An ASP-based Framework for Operating Room Scheduling

Carmine Dodaro [a], Giuseppe Galatà [b], Marco Maratea [a], Ivan Porro [b],

[a] *Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genova, Viale F. Causa 15, 16145, Genova (GE), Italia*
E-mail: {dodaro,marco}@dibris.unige.it
[b] *SurgiQ srl, Piazza della Vittoria 12/12, 16121, Genova (GE), Italia*
E-mail: {name.surname}@surgiq.com

Abstract. The Operating Room Scheduling (ORS) problem is the task of assigning patients to operating rooms, taking into account different specialties, the surgery and operating room session durations, and different priorities. Given that Answer Set Programming (ASP) has been recently employed for solving real-life scheduling and planning problems, in this paper we first present an off-line solution based on ASP for solving the ORS problem. Then, we present techniques for re-scheduling on-line in case the off-line schedule can not be fully applied. Results of an experimental analysis conducted on benchmarks with realistic sizes and parameters show that ASP is a suitable solving methodology also for the ORS problem. This analysis has been performed with a web framework for managing ORS problems via ASP that allows a user to insert the main parameters of the problem, solve a specific instance, and show results graphically in real-time.

1. Introduction

The Operating Room Scheduling (ORS) [1,13,40,43] problem is the task of assigning patients to operating rooms, taking into account different specialties, surgery durations, and operating room session durations. Given that patients may have priorities, the solution has to find an accommodation for the patients with highest priorities, and then to the other with lower priorities if space is still available. A proper solution to the ORS problem is crucial for improving the whole quality of the health-care and the satisfaction of patients. Indeed, modern hospitals are often characterized by long surgical waiting lists, which are caused by inefficiencies in operating room planning, leading to an obvious dissatisfaction of patients.

Complex combinatorial problems, possibly involving optimizations, such as the ORS problem, are usually the target applications of knowledge representation and reasoning formalisms such as Answer Set Programming (ASP). Indeed, its simple but rich syntax [18], which includes optimization statements as well as powerful database-inspired constructs like aggregates, and its intuitive semantics, combined with the readability of specifications (always appreciated by users) and availability of efficient solvers (see, e.g., [4,8,31,42]), make ASP an ideal candidate for addressing such problems, as witnessed by the ASP Competition series (see [19,30,32,33,34,35,41] for the last editions). Indeed, ASP has been already successfully employed for solving hard combinatorial and application problems in several research areas, including Artificial Intelligence [11,14,22], Bioinformatics [25], Hydroinformatics [29], Game theory [12], Knowledge management on the Web [3], and also employed in industrial applications (see, e.g., [2,23]).

In this paper we first present an off-line solution schedule based on ASP for solving the ORS problem, where problem specifications are modularly expressed as ASP rules, and ASP solvers are used to solve the resulting ASP program. Then, we also present techniques for re-scheduling on-line in case the off-line solution can not be fully applied given, e.g., some patients could not be operated in their assigned slot and have to be reallocated; in this case, the aim is of minimizing the changes needed to accommodate the new situation. Again, the re-scheduling is specified by modularly adding ASP rules to (part of) the (updated) original ASP encoding. We have then run a wide experimental analysis on ORS benchmarks with realistic sizes and parameters inspired from data of a hospital in the north-east of Italy. We have also performed a scalability analysis on the performance of the employed ASP solver and encoding for the scheduling problem w.r.t. schedule length. Overall, results show that ASP is a suitable solving methodology also for ORS, given that a high efficiency, defined as occupation of rooms, can be achieved in short timings in line with the need of the application. Additionally, we have also designed and implemented a web framework for managing ORS problems via ASP that allows a user to insert the main parameters of the problem, solve a specific instance, and show results graphically in real-time, and where the analysis mentioned before was indeed run.

To summarize, the main contributions of this paper are the following:

- We provide a formal mathematical description of the ORS problem (Section 4).
- We solve the ORS problem using an ASP encoding (Section 5).
- We report on an experimental analysis assessing the good performance of our ASP solution (Section 6).
- We describe a Graphical User Interface (GUI) which uses our ASP solution to produce a real-time scheduling of operating rooms (Section 7).

The paper is completed by Section 2, which contains needed preliminaries about ASP, Section 3, which describes the problem informally, Section 8, which analyzes related literature, and by conclusions in Section 9.

2. Background on ASP

Answer Set Programming (ASP) [15] is a programming paradigm developed in the field of nonmonotonic reasoning and logic programming. In this section we overview the language of ASP. More detailed descriptions and a more formal account of ASP, including the features of the language employed in this paper, can be found in [15,18]. Hereafter, we assume the reader is familiar with logic programming conventions.

Syntax. The syntax of ASP is similar to the one of Prolog. Variables are strings starting with uppercase letter and constants are non-negative integers or strings starting with lowercase letters. A *term* is either a variable or a constant. A *standard atom* is an expression $p(t_1, \dots, t_n)$, where p is a *predicate* of arity n and $t_1, \dots, t_n$ are terms. An atom $p(t_1, \dots, t_n)$ is ground if $t_1, \dots, t_n$ are constants. A *ground set* is a set of pairs of the form $\langle consts : conj \rangle$, where $consts$ is a list of constants and $conj$ is a conjunction of ground standard atoms. A *symbolic set* is a set specified syntactically as $\{Terms_1 : Conj_1; \cdots ; Terms_t : Conj_t\}$, where $t > 0$, and for all $i \in [1, t]$, each $Terms_i$ is a list of terms such that $|Terms_i| = k > 0$, and each $Conj_i$ is a conjunction of standard atoms. A *set term* is either a symbolic set or a ground set. Intuitively, a set term $\{X : a(X, c), p(X); Y : b(Y, m)\}$ stands for the union of two sets: the first one contains the X-values making the conjunction $a(X, c), p(X)$ true, and the second one contains the Y-values making the conjunction $b(Y, m)$ true. An *aggregate function* is of the form $f(S)$, where S is a set term, and f is an *aggregate function symbol*. Basically, aggregate functions map multisets of constants to a constant. The most common functions implemented in ASP systems are the following:

- *#count*, number of terms;
- *#sum*, sum of integers.

An *aggregate atom* is of the form $f(S) \prec T$, where $f(S)$ is an aggregate function, $\prec \in \{<, \leq, >, \geq, \neq, =\}$ is a comparison operator, and T is a term called guard. An aggregate atom $f(S) \prec T$ is ground if T is a constant and S is a ground set. An *atom* is either a standard atom or an aggregate atom. A *rule* r has the following form:

$$a_1 \vee \ldots \vee a_n \text{ :- } b_1, \ldots, b_k, \mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_m.$$

where $a_1, \dots, a_n$ are standard atoms, $b_1, \dots, b_k$ are atoms, $b_{k+1}, \dots, b_m$ are standard atoms, and $n, k, m \geq 0$. A literal is either a standard atom a or its negation $\mathit{not}\ \ a$. The disjunction $a_1 \vee \ldots \vee a_n$ is the *head* of r, while the conjunction $b_1, \dots, b_k, \mathit{not}\ b_{k+1}, \dots, \mathit{not}\ b_m$ is its *body*. Rules with empty body are called *facts*. Rules with empty head are called *constraints*. A variable that appears uniquely in set terms of a rule r is said to be *local* in r, otherwise it is a *global* variable of r. An ASP program is a set of *safe* rules, where a rule r is *safe* if both the following conditions hold: *(i)* for each global variable X of r there is a positive standard atom ℓ in the body of r such that X

appears in ℓ; and *(ii)* each local variable of r appearing in a symbolic set $\{Terms : Conj\}$ also appears in $Conj$.

A *weak constraint* [16] ω is of the form:

$$:\sim b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m.\ [w@l]$$

where w and l are the weight and level of ω, respectively. (Intuitively, $[w@l]$ is read "as weight w at level l", where weight is the "cost" of violating the condition in the body of w, whereas levels can be specified for defining a priority among preference criteria). An ASP program with weak constraints is $\Pi = \langle P, W \rangle$, where P is a program and W is a set of weak constraints.

A standard atom, a literal, a rule, a program or a weak constraint is *ground* if no variables appear in it.

