y$ of possible demand vectors, corresponding to the set of indices $\widetilde{M} \subseteq M$. We then consider the *reduced master problem*:

$$\begin{aligned}
\min\quad & \xi \\
\text{s.t.}\quad & (i) \quad \sum_{g\in G}\sum_{p\in C} x_{gp} \leqslant 1, \quad C \in \mathcal{C} \\
& (ii) \quad s^i_{gl} + \sum_{p\in P(l)} x_{gp} \geqslant b^i_{gl}, \quad g \in G, \quad l \in L, \quad i \in \widetilde{M} \\
& (iii) \quad \xi \geqslant \sum_{g\in G}\sum_{l\in L} c_{gl} s^i_{gl}, \quad i \in \widetilde{M} \\
& (iv) \quad \xi \in \mathbb{R}, s \in \mathbb{R}_+^{G\times L\times \widetilde{M}}, \quad x \in \{0,1\}^{G\times P}
\end{aligned} \tag{11}$$

The above problem is obtained from problem (10) by dropping the constraints (10ii) and (10iii) for $i \in M \setminus \widetilde{M}$ and thus it provides a relaxation which is typically easier to solve in practice.[2] More in general, let $\widetilde{Y} \subseteq \overline{Y} \subseteq Y$, and let $\tilde{\xi}$ and $\bar{\xi}$ be the optimal values to the master programs associated with $\widetilde{Y}$ and $\overline{Y}$, respectively. Then, we have $\tilde{\xi} \leqslant \bar{\xi}$. Also, let $\tilde{x}$ be the optimal solution to the master problem associated with $\widetilde{Y}$ and let $\tilde{\xi}$ be the corresponding objective function value. Then for any $b^i \in \widetilde{Y}$, we have that the queue cost satisfies $c(q(\tilde{x}, b^i)) \leqslant \tilde{\xi}$. On the other hand, for $b^j \in Y \setminus \widetilde{Y}$ we may have $c(q(\tilde{x}, b^j)) > \tilde{\xi}$. However, if this is not the case, then $\tilde{x}$ is optimal for the original problem (10).

The above discussion can be summarized by following proposition:

Proposition 4.1. *Let $(\tilde{x}, \tilde{s}, \tilde{\xi})$ be the optimal solution to the current master problem (associated with $\widetilde{Y}$ and $\widetilde{M}$) and, for any $b^k \in Y$, let $q^k = q(\tilde{x}, b^k)$ be the associated queue vector. If $c(q^k) \leqslant \tilde{\xi}$ for all $k \in M$, then $(\tilde{x}, \tilde{s}, \tilde{\xi})$ can be extended to an optimal solution $(\tilde{x}, s, \tilde{\xi})$ of the original aR-MS formulation (10) with the same cost $\tilde{\xi}$.*

[2] Observe that we also drop the variables s^i_{gl}, for $i \in M \setminus \widetilde{M}$ as they do not appear in any constraint or in the objective function and can be fixed to 0.

Proof. All we have to do is to let $s^k_{gl} = q^k_{gl}$ for $k \in M \setminus \widetilde{M}$ and $s^i_{gl} = \tilde{s}^i_{gl}$ for $i \in \widetilde{M}$. It is immediate to see that the solution $(\tilde{x}, s, \tilde{\xi})$ is feasible for (10). □

So, if the conditions of the above proposition are satisfied, the current optimal vector $\tilde{x}$ is an optimal MS and we do not need to solve the overall problem (10). Otherwise there exists $\hat{b} \in Y \setminus \widetilde{Y}$ such that $c(q(\tilde{x}, \hat{b})) > \tilde{\xi}$, we let $\widetilde{Y} = \widetilde{Y} \cup \{\hat{b}\}$ and solve the associated master program once again. Observe that the new problem contains $|G| \cdot |L| + 1$ additional constraints and $|G| \cdot |L|$ additional variables.

In order to test if the conditions of Proposition 4.1 are satisfied or to identify a demand vector $\hat{b} \in Y$ violating such conditions we set up a suitable MILP. In particular, we look for a demand vector $b^* \in Y$ maximizing the cost of the queue $q^* = q(\tilde{x}, b^*)$. If $c(q^*) > \tilde{\xi}$ then b^* is the required demand vector, otherwise $c(q^k) \leqslant \tilde{\xi}$ for all $k \in M$ (with $q^k = q(\tilde{x}, b^k)$) and $\tilde{x}$ is an optimal MS.