Semantics. Let P be an ASP program. The *Herbrand universe* U_P and the *Herbrand base* B_P of P are defined as usual. The ground instantiation G_P of P is the set of all the ground instances of rules of P that can be obtained by substituting variables with constants from U_P.

An *interpretation* I for P is a subset I of B_P. A ground literal ℓ (resp., $not\ \ell$) is true w.r.t. I if $\ell \in I$ (resp., $\ell \notin I$), and false (resp., true) otherwise. An aggregate atom is true w.r.t. I if the evaluation of its aggregate function (i.e., the result of the application of f on the multiset S) with respect to I satisfies the guard; otherwise, it is false.

A ground rule r is *satisfied* by I if at least one atom in the head is true w.r.t. I whenever all conjuncts of the body of r are true w.r.t. I.

A model is an interpretation that satisfies all rules of a program. Given a ground program G_P and an interpretation I, the *reduct* [26] of G_P w.r.t. I is the subset G_P^I of G_P obtained by deleting from G_P the rules in which a body literal is false w.r.t. I. An interpretation I for P is an *answer set* (or stable model) for P if I is a minimal model (under subset inclusion) of G_P^I (i.e., I is a minimal model for G_P^I) [26].

Given a program with weak constraints $\Pi = \langle P, W \rangle$, the semantics of Π extends from the basic case defined above. Thus, let $G_\Pi = \langle G_P, G_W \rangle$ be the instantiation of Π; a constraint $\omega \in G_W$ is violated by an interpretation I if all the literals in ω are true w.r.t. I. An *optimum answer set* for Π is an answer set of G_P that minimizes the sum of the weights of the violated weak constraints in G_W in a prioritized way.

Syntactic shortcuts. In the following, we also use *choice rules* of the form $\{p\}$, where p is an atom. Choice rules can be viewed as a syntactic shortcut for the rule $p \vee p'$, where p' is a fresh new atom not appearing elsewhere in the program, meaning that the atom p can be chosen as true.

3. Problem Description

Most modern hospitals are characterized by a very long surgical waiting list, often worsened, if not altogether caused, by inefficiencies in operating room planning. In this paper, the elements of the waiting list are called *registrations*. Each registration links a particular surgical procedure, with a predicted duration, to a patient.

The overall goal of the ORS problem is to assign the maximum number of registrations to the operating rooms (ORs). As first requirement, the assignments must guarantee that the sum of the predicted duration of surgeries assigned to a particular OR session does not exceed the length of the session itself: this is referred in the following as *surgery requirement*. Moreover, registrations are not all equal: they can be related to different medical conditions and can be in the waiting list for different periods of time. These two factors can be unified in a singular concept: *priority*. Registrations are classified according to three different priority categories, namely P_1, P_2 and P_3. The first one gathers either very urgent registrations or the ones that have been in the waiting list for a long period of time; it is required that these registrations are all assigned to an OR. Then, the registrations of the other two categories are assigned to the top of the ORs capacity, prioritizing the P_2 over the P_3 ones (*minimization*).

However, in hospital units it is frequent that one planned assignment of ORs cannot be fulfilled due to complications or conflicts that may occur either during the surgery or before. In particular, surgeries may last longer than expected or some patients may delete the registration. Therefore, in such cases it is required to compute a new schedule which reallocates the ORs and, at the same time, minimizes the differences with a previously computed schedule. This problem is usually referred to as *rescheduling*. In our case of study, it often occurs that a registration is delayed and must be reassigned to a later session. The choice of the session where the delayed registration is to be included is done by a health-care operator and is part of the input of our problem. Our framework reacts to this decision and computes a new scheduling for accommodating the preference expressed by the operator, starting from the first day touched by the changes. It is important to emphasize here that such situations are usually independent from the quality of the original schedule, indeed they are often due to unpredictable events.

The ORS problem we deal with entails two subproblems: (i) the computation of an initial schedule for a given planning period (usually one week in hospitals, which is thus our target), and (ii) the rescheduling, i.e., the generation of an altered schedule based on complications or conflicts that require changes in the initial schedule.

The implementation described in Section 5 supports both the generation of an optimized initial schedule of the surgeries and its alteration and rearrangement in case of needed rescheduling, where the case of delayed registrations is considered.

4. Mathematical formulation

In this section we proceed by expressing the ORS problem in a more rigorous mathematical formulation.

The first step is to describe more rigorously the elements we are dealing with. Let

- R be a set of registrations,
- S be a set of specialties,
- K be a set of operating rooms,
- T be the set of sessions in the planning period,
- D be the set of days in the planning period.

An OR time block within the planning horizon is a pair of indices (k, t), $k \in K$ and $t \in T$, representing the OR and the session when the block is scheduled, respectively.

We are ready to define the functions that can help establish the relations between the elements of the ORS problem.

Definition 4.1. (**ORS problem**) Let

- $p : R \to \{1, 2, 3\}$ be a function associating each registration to a priority,
- $\delta : R \to \mathbb{N}$ be a function associating each registration to a duration,
- $\Delta : K \times T \to \mathbb{N}$ be a function associating each OR block to a duration,
- $\sigma : R \times S \to \{0, 1\}$ be a function such that $\sigma(r, s) = 0$ if the registration r is associated to the specialty s, and 1 otherwise,
- $\tau : S \times K \times T \to \{0, 1\}$ be a function such that $\tau(s, k, t) = 0$ if the OR k is reserved to the specialty s during the planning period t, and 1 otherwise. Note that function τ represents a cyclic timetable referred to as the Master Surgical Schedule (MSS). An example of a MSS is reported in Table 1.

Let $x : R \times K \times T \to \{0, 1\}$ be a function such that $x(r, k, t) = 0$ if the registration r is assigned to the (k, t) block, and 1 otherwise. Moreover, for a scheduling x let $A_x = \{(r, k, t) \mid x(r, k, t) = 0\}$ and $R^*_x = \{r \mid x(r, k, t) = 0, r \in R, k \in K, t \in T\}$. Then, given sets R, S, K, T, D, and functions p, δ, Δ, σ, τ, the ORS problem is defined as the problem of finding a scheduling x, such that

(c_1) $|\{(k, t) \mid x(r, k, t) = 0, k \in K, t \in T\}| \leq 1 \quad \forall r \in R$;

(c_2) $\tau(s, k, t) + \sigma(r, s) = 0 \quad \forall\, s \in S, (r, k, t) \in A_x$;

(c_3) $\sum_{r \in R, x(r,k,t)=0} \delta(r) \leq \Delta(k, t) \quad \forall\, k \in K, t \in T$;

(c_4) $\{r \mid r \in R, p(r) = 1\} \subseteq R^*_x$.

Condition (c_1) assures that each registration can be assigned at most once.

Condition (c_2) ensures the respect of the MSS.

Condition (c_3) excludes all cases where the sum of registration durations assigned to an OR block exceeds the duration of the block itself.

Finally, condition (c_4) imposes all priority 1 registrations to be assigned.

A solution ψ is a scheduling x that satisfies (c_1), (c_2), (c_3), and (c_4).

Definition 4.2. (**Unassigned registrations**) Given a scheduling solution ψ, let $R^{pr}_{\psi} = \{r \mid r \in R, p(r) = pr, r \notin R^*_{\psi}\}$. Intuitively, R^{pr}_{ψ} represents the set of registrations of priority pr that were not assigned to any operating room.

Definition 4.3. (**Minimal scheduling solution**) A scheduling solution ψ is said to dominate a scheduling solution ψ' if $|R^2_{\psi}| < |R^2_{\psi'}|$, or if $|R^2_{\psi}| = |R^2_{\psi'}|$ and $|R^3_{\psi}| < |R^3_{\psi'}|$. A scheduling solution is *minimal*, if it is not dominated by any other scheduling solutions.

In the rescheduling problem we start from a subset of a previously calculated schedule for the registrations associated to specialty s^*, interrupted during its implementation at a given session t^*. In particular, we took into account the case where some patients could not be operated in their assigned slot and must be reallocated

Table 1

The MSS schema shows the specialty assigned to each OR and session combination (in this example, numbers from 1 to 5). n_D and n_T represent the total number of days and sessions, respectively.