To this end, we can solve the following mixed integer non-linear program:

$$\begin{aligned} \max \quad & \sum_{g\in G}\sum_{l\in L} c_{gl} w_{gl} \\ \text{s.t.} \quad & (i) \quad w_{gl} = \max\left(0, y_{gl} - \sum_{p\in P(l)} \tilde{x}_{gp}\right) \quad g \in G, \quad l \in L \\ & w \in \mathbb{R}^{G\times L}, \quad y \in Y \end{aligned} \tag{12}$$

The non-linear constraints (12.i) ensure that $w = q(\tilde{x}, y)$, i.e. w is the queue associated with the demand vector $y \in Y$ (and MS $\tilde{x}$). In order to linearize constraint (12.i) we cannot proceed as for the master problem simply by introducing a suitable slack variable, because of the specific form of the objective function in (12). Instead we introduce, for each $g \in G$, $l \in L$, a binary variable z_{gl} which is 1 if $y_{gl} - \sum_{p\in P(l)} \tilde{x}_{gp} < 0$, and 0 otherwise. Then we substitute each constraint (12.i) with the following linear constraints:

$$\begin{aligned} &(\text{i}) \quad w_{gl} \leqslant Q z_{gl} + y_{gl} - \sum_{p\in P(l)} \tilde{x}_{gp} \\ &(\text{ii}) \quad w_{gl} \leqslant Q(1 - z_{gl}) \\ &(\text{iii}) \quad w_{gl} \geqslant 0 \end{aligned} \tag{13}$$

where Q is a suitable large constant. It is not difficult to see that $y_{gl} - \sum_{p\in P(l)} \tilde{x}_{gp} < 0$ (no queue) implies $z_{gl} = 1$ and, in turn, $w_{gl} = 0$ (from (13ii) and (13.iii)). On the other hand, when $z_{gl} = 0$ then (13ii) becomes redundant while (13i) becomes

$$w_{gl} \leqslant y_{gl} - \sum_{p\in P(l)} \tilde{x}_{gp}.$$

In this case, the strictly positive coefficient in the objective function implies that $w_{gl} = y_{gl} - \sum_{p\in P(l)} \tilde{x}_{gp}$ in every optimal solution to the linearized version of (12).

The implementor/adversary algorithm. We come back now to the relation between our column and row generation approach and the implementor/adversary (*I/A*) algorithm for robust optimization proposed by Bienstock [9] (we use here the same notation introduced by Bienstock and also the algorithm definition is carbon-copied from the original).

The I/A algorithm solves a problem of the form

$$F^* = \min_{x\in X} \max_{y\in Y} f(x, y). \tag{14}$$

and can be summarized as follows:

implementor/adversary Algorithm

Output: values L and U with $L \leqslant F^* \leqslant U$, and $x^* \in X$ such that $L = \max_{y\in \widetilde{Y}} f(x^*, y)$.

Initialization: $\widetilde{Y} = \emptyset$, $L = -\infty$, $U = +\infty$.

Iterate:

1. **Implementor Problem:** solve $\min_{x\in X} \max_{y\in \widetilde{Y}} f(x, y)$, with solution x^*. Reset $L \leftarrow \max_{y\in \widetilde{Y}} f(x^*, y)$
2. **Adversary Problem:** solve $\max_{y\in Y} f(x^*, y)$, with solution y^*. Reset $U \leftarrow \min\{f(x^*, y^*), U\}$.
3. **Test:** If $U - L$ is small, exit, else reset $\widetilde{Y} \leftarrow \widetilde{Y} \cup \{y^*\}$ and go to **1**.

It is immediate to verify that the implementor problem at Step 1 of the above algorithm coincides with the master problem (11) associated with the set $\widetilde{Y}$. We have already observed that the optimal value of the master program can only increase as the set $\widetilde{Y}$ gets larger, since the minimization problem becomes increasingly more constrained. This is why at Step 1 we update the lower bound L with the optimal value of the current master program. In contrast, the solution of the adversary (slave) problem at Step 2 depends on the current master solution x^* and, though always providing an upper bound for the original problem, such value can change unpredictably from one iteration to the next.