	Day (d)	1		2		...	n_D
	Session (t)	1	2	3	4	...	n_T
	1	1	1	1	1	1	1
	2	1	1	1	1	1	1
	3	1	1	1	1	1	1
	4	2	2	2	2	2	2
OR	5	2	2	2	2	2	2
(k)	6	3	3	3	3	3	3
	7	3	3	3	3	3	3
	8	4	4	4	4	4	4
	9	5	5	5	5	5	5
	10	5	5	5	5	5	5

in one of the slots in the remaining part of the original planning period. The remaining part of the schedule is then disrupted, for example by introducing other registrations with assignments forced manually by the user, and must be replaced with new assignments. It is important to notice that usually the disruption can be confined to the schedule of the single specialty s^*. This is the case we analyzed in the current work, however the formulas can be easily generalized in case the disruption affects the schedule of several specialties. In order to free time for the reallocated registrations, it may be necessary to exclude some registrations assigned toward the end of the disrupted schedule. Since this is a particularly sensitive decision that can involve many factors besides the ones treated in this work, the choice of the excluded registrations has been left to the user. In order to formalize the rescheduling part of the ORS problem, we need to define some additional functions and sets.

Definition 4.4. (**Rescheduling problem**) Let

- s^* be a specialty in S,
- t^* be a session in T,
- $\gamma : T \to \{0, 1\}$ be a function such that $\gamma(t) = 1$ if $t \leq t^*$ (i.e. the "past" with respect to the disruption), and 0 otherwise,
- $\pi : R \to \{0, 1\}$ be a function such that $\pi(r) = 1$ if the registration r has been removed from the old schedule by the user, and 0 otherwise,
- $\epsilon : T \times D \to \{0, 1\}$ be a function such that $\epsilon(t, d) = 0$ if the session t is planned in the day d, and 1 otherwise,
- $R' = \{r \mid x(r, k, t) = 0, \tau(s^*, k, t) + \sigma(r, s^*) + \gamma(t) = 0, \pi(r) = 0, r \in R, \forall k \in K, t \in T\}$ be the subset of registrations belonging to the old schedule that we need to reschedule. Note that here $x(r, k, t)$ refers to the assignments of the old, disrupted schedule.

Let now $y : R' \times K \times T \to \{0, 1\}$ be a function such that $y(r, k, t) = 0$ if the registration r is assigned to the $(k,\, t)$ block, and 1 otherwise. Then, given t^*, s^*, sets R', K, T, D, and functions δ, Δ, σ, τ, γ, π, ϵ, the ORS rescheduling problem is defined as the problem of finding a scheduling y, such that

(c_5) $|\{(k, t) \mid y(r, k, t) = 0, k \in K, t \in T\}| = 1 \quad \forall r \in R'$;

(c_6) $\tau(s^*, k, t) + \gamma(t) = 0 \quad \forall\, k \in K, t \in T$;

(c_7) $\sum\limits_{r \in R', y(r,k,t)=0} \delta(r) \leq \Delta(k, t) \quad \forall\, k \in K, t \in T$.

Condition (c_5) ensures that each registration to be rescheduled is assigned once. Condition (c_6) assures that the MSS is respected and that only the "future" (i.e. the sessions with $t > t^*$) is rescheduled. Finally, condition (c_7), analogously to (c_3), does not allow the overrun of the (k, t) blocks. A solution ϕ is a scheduling y that satisfies (c_5), (c_6), and (c_7).

Definition 4.5. (**Temporal difference**)
Given a rescheduling solution ϕ, let us define the temporal difference between the assignments of the disrupted schedule and the associated rescheduled assignments as $\Theta_\phi = \sum\limits_{r \in R'} \theta(r)$, where $\theta : R' \to \mathbb{N}$ is a function such that

$$\theta(r) = \sum_{k,k' \in K, t,t' \in T, d,d' \in D} \mid d - d' \mid \big(1 - x(r,k,t)\big)\big(1 - y(r,k',t')\big)\big(1 - \epsilon(t,d)\big)\big(1 - \epsilon(t',d')\big).$$

Definition 4.6. (**Minimal rescheduling solution**) A rescheduling solution ϕ is said to be closer to the disrupted schedule than another solution ϕ' if $\Theta_{\phi} < \Theta_{\phi'}$. A rescheduling solution is *minimal*, if no other rescheduling solution is closer to the disrupted schedule.

5. ASP Encoding

In this section the scheduling and rescheduling problems are described in the ASP language, in particular following the ASP-CORE-2 input language specification [18], in two separate sub-sections.

5.1. OR scheduling

5.1.1. Data Model

The input data is specified by means of the following atoms:

- Instances of *registration(R,P,SU,SP)* represent the registrations, characterized by an id (R), a priority score (P), a surgery duration (SU) and the id of the specialty (SP) it belongs to.
- Instances of *mss(O,S,SP,D)* link each operating room (O) to a session (S) for each specialty and planning day (D) as established by the hospital MSS.
- The OR sessions are represented by the instances of the predicate *duration(N,O,S)*, where N is the session duration.

The output is an assignment represented by atoms of the form *x(R,P,O,S,D)*, where the intuitive meaning is that the registration R with priority P is assigned to the operating room O during the session S and the day D.

5.1.2. Encoding

Following the schema of the mathematical formulation in the previous section, rule (r_1) guesses an assignment for the registrations to an OR in a given day and session among the ones permitted by the MSS for the particular specialty the registration belongs to.

The same registration should not be assigned more than once, in different OR or sessions. This is assured by the constraints (r_2) and (r_3). Note that in our setting there is no requirement that every registration must actually be assigned.

Surgery requirement. With rules (r_4) and (r_5), we impose that the total length of surgery durations assigned to a session is less than or equal to the session duration.

Minimization. We remind that we want to be sure that every registration having priority 1 is assigned, then we assign as much as possible of the others, giving precedence to registrations having priority 2 over those having priority 3. This is accomplished through constraint (r_6) for priority 1 and the weak constraints (r_7) and (r_8) for priority 2 and 3, respectively. Note that in this encoding *totRegsP1*, *totRegsP2* and *totRegsP3* are constants representing the total number of registrations having priority 1, 2 and 3, respectively.

Minimizing the number of unassigned registrations could cause an implicit preference towards the assignments of the registrations with shorter surgery durations. To avoid this effect, one can consider to minimize the idle time, however this is in general slower from a computational point of view and unnecessary, since the shorter surgeries preference is already mitigated by our three-tiered priority schema.

5.2. Rescheduling

We now formulate the rescheduling in ASP.

5.2.1. Data Model

The input data is specified by means of the following atoms:

- The old planning is encoded through facts represented by instances of the predicate *x(R,P,O,S,D)*.
- MSS, registrations and sessions are described by the same predicates as in the previous section.

The output is a new assignment, represented by atoms of the form *y(R,P,O,S,D)*.

5.2.2. Encoding

The new encoding is reported in Figure 2. It basically includes only rules (r_1), (r_2), (r_3), (r_4), and (r_5) from the previous encoding, where atoms over the predicate x are replaced with y, respectively. Additionally, the constraint (r_9) must be added to ensure that for every single registration in the old schedule (x predicate) there is an assignment in the new one (y predicate).