The dynamic simplex algorithm for problem (9) can thus be viewed as a possible implementation of the I/A algorithm. We can then apply some classical enhancements of the dynamic simplex algorithm. In particular, even if the master problem (11) is a MILP, for a suitable number of (typically initial) iterations we can generate rows by solving the linear relaxation and separating the corresponding fractional solution, significantly speeding up the algorithm. This is explained more in detailed in the following paragraph.

Speeding up the I/A algorithm. For our instances, the adversary problem is a relatively small integer program, which is actually solved quite efficiently by any commercial solver or by dynamic programming. In contrast, the implementor problem is the large MILP (11) and most computing time is spent for its solution. As observed in [9], a smart choice of the initial set $\widetilde{Y}$ of demand vectors can significantly speed up the convergence of the I/A algorithm. In order to identify a good initial set we proceed as follows. Solve the linear relaxation of (11) and denote by $\bar{x}$ the optimal (possibly fractional) MS and by $\bar{\xi}$ its value; then it is not difficult to see that:

- $\bar{\xi}$ is a lower bound for the overall min–max problem (9) and L can still be updated as at Step 1 of the I/A algorithm.
- the adversary problem can still be defined and solved using the fractional solution $\bar{x}$: indeed its optimal solution y^* is a feasible demand vector (i.e. $y^* \in Y$). Such demand vector is used to extend the current set $\widetilde{Y}$. In contrast, its value is not (in general) an upper bound for the overall min–max problem, since $\bar{x}$ is not a feasible schedule.

So, in the first iterations of the I/A algorithm, at Step 1 we solve the relaxed version of the master program. When the upper bound returned by the adversary coincides with the lower bound returned by the implementor, we resort to the original integer master program for the remaining iterations. The effect of this is to quickly populate the set $\widetilde{Y}$ (recall that at each iteration of the I/A algorithm the current set $\widetilde{Y}$ is extended with a new demand vector returned by the adversary). More formally, the solution approach can be viewed as decomposed in two phases. In a first phase (later refer to as *relaxed I/A algorithm*) the program (11) realizing the implementor is relaxed to an LP in order to establish a promising demand vectors set $\widetilde{Y}$. In the second phase the standard I/A algorithm is executed, with initial demand vectors set $\widetilde{Y}$. This choice proved to be effective to reduce overall computational times, as shown in the next section.

5. Computational experience

Our experiments deal with the Master Surgery Scheduling Problem (MSS), which amounts to assigning operation rooms to surgical specialties (see [16] for a detailed description of the problem and of the corresponding pattern formulation). In all cases we have six surgery specialties, or groups, namely Gastroenterology (*Group 1*), General Cardiology (*Group 2*), Gynecology (*Group 3*), Medicine (*Group 4*), Orthopedics (*Group 5*), and Urology (*Group 6*). Each group (specialty) must be assigned a suitable number of slots of operation room in the planning horizon.

The computational experience has two major purposes. Firstly, to evaluate our specific implementation choices and assess the quality of the algorithm. Secondly, to evaluate the capability of the approach to cope with practical real-life scheduling problems of medical resources in hospitals.

The algorithm was tested on a linux computer with 8 GB memory and 43 GHz multi-core i7–870 processors, of which only one thread was used for computations. The code was implemented in *python*, using CPLEX 12.2 to solve the linear and integer programs.

For all our experiments, we let the objective function be the following convex piece-wise linear function of the queue q:

$$\sum_{g \in G} \sum_{l \in L} l \cdot c(q_{gl}) \cdot q_{gl}$$

where $c(r) = 1$ if $0 \leqslant r \leqslant 3$ ($0 \leqslant r \leqslant 1$ for the artificial, smaller instance), and $c(r) = 3$ otherwise. In other words, the marginal cost is larger for long queues; also, patients with longer operation durations are more costly. Other choices of the coefficients can of course be made by the hospital management to reflect specific priorities.

5.1. Experiments for algorithm assessment

This first set of experiments is designed to evaluate some implementation choices and the general behavior of the algorithm. To this purpose we created a small artificial data set, described in Table 1, corresponding to one week demand.