As we have seen, the main objective of the scheduling was to assign the largest possible number of registrations to the OR sessions, while in the rescheduling problem the objective is to reassign all the previ-

$$
\begin{aligned}
\{x(R,P,O,S,D)\} &\text{ :- } \mathit{registration}(R,P,\_,SP), \mathit{mss}(O,S,SP,D). && (r_1)\\
&\text{ :- } x(R,P,O,S1,\_), x(R,P,O,S2,\_), S1 != S2. && (r_2)\\
&\text{ :- } x(R,P,O1,\_,\_), x(R,P,O2,\_,\_), O1 != O2. && (r_3)\\
\mathit{surgery}(R,SU,O,S) &\text{ :- } x(R,\_,O,S,\_), \mathit{registration}(R,\_,SU,\_). && (r_4)\\
&\text{ :- } x(\_,\_,O,S,\_), \#\mathit{sum}\{SU,R : \mathit{surgery}(R,SU,O,S)\} > N, \mathit{duration}(N,O,S). && (r_5)\\
&\text{ :- } N = \mathit{totRegsP1} - \#\mathit{count}\{R : x(R,1,\_,\_,\_)\},\ N > 0. && (r_6)\\
&\ :\sim\ N = \mathit{totRegsP2} - \#\mathit{count}\{R : x(R,2,\_,\_,\_)\}.\ [N@3] && (r_7)\\
&\ :\sim\ N = \mathit{totRegsP3} - \#\mathit{count}\{R : x(R,3,\_,\_,\_)\}.\ [N@2] && (r_8)
\end{aligned}
$$

Fig. 1. ASP encoding of the ORS problem (scheduling)

$$
\begin{aligned}
\{y(R,P,O,S,D)\} &\text{ :- } \mathit{registration}(R,P,\_,SP), \mathit{mss}(O,S,SP,D). && (r'_1)\\
&\text{ :- } y(R,P,O,S1,\_), y(R,P,O,S2,\_), S1 != S2. && (r'_2)\\
&\text{ :- } y(R,P,O1,\_,\_), y(R,P,O2,\_,\_), O1 != O2. && (r'_3)\\
\mathit{surgery}(R,SU,O,S) &\text{ :- } y(R,\_,O,S,\_), \mathit{registration}(R,\_,SU,\_). && (r'_4)\\
&\text{ :- } y(\_,\_,O,S,\_), \#\mathit{sum}\{SU,R : \mathit{surgery}(R,SU,O,S)\} > N, \mathit{duration}(N,O,S). && (r'_5)\\
&\text{ :- } \mathit{not}\ y(R,P,\_,\_,\_),\ x(R,P,\_,\_,\_). && (r_9)\\
\mathit{difference}(DF,R) &\text{ :- } y(R,\_,\_,\_,D),\ x(R,\_,\_,\_,\mathit{OldD}),\ DF = |D - \mathit{OldD}|. && (r_{10})\\
&\ :\sim\ T = \#\mathit{sum}\{DF,R :\ \mathit{difference}(DF,R)\}.\ [T@1] && (r_{11})
\end{aligned}
$$

Fig. 2. ASP encoding of the ORS problem (rescheduling)

ously allocated registrations and the reallocated ones with the least possible disruption to the old schedule. The computation and minimization of the difference in days between the new and old assignments for each registration is done by replacing rules (r_6), (r_7) and (r_8) by the rules (r_{10}) and (r_{11}).

6. Experimental Results

In this section we report about the results of an empirical analysis of the scheduling and rescheduling problems. For the initial scheduling problem, data have been randomly generated but having parameters and sizes inspired by real data, then a part of the results of the planning has been used as input for the rescheduling (as we will detail later). Both experiments were run on a Intel Core i7-7500U CPU @ 2.70GHz with 7.6 GB of physical RAM. The ASP system used was CLINGO [31], version 5.5.2.

6.1. ORS

The test cases we have assembled for the initial planning is based on the requirements of a typical middle sized hospital, with five surgical specialties to be managed. To test scalability, other than the 5-days planning period, which is the one that is widely used in Italian hospital units, seven benchmarks of different dimension were created. Each benchmark was tested 10 times with different randomly generated input. The characteristics of the tests are the following:

– 7 different benchmarks, comprising a planning period of 15, 10, 7, 5, 3, 2 and 1 work days, respectively;
– 10 ORs (that can represent a hospital of small-medium size), unevenly distributed among the specialties;
– 5 hours long morning and afternoon sessions for each operating room, summing up to a total of respectively 1500, 1000, 700, 500, 300, 200 and

Table 2
Parameters for the random generation of the scheduler input

Specialty	Registrations							ORs	Avg. Surgery	Coefficient
	15-day	10-day	7-day	5-day	3-day	2-day	1-day		Duration (min)	of Variation
1	240	160	112	80	48	32	16	3	124	48%
2	210	140	98	70	42	28	14	2	99	18%
3	210	140	98	70	42	28	14	2	134	19%
4	180	120	84	60	36	24	12	1	95	21%
5	210	140	98	70	42	28	14	2	105	29%
Total	1050	700	490	350	210	140	70	10		

100 hours of ORs available time for the 7 benchmarks;

- for each benchmark, we generated 1050, 700, 490, 350, 210, 140 and 70 registrations, respectively, from which the scheduler will draw the assignments. Registrations are characterized by a surgery duration, a specialty and a priority. In this way, we simulate the common situation where a hospital manager takes an ordered, w.r.t. priorities, waiting list and tries to assign as many elements as possible to each OR.

The surgery durations have been generated assuming a normal distribution, while the priorities have been generated from a quasi-uniform distribution of three possible values (with weights respectively of 0.30, 0.33 and 0.37 for registrations having priority 1, 2 and 3, respectively). The parameters of the test have been summed up in Table 2. In particular, for each specialty (1 to 5), we reported the number of registrations generated for each benchmark (15-, 10-, 7-, 5-, 3-, 2- and 1-day), the number of ORs assigned to the specialty, the average duration of surgeries, and the coefficient of variation (defined as the standard deviation over the mean), respectively.

Results of the experiment are reported in Table 3, as the average of 10 runs for each benchmark. Table 3 reports, for each benchmark, the average number of assigned registrations (shown as assigned/generated ratio). The efficiency column shows the percentage of the total OR time occupied by the assigned registrations. A time limit of 20 seconds was given in view of a practical use of the program: on the target 5-days planning length, an efficiency of the 95% was reached. As a general observation, we report that with all the considered benchmarks, except with the one having planning length of 15-day, we obtained an efficiency greater than or equal to 90%. The 1-day test managed to converge after around 10 seconds.

A more detailed analysis of the performance is reported in Table 4 for the target 5-day planning period. In particular, for each of the 10 runs executed, Table 4 reports the number of the assigned registrations out of the generated ones for each priority, and a measure of the total time occupied by the assigned registrations as a percentage of the total OR time available. In this case, it is possible to observe that the efficiency is always greater or equal to 95%, but for an instance having efficiency of 92%.

Finally, in 5 plots of Figure 3 we (partially) present the results achieved on one instance (i.e., the first instance of Table 4) with 350 registrations for 5 days. Each colored block in the respective plots corresponds to a registration assigned to one of the 10 ORs. The remaining space up to the 300 minutes limit represents the idle time of the OR. Only the data about the morning assignments are showed: the ones for the afternoon are (qualitatively) similar. The bottom-right plot shows, instead, the evolution of the solution quality when 600 seconds are granted to the same instance: we can notice that after around 10 seconds the number of assigned registrations does not change significantly.

6.2. *Rescheduling*

The rescheduling is applied to a previously planned schedule in the case this could not be carried on to the end. Once planned, a specialty schedule does not normally influence the other specialties, thus it makes sense to re-schedule one specialty at a time.

To test the rescheduler we have defined three different scenarios. Considering the target planning schedule of 5-day, we assumed that in the second day a number of surgeries in specialty 1 had to be postponed to the next day. This number was set to 1 (scenario A), 3 (scenario B) or 6 (scenario C), respectively. Thus, we have to re-schedule the three remaining days of the planning.

Table 3
Averages of the results for the 15, 10, 7, 5, 3, 2 and 1-day benchmarks

Benchmark	Priority 1	Priority 2	Priority 3	Total	Efficiency
15 days	319 / 319	169 / 342	42 / 389	530 / 1050	66%
10 days	210 / 210	201 / 229	81 / 261	492 / 700	90%
7 days	147 / 147	152 / 166	55 / 177	353 / 490	92%
5 days	106 / 106	102 / 113	50 / 130	258 / 350	95%
3 days	62 / 62	62 / 67	35 / 81	159 / 210	94%
2 days	42 / 42	40 / 46	22 / 52	104 / 140	95%
1 day	21 / 21	20 / 23	12 / 26	53 / 70	96%

Table 4
Scheduling results for the 5-day benchmark

Assigned Registrations				OR time Efficiency
Priority 1	Priority 2	Priority 3	Total	
103 out of 103	104 out of 121	61 out of 126	268 out of 350	96%
114 out of 114	90 out of 94	54 out of 142	258 out of 350	95%
102 out of 102	116 out of 116	43 out of 123	261 out of 350	95%
112 out of 112	90 out of 102	50 out of 136	252 out of 350	95%
103 out of 103	95 out of 107	35 out of 140	233 out of 350	92%
99 out of 99	99 out of 122	66 out of 129	264 out of 350	95%
101 out of 101	108 out of 110	44 out of 139	253 out of 350	95%
114 out of 114	115 out of 124	41 out of 112	270 out of 350	96%
114 out of 114	114 out of 129	34 out of 107	262 out of 350	96%
98 out of 98	91 out of 108	73 out of 144	262 out of 350	95%

In order to be able to insert the postponed registrations, we have to make sure that the starting schedule leaves enough available OR time by removing the necessary registrations from the old schedule, beginning from the last day of the period and from the registrations in the priority 3 category. In an actual hospital, this action would be performed by a health-care operator according to criteria that may vary from the one we chose to follow.

The three tests performed had the following characteristics: *(i)* in all scenarios the postponed registrations have been generated with an average surgery duration of 100 minutes, *(ii)* the postponed registrations were manually inserted in the first slot available (i.e. the morning of the third day) of one (scenarios A and B) or two (scenario C) ORs, *(iii)* the number of registrations present in the old schedule is 43, *(iiii)* 0, 1 and 4 priority 3 old schedule registrations had to be removed from the last planning day in scenarios A, B and C, respectively.

The results are summarized in Table 5, where we report the scenario, the number of registrations that were inserted in each scenario (Postponed Registrations), the number of registrations coming from the old schedule (Old Registrations), and the total displacement, calculated as shown in rule (r_{12}), showing the sum of all day displacements the old registrations were subject to in the resulting new schedule.

Table 5
Results for the three rescheduling scenarios

Scenario	Postponed Registrations	Old Registrations	Total Displacements (Days)
A	1	43	3
B	3	42	4
C	6	39	6

7. Application of our ASP solution

Our ASP solution presented in this paper is part of a more general real-life application that we are developing. The application can be accessed through a web-interface where the parameters of the problem can be

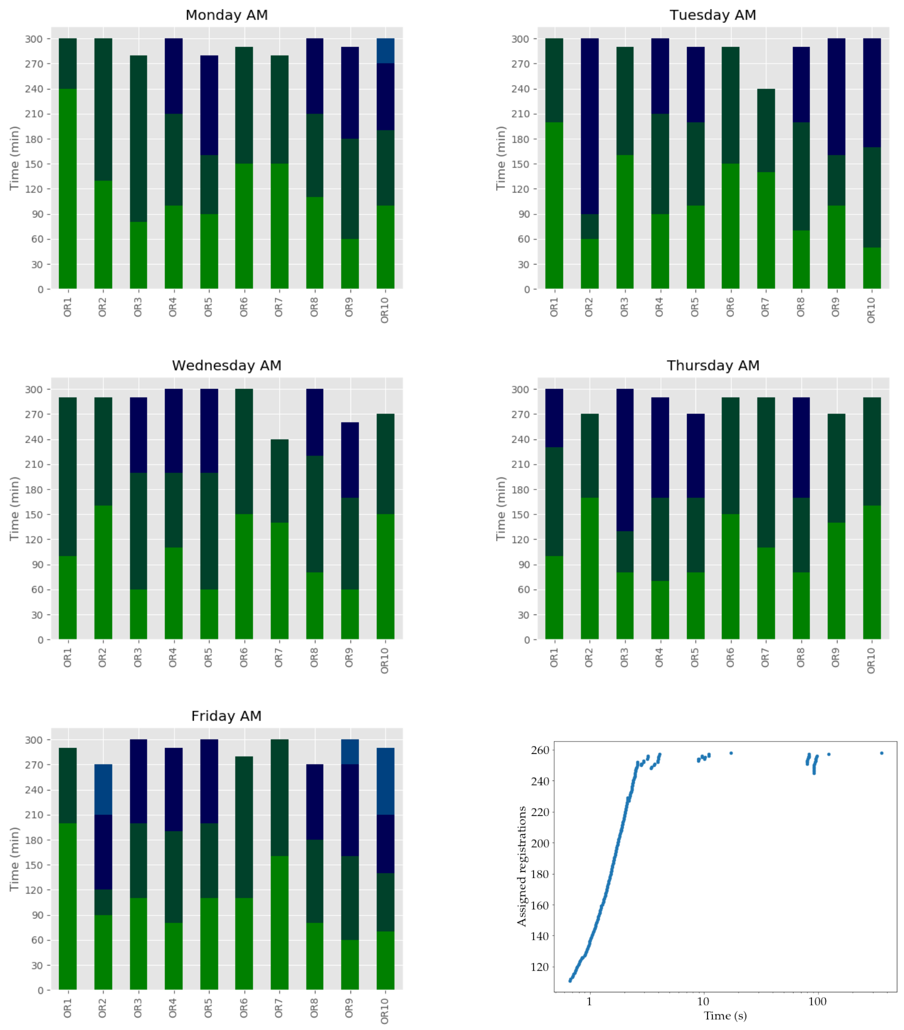


Fig. 3. Example of scheduling with 350 registrations for 5 days, and time scale (bottom-right)

specified (see Subsection 7.1). Moreover, the interface allows the user to interact with the ASP encoding by offering web forms for adding the so called "customizable constraints", that express user requirements and preferences.

7.1. *Web application*

We have wrapped the ASP encoding and the CLINGO solver inside a Node.js architecture and, at the same time, we have created a graphical user interface (GUI) for it. This solution allows the solver to be embedded inside an easily reachable and usable web application, removing the hurdle that installing and managing the solver may represent for a non-expert user. The application includes:

– a registration and authentication process,
– a database for storing and retrieving previous test data, and
– a GUI to easily create and customize new test scenarios or load pre-made ones.

While we are still at an early stage of development, the user can already freely set the parameters for the generator through an Input screen (see Figure 4). In this picture two tables are shown: the right side one is used to manage the bed usage in the wards after the surgery through encodings not included in this work.

In the left table the user can set the parameters for the generator. From left to right these are:

– the specialty names;
– the number of registrations we aim to assign for each specialty;
– the parameters (mean and coefficient of variation) of the Gaussian distribution used to generate the predicted length of stay (LOS) in the ward after the surgery, necessary for the bed management part;
– the parameters (mean and coefficient of variation) of the Gaussian distribution used to generate the predicted surgery durations;
– the ratio of patients predicted to need a place in the intensive care (IC) ward;
– the parameters (mean and coefficient of variation) of the Gaussian distribution used to generate the predicted LOS in the IC ward after the surgery, necessary for the bed management part.

We stress again that our paper focuses on the case where issues related to beds management are not taken into account.

We have, then, a Results screen (see Figure 5) where the user can monitor in real-time the evolution of the process and, finally, read the final results.

– At the top of the screen there are three cards containing the number of assigned registrations out of the total, arranged according to their priority class, for each solution found by the solver engine. Each number is continuously updated during the execution whenever a new solution is found. The percentage of assigned registrations is represented by a progress bar at the bottom of each card.
– At the bottom we summarize the final results at the end of the execution. In particular the OR time out of the total available is reported, both in numbers and as a percentage through a progress bar.

The last screen we want to show (see Figure 6) contains a graphical representation of each OR occupancy for each session of the planning period, through a carousel of stacked bar graphs.

7.2. Customizable constraints

The customizable constraints are not strictly required for the correctness of the program but allow the users to guide the final results as they prefer. We have identified different constraints that combined together cover most user needs; note that such customizable constraints can be used for both scheduling and rescheduling. Each of these constraints can be activated at runtime for multiple registrations and can involve different selection of days, ORs, and sessions. The set of all constraints that can be added is listed in Figure 7.

In particular, given a set of n registrations, defined by the user and characterized by the ids $reg_i, i = 1, .., n$, constraints (r_{12}) and (r_{13}) impose that such registrations can be assigned only in a chosen period, defined as all the operating room sessions between the initial (*init*) and the final (*fin*) days, where i and f are parameters provided by the user.

Constraint (r_{14}) can be used to forbid a specific session s to the chosen registrations.