In this experiment we ran the standard (non-relaxed, phase 2) version of the I/A algorithm. The different specialities are given in each row whereas the demand for each block length is given in the columns labelled from 1 to 4. The two numbers separated by hyphen in each entry are lower and upper limits on the demand, respectively. The last two columns report, for each specialty, the total (minimum and maximum) number of patients and the total (minimum and maximum) number of slots required in the planning horizon, respectively.

The schedule considered a planning horizon of 1 week, with 5 days and 6 slots/day. The medical resources amounted to 5 rooms. Consequently, we have 150 superslots (=# days × # slots × #rooms) and correspondingly, 2700 decision binary variables. For this experiment, we fixed the total slot demand $K = 150$. In a first run we evaluate the original version of the implementor/adversary algorithm, that is at each iteration both master and slave problems are solved to integral optimality.

The results are plotted in Fig. 1. The algorithm ran for 2545 seconds. and for 36 iterations in total. For each iteration we plot the lower bound returned by the implementor (lower dotted line) and the upper bound returned by the adversary (upper solid line), respectively. The value computed at each iteration corresponds to a small cross. Remarkably, already after 22 iterations, the gap was reduced to less than 5%. Actually, the corresponding implementor solution was globally optimal, but we needed other 14 iterations to prove it. It is important to remark here that, at any iteration i the solution x^i produced by the ith master program corresponds to a feasible schedule. So, when solving the following adversary problem, we are actually computing the worst-case cost when the schedule is x^i. This cost is thus the upper bound depicted, for our tiny instance, by the small cross points in the upper line of Fig. 1. It is not difficult to see that already in the very early iterations of the algorithm the returned schedules are quite good, even if later on in the process we also obtain much worse solutions. A similar behavior was also observed in several experiments on random instances as reported in [14].

Finally, the time required to solving the adversary problem sums up to less than 4% of the overall running time. For this reason, we concentrated on the implementor implementation when striving for speed-ups.

To this end, we run the two phase solution algorithm described at the end of Section 4. In Table 2 we summarize the benefits of running phase 1 with the relaxed I/A algorithm against the standard I/A algorithm (with no phase 1). We also tested different techniques to populate the set $\widetilde{Y}$ with a number of pre-computed demand scenarios before starting the I/A algorithm. In Table 2 we show the figures only for the scenario generation technique providing the best results, which goes as follows: First observe that in the adversary problem (12) we can replace the feasible demand set Y with its convex hull conv (Y) and the optimal solutions stay unchanged. Also, it is immediate to see that, for any $\tilde{x}$, there exists an optimal solution (w^*, y^*) to (12) such that y^* is a non-dominated vertex of conv (Y). So, we generate an initial set of scenarios by simply picking at random (with uniform distribution) the non-dominated vertices of conv (Y), disregarding the dominated ones. The initial set of demand vectors is denoted by $\widetilde{Y}^0$.

In Table 2, each row corresponds to a specific value of the parameter K and number of pre-computed scenarios (by our randomized technique). For fixed K, the set of pre-computed scenario

Table 1
The small artificial data set.

	Block lengths				Total patients	Total slots
	1	2	3	4		
	Patients					
Group 1	5–7	2–5	3–5	0–1	10–18	18–36
Group 2	4–5	1–3	1–3	4–7	10–18	25–48
Group 3	2–3	5–6	1–3	2–4	10–16	23–40
Group 4	2–3	4–6	0–0	2–4	8–13	18–31
Group 5	3–5	5–6	2–4	1–2	11–17	23–37
Group 6	4–7	3–4	3–5	1–2	11–18	23–38
Sum	20–30	20–30	10–20	10–20	60–100	130–230

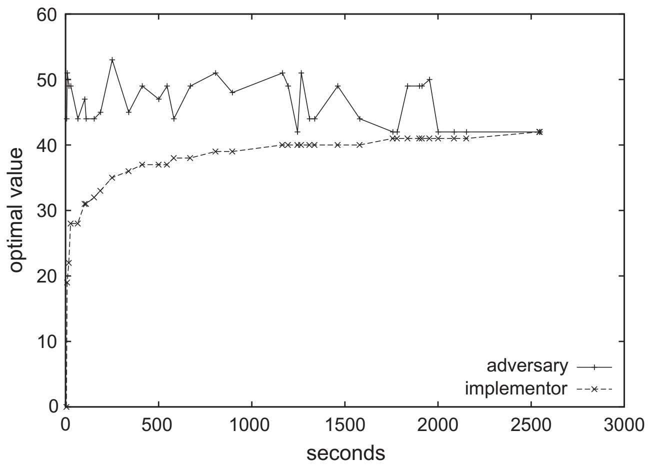

Fig. 1. Run plot: small instance. The lower line corresponds to the lower bounds returned by the implementor. The upper line shows the upper bounds returned by the adversary.