Constraints (r_{15}) and (r_{16}) allow the user to forbid or enforce the use of a specific OR o for a set of registrations, respectively.

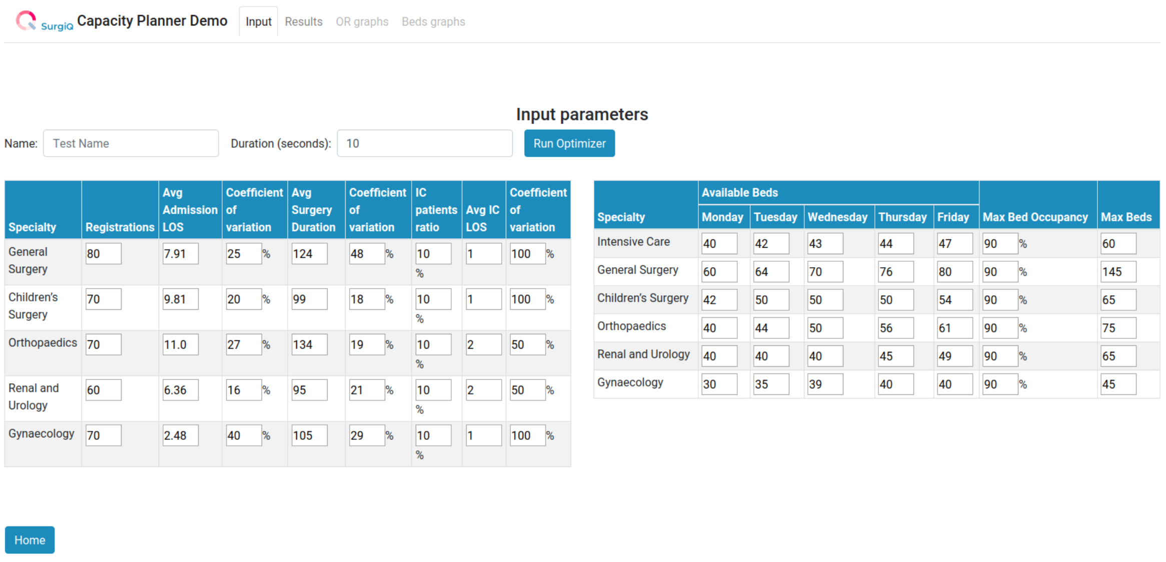
Finally, rules (r_{17}) and (r_{18}) can be used if the user wants to assign a set of registrations as temporally close as possible to a specific OR session, without actually enforcing it. This can be accomplished by defining a predicate (*distance(N,R)*) that computes the *distance* (N) between the assigned (S) and suggested (represented by the parameter $prefS$) sessions and tries to minimize it.

All these rules can be applied to different sets of registrations at the same time, using different parameters.

8. Related Work

A preliminary version of this paper was presented at the 17th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2018) [24]. Compared to such conference version, this paper adds (i) a mathematical formalization of the ORS problem, which is independent from the employed solving methodology, (ii) a description of a graphical user interface for producing a real-time scheduling of operating rooms, and (iii) a more detailed and improved related work.

We are not aware of any previous attempt to solve the ORS problem using ASP algorithms (other than the mentioned previous version [24]); however, an extensive literature approaching this problem with different techniques has been developed.



Input parameters

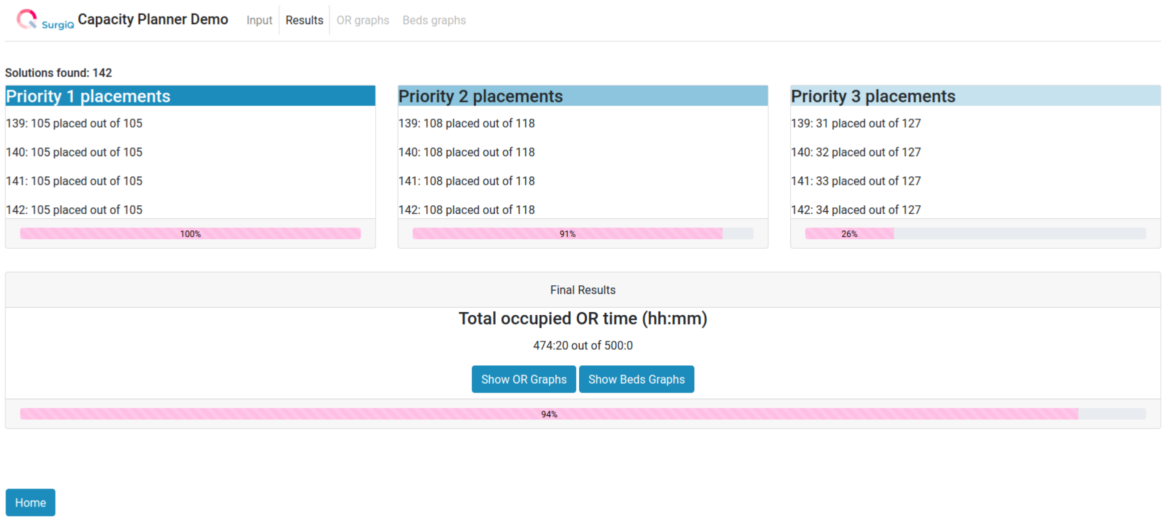
Name: Test Name Duration (seconds): 10 Run Optimizer

Specialty	Registrations	Avg Admission LOS	Coefficient of variation	Avg Surgery Duration	Coefficient of variation	IC patients ratio	Avg IC LOS	Coefficient of variation
General Surgery	80	7.91	25 %	124	48 %	10 %	1	100 %
Children's Surgery	70	9.81	20 %	99	18 %	10 %	1	100 %
Orthopaedics	70	11.0	27 %	134	19 %	10 %	2	50 %
Renal and Urology	60	6.36	16 %	95	21 %	10 %	2	50 %
Gynaecology	70	2.48	40 %	105	29 %	10 %	1	100 %

Specialty	Available Beds					Max Bed Occupancy	Max Beds
	Monday	Tuesday	Wednesday	Thursday	Friday		
Intensive Care	40	42	43	44	47	90 %	60
General Surgery	60	64	70	76	80	90 %	145
Children's Surgery	42	50	50	50	54	90 %	65
Orthopaedics	40	44	50	56	61	90 %	75
Renal and Urology	40	40	40	45	49	90 %	65
Gynaecology	30	35	39	40	40	90 %	45

Home

Fig. 4. Input screen for the registration generator parameters.



Solutions found: 142

Priority 1 placements	Priority 2 placements	Priority 3 placements
139: 105 placed out of 105	139: 108 placed out of 118	139: 31 placed out of 127
140: 105 placed out of 105	140: 108 placed out of 118	140: 32 placed out of 127
141: 105 placed out of 105	141: 108 placed out of 118	141: 33 placed out of 127
142: 105 placed out of 105	142: 108 placed out of 118	142: 34 placed out of 127
100%	91%	26%

Final Results

Total occupied OR time (hh:mm)

474:20 out of 500:0

Show OR Graphs Show Beds Graphs

94%

Home

Fig. 5. Results screen.