Table 2
Enhancing the standard I/A algorithm.

K	$\|\bar{Y}^0\|$	Phase 2 (alone)		Phase 1 + phase 2	
		Iter.	Time (seconds)	Iter.	Time (seconds)
150	0	39	2369	42	204
150	24	31	1186	33	258
150	50	33	1261	36	355
150	100	20	2315	23	421
150	200	15	1970	15	460
150	400	6	3198	9	519
150	800	2	604	4	371
155	0	77	3158	80	941
155	24	75	4133	70	1062
160	0	246	29212	233	11980

in a given row includes the set in the previous row. Columns 3 and 4 are the number of iterations and the total running time when only phase 2 (standard I/A algorithm) is executed. Columns 5 and 6 report the same figures when both phase 1 and phase 2 are applied. The running times include the (negligible) time to generate the pre-computed scenarios.

The results are somehow striking. The rows with 0 pre-computed scenarios show that running phase 1 is very beneficial, ranging from the 60% improvement in running time when $K = 160$ up to 92% improvement when $K = 150$. Concerning pre-computed scenarios, one can notice that this generally decreases running times when only phase 2 is run, but the effect is opposite when phase 1 is run first. Typically, the more scenarios are generated, the less iterations are necessary to converge. However, this is counterbalanced by the fact that each iteration becomes more time consuming. When looking at the figures, always recall that each iteration in phase 1 is much cheaper than in phase 2, since in the latter we solve an integer program. This is why, e.g., in the first row the 36 iterations of Column 3 require much more time than 42 iterations of Column 5.

We complete this subsection by discussing experiments which highlight the dramatic effect of applying the dynamic simplex method VS the direct solution of the original (full) problem (10), even for small instances as the one considered here. In particular, we again consider different values for the constant K, showed in the first column of Table 3. The second column is the corresponding number of different (non-dominated) demand vectors. The third column is the time spent to solve the corresponding large MILP. Remark that we do not include the time spent to generate the scenarios. The last two columns refer to the performance of the dynamic simplex algorithm.

Even though for small values of K the "brute force" approach may appear to be competitive, for $K \geqslant 138$ the running time to solve the full original MILP exceeds time limits (30 minutes). In contrast, for all values of K the dynamic simplex method solves the problem to optimality in a few seconds.

From the table it is apparent that, even for such a small instances, as soon as the number of feasible scenarios increases, the "brute force" approach becomes inapplicable.

Table 3
Comparison of dynamic simplex algorithm versus solving the full original problem.

K	Full		Dynamic simplex	
	Size	Time	Time (seconds)	# Iter.
135	540	0.4 seconds	19	19
136	1250	9 seconds	19	14
137	2707	24 seconds	21	16
138	5543	NA	5	6
140	20,218	NA	31	20

5.2. Real-life test case

Our real-life test case refers to historical patient data from SAB (*Sykehuset Asker og Bærum HF*), a major hospital in the city of Oslo (see also [14,16]).

The data were collected at SAB during three years, from 2007 to 2009. We have made this data publicly available for the research community, see [17]. According to such data, each day is subdivided into four 2-h slots and the feasible patterns are built consequently. The hospital hosts the six surgical groups listed at the beginning of this section. For a given month and for each group g, the actual operations were partitioned according to their surgery length into four classes $K_g^l,\ l = 1, \ldots, 4$. Class K_g^1 contains all operations lasting at most one hour, class K_g^2 those lasting between one and three hours, etc. Then the demand (i.e. number of operations) for each surgical group and each block length were computed accordingly.

Namely, for each specialty $g \in G$ and each possible block length $l \in L$, we consider the minimum m_{gl} and the maximum value M_{gl} achieved in a 7 week-period (corresponding to our planning horizon) of the three years and select at random the lower bound $\beta_{gl} \in [m_{gl}, M_{gl}]$. Similarly, we take the upper bound γ_{gl} at random in the interval $[M_{gl}, 1.5M_{gl}]$. This is to cope with possible future increase in demand. Of course, other choices were also possible as, e.g., choosing the lower and the upper bounds to a multiple of m_{gl} and M_{gl}, respectively. Indeed, the vectors β and γ are design parameters and can be established by the practitioners according to their preferences and targets. In this context, we are mostly interested in showing the feasibility of the approach, and every sensible choice is suitable, also considering that the size of the instances is substantially not affected.

The final scenario is summarized in Table 4.

We choose a planning horizon of 7 weeks, with 5 days a week and 6 slots a day, giving 210 slots for the 6 groups. With 5 rooms available, this makes 1050 superslots available.

We run our tests for different values of the parameter K (the expected total demand of slots) ranging in the interval [903, 1179].

Measuring robustness. In our specific application, the goal of robust optimization is to build prudent solutions, that is solutions which are intended to be protected against the occurrence of the worst-scenario (w.r.t. the solution built). We first use this criterion to compare our robust schedule against two deterministic schedules (found by selecting two suitable demand vectors). Each of the three schedules is evaluated when the worst possible scenario (for each of them) occurs. The worst possible scenario is simply computed by solving the adversary problem (12) for each schedule.

We will then also report a second standard robustness measure, namely the *regret*, which compares the robust solution evaluated when its worst possible scenario occurs against the best possible solution with respect to the same scenario.

Table 4
Patient demand, real-life data set. Number of weeks and decision variables increased 7-fold compared to the small instance.

	Block lengths				Total patients	Total slots
	1	2	3	4		
	Patients					
Group 1	7–10	55–70	14–20	2–3	78–103	167–222
Group 2	14–18	62–75	2–4	0–0	78–97	144–180
Group 3	31–40	84–95	1–2	0–0	116–137	202–236
Group 4	13–15	20–35	1–2	0–0	34–52	56–91
Group 5	5–10	73–90	17–25	1–2	96–127	206–273
Group 6	16–25	53–70	2–4	0–0	71–99	128–177
Sum	86–118	347–435	37–57	3–5	473–615	903–1179

Table 5
Comparing the I/A algorithm against the deterministic approach with demand fixed to maximum and to average values.

K	Absolute Robustness			Regret		
	$c(q^R)$	$c(q^M)$	$c(q^A)$	(x^R)	(x^M)	(x^A)
903	0	19	5	0	19	5
925	42	85	42	24	67	24
950	94	135	100	9	70	39
975	148	174	148	69	91	14
1000	181	195	187	3	82	6
1025	189	204	214	4	73	42
1050	200	213	235	15	28	84
1075	206	219	263	21	34	78
1100	210	225	275	25	31	90
1125	215	225	275	22	31	90
1150	219	225	275	25	31	90
1179	225	225	275	0	0	50

The two deterministic solutions are obtained by solving the deterministic scheduling problem (7) for two given demand vectors in Y.

In particular, we consider two possible scenarios: one in which all demands are at their upper bounds γ_{gl} and one in which demands are at their average values $\lfloor(\beta_{gl}+\gamma_{gl})/2\rfloor$. Let us denote by x^M and x^A the corresponding optimal schedules, and by x^R the robust solution produced by the I/A algorithm. We then compute the worst possible demand vector b^R, b^M and b^A for x^R, x^M and x^A, respectively. This can be done by solving the adversary problem (12) with input solution x^R, x^M and x^A, respectively. Observe that in order to solve the adversary problem (12) we need to establish the maximum total demand K. In our experiments, we simply run our comparisons for different values of K (in a practical context this parameter is decided by the hospital board). Once b^R, b^M and b^A are generated, we can compute the corresponding costs, by first evaluating the queues $q^R = q(x^R, b^R)$, $q^M = q(x^M, b^M)$ and $q^A = q(x^A, b^A)$ and finally their costs $c(q^R)$, $c(q^M)$, $c(q^A)$.

As said, we also evaluate the regret. For a given solution x^R, let b^R be the associated worst demand vector and q^R be the corresponding queue. Now, let x^* be the best possible schedule when the demand is b^R, which can be computed by solving problem (7) with demand b^R, and let $q^* = q(x^*, b^R)$ be the corresponding queue. The regret reg (x^R) is defined as reg $(x^R) = c(q^R) - c(q^*)$: informally, it measures how better we could have performed if we had chosen a better schedule for the given input.