Solving ORS problems. Aringhieri et al. [13] addressed the joint OR planning (MSS) and scheduling problem, described as the allocation of OR time blocks to specialties together with the subsets of patients to be scheduled within each time block over a one week planning horizon. They developed a 0-1 linear programming formulation of the problem and used a two-level meta-heuristic to solve it. Its effectiveness was demonstrated through extensive numerical experiments carried out on a set of instances based on real data and resulted, for benchmarks of 80-100 assigned registrations, in a 95-98% average OR utilization rate, for a number of ORs ranging from 4 to 8. The execution times were around 30-40 seconds. In [40], the same authors introduced a hybrid two-phase optimization algorithm which exploits neighborhood search techniques combined with Monte Carlo simulation, in order to solve the joint advance and allocation scheduling problem, taking into account the inherent uncertainty of surgery durations. Abedini et al. [1] devel-

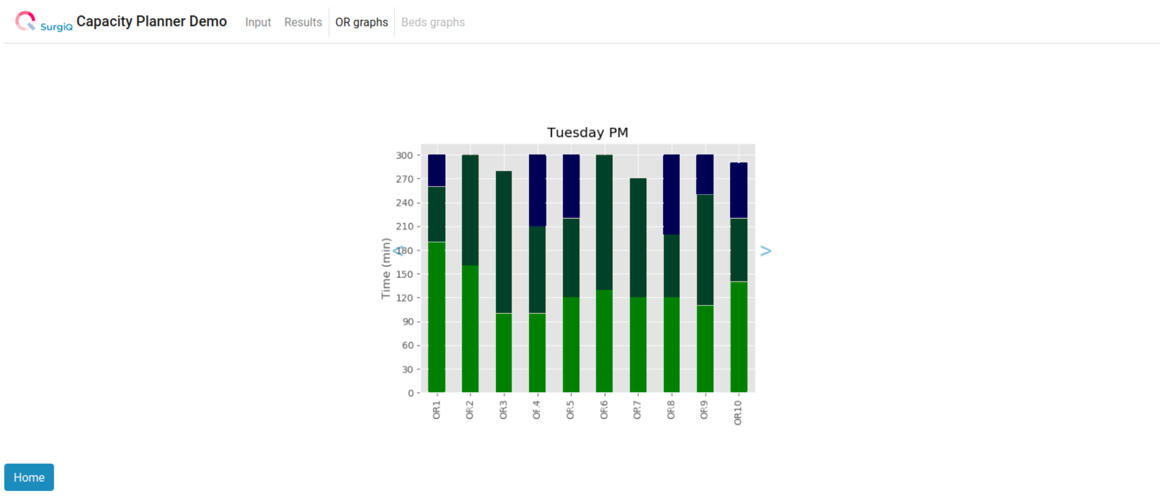


Fig. 6. Graphical representation of the results.

$$:-\ x(reg_i,\_,\_,S,\_),\ S < init. \qquad (r_{12})$$

$$:-\ x(reg_i,\_,\_,S,\_),\ S > fin. \qquad (r_{13})$$

$$:-\ x(reg_i,\_,\_,s,\_). \qquad (r_{14})$$

$$:-\ x(reg_i,\_,o,\_,\_), registration(reg_i,\_,\_,\_), mss(o,\_,\_,\_). \qquad (r_{15})$$

$$:-\ not\ x(reg_i,\_,o,\_,\_), registration(reg_i,\_,\_,\_), mss(o,\_,\_,\_). \qquad (r_{16})$$

$$distance(N,reg_i) :-\ x(reg_i,\_,\_,S,\_),\ N = |S - prefS|. \qquad (r_{17})$$

$$:\sim\ T = \#sum\{N,R:\ distance(N,R)\}.\ [T@4] \qquad (r_{18})$$

Fig. 7. Customizable constraints

oped a bin packing model with a multi-step approach and a priority-type-duration rule. The model maximizes utilization and minimizes the idle time, which consequently affects the cost at the planning phase and was programmed using MATLAB. Five hundred elective surgeries were generated for a week and a 90% average OR utilization rate was reached. Molina-Pariente et al. [43] tackled the problem of assigning an intervention date and an operating room to a set of surgeries on the waiting list, minimizing access time for patients with diverse clinical priority values. The algorithms used to allocate surgeries were various bin packing operators. They adapted existing heuristics to the problem and compared them to their own heuristics. The tests were performed with the software Gurobi. The authors used four, two and one week planning horizons, with benchmarks of 50 to 182 registrations and 3 or 9 ORs. They reached an efficiency between 87% and 91%.

The rescheduling problem was addressed by Shu et al. [47], using an extension of the Longest Processing Time algorithm, which was used to solve the atomic job shop scheduling problem. Zhang et al. [48] addressed the problem of OR planning with different demands from both elective patients and non-elective ones, with priorities in accordance with urgency levels and waiting times. This problem is formulated as a penalty stochastic shortest-path Markov Decision Process with dead ends, and solved using MATLAB by the method of asynchronous value iteration.

ASP in scheduling problems. We already mentioned in the introduction that ASP has been already successfully used for solving hard combinatorial and application problems in several research areas. Concerning scheduling problems other than ORS, ASP encodings were proposed for the following problems: *Incremental Scheduling Problem* [19], where the goal is to assign jobs to devices such that their executions do not overlap one another; *Team Building Problem* [45], where the goal is to allocate the available personnel of a seaport for serving the incoming ships; *Nurse Scheduling Problem* [5,6,21], where the goal is to create a scheduling for nurses working in hospital units, and also *Interdependent Scheduling Games* [10], *Conference Paper Assignment Problem* [11], and *Stable Roommates Problem* [9].

GUI to support ASP at work. In the recent years, several tools have been proposed for simplifying the design of ASP applications. Among them, Integrated Development Environments (IDEs), such as ASPIDE [27] and SEALION [17], support the user during the life-cycle of development by combining a collection of tools for developing, testing and debugging ASP programs. A web-based programming environment for the IDP system is presented in [20], which also features supporting tools for developing and debugging logic programs. In [45], the *Team Building Problem* was solved by means of an ASP encoding and a java-based GUI was presented for the interaction with users. Other GUIs were presented for *E-Tourism* [44] and *E-Learning* [28,39].

9. Conclusions

In this paper we presented an ASP encoding to provide a solution to the ORS problem, where specifications of the problem are modularly expressed as ASP rules. Then, we also presented techniques for rescheduling on-line in case the off-line solution can not be fully applied given, e.g., canceled registrations. In this case, the goal is to minimize the changes needed to accommodate the new situation. Again, the rescheduling is specified by modularly adding ASP rules to (part of) the (updated) original ASP program. Finally, we presented the results of an experimental analysis on ORS benchmarks with realistic sizes and parameters showing that our scheduling solution obtains around 95% of efficiency after few seconds of computation on planning length of 5 days usually used in Italian hospitals. Our solution also enjoys good scalability property, having an efficiency over or equal to 90% for planning periods up to 10 days, i.e., double w.r.t. the target period. Also our rescheduling solution reached positive results. Future work includes the analysis of preference-based heuristics (see, e.g. [7,36,37,38,46]) to further improve performance and scalability.
All benchmarks and encodings employed in this work can be found at: `http://www.star.dist.unige.it/~marco/AIIA2018/material.zip`, while our web application can be reached out at `http://aidemo.surgiq.com`.

Acknowledgments. We thank the AI*IA 2018 and Intelligenza Artificiale reviewers for useful and constructive comments. The research described in this paper was partially funded by the POR FESR Liguria 2014-2020 funding programme.

References

[1] Amin Abedini, Honghan Ye, and Wei Li. Operating Room Planning under Surgery Type and Priority Constraints. *Procedia Manufacturing*, 5:15–25, 2016.

[2] Michael Abseher, Martin Gebser, Nysret Musliu, Torsten Schaub, and Stefan Woltran. Shift design with answer set programming. *Fundamenta Informaticae*, 147(1):1–25, 2016.

[3] Weronika T. Adrian, Marco Manna, Nicola Leone, Giovanni Amendola, and Marek Adrian. Entity set expansion from the web via ASP. In *ICLP (Technical Communications)*, volume 58 of *OASICS*, pages 1:1–1:5. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[4] Mario Alviano and Carmine Dodaro. Anytime answer set optimization via unsatisfiable core shrinking. *Theory and Practice of Logic Programming*, 16(5-6):533–551, 2016.

[5] Mario Alviano, Carmine Dodaro, and Marco Maratea. An advanced answer set programming encoding for nurse scheduling. In *AI*IA*, volume 10640 of *LNCS*, pages 468–482. Springer, 2017.

[6] Mario Alviano, Carmine Dodaro, and Marco Maratea. Nurse (re)scheduling via answer set programming. *Intelligenza Artificiale*, 12(2):109–124, 2018.

[7] Mario Alviano, Javier Romero, and Torsten Schaub. Preference relations by approximation. In Michael Thielscher, Francesca Toni, and Frank Wolter, editors, *KR*, pages 2–11. AAAI Press, 2018.