The results are shown in Table 5. For the different values of K (first column), the next three columns display the corresponding cost of the solutions generated by the three different approaches. In particular, for a given K, the corresponding entry in the second column $c(q^R)$ is the cost of the I/A solution with input K. The next two costs $c(q^M)$ and $c(q^A)$ are computed by running the adversary problem with input K and schedule x^M and x^A, respectively. The figures in Table 5 clearly show that the robust approach outperforms the other approaches when considering the Absolute Robustness measure.

Also, for medium/high total demand K, the deterministic approach with maximum demand seems to perform better than the deterministic approach with average demand, whereas the situation is reversed for low total demand values.

Concerning the regrets, shown in the last three columns, the robust solution (column (x^R)) performs better in almost all cases. This is a surprising positive result as the algorithm is tailored to optimize a different measure.

Robust schedules with different estimations and realizations of the parameter K. Robust scheduling considers all possible scenarios in a given uncertainty set Y. The uncertainty set must be determined in a prior phase, and, in our case, also depends on the estimated total demand K: to highlight such dependency we let $Y = Y(K)$. However, once a suitable value $\overline{K}$ for K is established (by some statistical analysis, beyond the scope of the present paper), and an optimal robust schedule $\bar{x}$ w.r.t. $Y(\overline{K})$ is computed, then a different value of total demand may actually occur in practice. How "robust" is our robust schedule $\bar{x}$ against a wrong estimation of the parameter K? In principle, answering this question requires the definition of a new robust model, but this would go beyond the scope of the current paper. We give an alternative, experimental answer to this question in Table 6. Namely, we use the estimated value $\overline{K}$ of K to run the I/A algorithm with uncertainty set $Y(\overline{K})$ and compute a robust scheduling $x(\overline{K})$. But then we suppose that the actual total demand $\widetilde{K}$ may be different from the one used to compute the robust schedule. Next, we compute the worst possible demand vector in $Y(\widetilde{K})$ w.r.t. $x(\overline{K})$ for several values of "actual total demand" $\widetilde{K}$. Every column in Table 6 is built in this way. In particular, for each estimated total demand $\overline{K}$ and each actual total demand $\widetilde{K}$, both ranging from 903 to 1179, we show the cost of the queue associated with the robust schedule $x(\overline{K})$ and the worst demand vector in $Y(\widetilde{K})$.

Not surprisingly, the above table shows that, for a given value of the actual demand the minimum cost is achieved when it equals the estimated one (diagonal entries in bold). Most important, the table shows that when the estimated demand is close to the actual one, then the solution cost does not increase too much. In particular, in most cases it is preferable to overestimate rather than underestimate. Observe that the ratio between the largest and the smallest entry in a row can be quite large (over 100% in some cases). This means that a non-careful planning can actually be very costly for a hospital, if a worst-case scenario occurs.

Concluding, this experiment shows that if the hospital estimates are reasonably close to the actual demand, then our approach is able to generate safe solutions.

Table 6
Solution costs for different values of estimated and actual overall demand.

Actual demand $\widetilde{K}$	Estimated demand $\overline{K}$											
	903	925	950	975	1000	1025	1050	1075	1100	1125	1150	1179
903	**0**	5	7	5	7	11	25	43	19	19	52	19
925	54	**42**	53	68	55	63	88	106	73	85	115	85
950	111	100	**94**	119	109	108	133	151	121	132	160	135
975	168	148	151	**148**	148	154	151	171	172	166	182	174
1000	223	187	196	187	**181**	181	182	183	188	181	192	195
1025	268	221	235	206	202	**189**	194	196	195	194	197	204
1050	284	231	270	227	227	212	**200**	202	201	203	201	213
1075	287	243	296	237	237	223	212	**206**	206	208	207	219
1100	289	245	323	242	242	234	218	212	**210**	212	213	225
1125	289	245	329	243	246	239	222	216	216	**215**	218	225
1150	289	247	333	243	251	244	227	221	221	221	**219**	225
1179	289	247	336	243	257	245	230	225	225	225	225	**225**

Current findings and future developments. Summarizing, in this paper we introduce a model for a cyclic and robust scheduling problem in health-care and we develop a row and column generation approach for its solution. Also, we show how to interpret this algorithm as the implementor/adversary algorithm introduced by Bienstock for portfolio optimization. Finally, we solve real-life instances of surgery scheduling provided by a major hospital in the city of Oslo.