[8] Mario Alviano, Giovanni Amendola, Carmine Dodaro, Nicola Leone, Marco Maratea, and Francesco Ricca. Evaluation of disjunctive programs in WASP. In *LPNMR*, volume 11481 of *Lecture Notes in Computer Science*, pages 241–255. Springer, 2019.

[9] Giovanni Amendola. Solving the stable roommates problem using incoherent answer set programs. In *RiCeRcA@AI*IA*, volume 2272 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.

[10] Giovanni Amendola. Preliminary results on modeling interdependent scheduling games via answer set programming. In *RiCeRcA@AI*IA*, volume 2272 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.

[11] Giovanni Amendola, Carmine Dodaro, Nicola Leone, and Francesco Ricca. On the application of answer set programming to the conference paper assignment problem. In *AI*IA*, volume 10037 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2016.

[12] Giovanni Amendola, Gianluigi Greco, Nicola Leone, and Pierfrancesco Veltri. Modeling and reasoning about NTU games via answer set programming. In *IJCAI*, pages 38–45. IJCAI/AAAI Press, 2016.

[13] Roberto Aringhieri, Paolo Landa, Patrick Soriano, Elena Tànfani, and Angela Testi. A two level metaheuristic for the operating room scheduling and assignment problem. *Computers & Operations Research*, 54:21–34, 2015.

[14] Marcello Balduccini, Michael Gelfond, Richard Watson, and Monica Nogueira. The USA-Advisor: A case study in answer set planning. In *LPNMR*, volume 2173 of *LNCS*, pages 439–442. Springer, 2001.

[15] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103, 2011.

[16] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000.

[17] Paula-Andra Busoniu, Johannes Oetsch, Jörg Pührer, Peter Skocovsky, and Hans Tompits. Sealion: An eclipse-based IDE for answer-set programming with advanced debugging support. *Theory and Practice of Logic Programming*, 13(4-5): 657–673, 2013. doi: 10.1017/S1471068413000410.

[18] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. ASP-Core-2 Input Language Format, 2013. URL `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf`.

[19] Francesco Calimeri, Martin Gebser, Marco Maratea, and Francesco Ricca. Design and results of the Fifth Answer Set Programming Competition. *Artificial Intelligence*, 231:151–181, 2016.

[20] Ingmar Dasseville and Gerda Janssens. A web-based IDE for IDP. *CoRR*, abs/1511.00920, 2015. URL `http://arxiv.org/abs/1511.00920`.

[21] Carmine Dodaro and Marco Maratea. Nurse scheduling via answer set programming. In *LPNMR*, volume 10377 of *LNCS*, pages 301–307. Springer, 2017.

[22] Carmine Dodaro, Nicola Leone, Barbara Nardi, and Francesco Ricca. Allotment problem in travel industry: A solution based on ASP. In *RR*, volume 9209 of *LNCS*, pages 77–92. Springer, 2015.

[23] Carmine Dodaro, Philip Gasteiger, Nicola Leone, Benjamin Musitsch, Francesco Ricca, and Konstantin Schekotihin. Combining answer set programming and domain heuristics for solving hard industrial problems (application paper). *Theory and Practice of Logic Programming*, 16(5-6):653–669, 2016.

[24] Carmine Dodaro, Giuseppe Galatà, Marco Maratea, and Ivan Porro. Operating room scheduling via answer set programming. In *AI*IA*, volume 11298 of *LNCS*, pages 445–459. Springer, 2018.

[25] Esra Erdem and Umut Öztok. Generating explanations for biomedical queries. *Theory and Practice of Logic Programming*, 15(1):35–78, 2015.

[26] Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011.

[27] Onofrio Febbraro, Kristian Reale, and Francesco Ricca. ASPIDE: integrated development environment for answer set programming. In *LPNMR*, volume 6645 of *LNCS*, pages 317–330. Springer, 2011.

[28] Alfredo Garro, Luigi Palopoli, and Francesco Ricca. Exploiting agents in e-learning and skills management context. *AI Communications*, 19(2):137–154, 2006.

[29] Marco Gavanelli, Maddalena Nonato, and Andrea Peano. An ASP approach for the valves positioning optimization in a water distribution system. *Journal of Logic and Computation*, 25 (6):1351–1369, 2015.

[30] Martin Gebser, Marco Maratea, and Francesco Ricca. The Design of the Sixth Answer Set Programming Competition. In *LPNMR*, volume 9345 of *LNCS*, pages 531–544. Springer, 2015.

[31] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory solving made easy with clingo 5. In *ICLP (Technical Communications)*, volume 52 of *OASICS*, pages 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[32] Martin Gebser, Marco Maratea, and Francesco Ricca. What's Hot in the Answer Set Programming Competition. In *AAAI*, pages 4327–4329. AAAI Press, 2016.

[33] Martin Gebser, Marco Maratea, and Francesco Ricca. The design of the seventh answer set programming competition. In Marcello Balduccini and Tomi Janhunen, editors, *LPNMR*, volume 10377 of *Lecture Notes in Computer Science*, pages 3–9. Springer, 2017.

[34] Martin Gebser, Marco Maratea, and Francesco Ricca. The sixth answer set programming competition. *Journal of Artificial Intelligence Research*, 60:41–95, 2017.

[35] Martin Gebser, Nicola Leone, Marco Maratea, Simona Perri, Francesco Ricca, and Torsten Schaub. Evaluation techniques and systems for answer set programming: a survey. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, pages 5450–5456. ijcai.org, 2018.

[36] Enrico Giunchiglia and Marco Maratea. On the relation among answer set solvers. *Ann. Math. Artif. Intell.*, 53(1-4):169–204, 2008.

[37] Enrico Giunchiglia, Marco Maratea, and Armando Tacchella. Dependent and independent variables in propositional satisfiability. In Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, editors, *JELIA*, volume 2424 of *Lecture Notes in Computer Science*, pages 296–307. Springer, 2002.

[38] Enrico Giunchiglia, Marco Maratea, and Armando Tacchella. (in)effectiveness of look-ahead techniques in a modern SAT solver. In Francesca Rossi, editor, *CP*, volume 2833 of *Lecture Notes in Computer Science*, pages 842–846. Springer, 2003.

[39] Giovambattista Ianni, Claudio Panetta, and Francesco Ricca. Specification of assessment-test criteria through ASP specifications. In *Answer Set Programming*, volume 142 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[40] Paolo Landa, Roberto Aringhieri, Patrick Soriano, Elena TÃănfani, and Angela Testi. A hybrid optimization algorithm for surgeries scheduling. *Operations Research for Health Care*, 8: 103–114, 2016.

[41] Yuliya Lierler, Marco Maratea, and Francesco Ricca. Systems, engineering environments, and competitions. *AI Magazine*, 37 (3):45–52, 2016.

[42] Marco Maratea, Luca Pulina, and Francesco Ricca. A multi-engine approach to answer-set programming. *Theory and Practice of Logic Programming*, 14(6):841–868, 2014.

[43] Jose M. Molina-Pariente, Erwin W. Hans, Jose M. Framinan, and Tomas Gomez-Cia. New heuristics for planning operating rooms. *Computers & Industrial Engineering*, 90:429–443, 2015.

[44] Francesco Ricca, Antonella Dimasi, Giovanni Grasso, Salvatore Maria Ielpa, Salvatore Iiritano, Marco Manna, and Nicola Leone. A logic-based system for e-tourism. *Fundamenta Informaticae*, 105(1-2):35–55, 2010.

[45] Francesco Ricca, Giovanni Grasso, Mario Alviano, Marco Manna, Vincenzino Lio, Salvatore Iiritano, and Nicola Leone. Team-building with answer set programming in the Gioia-Tauro seaport. *Theory and Practice of Logic Programming*, 12 (3):361–381, 2012.

[46] Emanuele Di Rosa, Enrico Giunchiglia, and Marco Maratea. A new approach for solving satisfiability problems with qualitative preferences. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *ECAI*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 510–514. IOS Press, 2008.

[47] Albert C. Shu, Indu Subbaraj, and Linh Phan. Operating Room Rescheduler. 2015.

[48] Jian Zhang, Mahjoub Dridi, and Abdellah El Moudni. A stochastic shortest-path MDP model with dead ends for operating rooms planning. In *ICAC*, pages 1–6. IEEE, 2017.