Interesting future investigations may concern both the model and the algorithms. In particular, one major question is the destiny of the unserved patients. One may expect that a share of such patients will pop up again in a subsequent period. A multi-period robust model should then be considered to tackle this case. Another interesting development would be to study different definitions for the set Y and the corresponding properties, so as to restrict the computations to suitable subsets of Y.

Acknowledgements

We wish to thank two anonymous referees for their precious comments and suggestions.

References

[1] D. Alvras, M.W. Padberg, Linear Optimization and Extensions: Problems and Solutions, Springer-Verlag, Berlin, Germany, 2001.

[2] A. Atamtürk, M. Zhang, Two-stage robust network flow and design under demand uncertainty, Operations Research 55 (4) (2007).

[3] I. Averbakh, V. Lebedev, On the complexity of minimax regret linear programming, European Journal of Operational Research 160 (2005) 227–231.

[4] J. Beliën, Exact and Heuristic Methodologies for Scheduling in Hospitals: Problems, Formulations and Algorithms. PhD Thesis, Faculty of Business and Economics, Katholieke Universiteit Leuven, 2006.

[5] J. Beliën, E. Demeulemeester, Building cyclic master surgery schedules with leveled resulting bed occupancy, European Journal of Operational Research 176 (2) (2007) 1185–1204.

[6] A. Ben-Tal, L. El Ghaoui, A. Nemirovski, Robust optimization methodology and applications, Mathematical Programming 92 (3) (2002) 453–480.

[7] A. Ben-Tal, A. Goryashko, E. Guslitzer, A. Nemirovski, Adjustable robust solutions of uncertain linear programs, Mathematical Programming 99 (2) (2004) 351–376.

[8] D. Bertsimas, M. Sim, The price of robustness, Operations Research 52 (1) (2004) 35–53.

[9] D. Bienstock, Histogram models for robust portfolio optimization, Journal of Computational Finance 11 (2007) 1–64.

[10] B. Cardoen, E. Demeulemeester, J. Beliën, Operating room planning and scheduling: a literature review, European Journal of Operational Research 201 (3) (2010) 921–932.

[11] G. Desaulniers, J. Desrosiers, M.M. Solomon (Eds.), Column Generation, Springer, USA, 2005.

[12] M. Fischetti, M. Monaci, Light robustness, Book Series Lecture Notes in Computer Science 5868 (3) (2009) 61–84.

[13] E.W. Hans, G. Wullink, M. van Houdenhoven, G. Kazemier, Robust surgery loading, European Journal on Operational Research 185 (3) (2008) 1038–1050.

[14] M. Holte, A Cutting Plane Algorithm for Robust Scheduling Problems in Medicine, Master Thesis University of Oslo, 2010.

[15] P. Kouvelis, G. Yu, Robust Discrete Optimization and Its Applications, Kluwer Academic Publishers, 1997.

[16] C. Mannino, E.J. Nilssen, T.E. Nordlander, A pattern based, robust approach to cyclic master surgery scheduling, Journal of Scheduling 15 (5) (2012) 553–563.

[17] C. Mannino, E.J. Nilssen, T.E. Nordlander, Sintef ict: Mss-adjusts surgery data, February 2010. <http://www.comihc.org/index.php/Test-Beds/admission-planning-benchmarks.html>.

[18] G. Nemhauser, L. Wolsey, Integer and Combinatorial Optimization, John Wiley & Sons, Hoboken, USA, 1988.

[19] J.M. Oostrum, M. van Houdenhoven, J.L. Hurink, E.W. Hans, G. Wullink, G. Kazemier, A master surgical scheduling approach for cyclic scheduling in operating room departments, OR Spectrum 30 (2) (2008) :355–374.

[20] J. Patrick, M.L. Puterman, M. Queyranne, Dynamic multipriority patient scheduling for a diagnostic resource, Operations Research 56 (6) (2008) 1507–1525.

[21] A. Rais, A. Viana, Operations research in healthcare, International Transactions in Operational Research (2010) 1–31.