Operating Room Scheduling via Answer Set Programming

Carmine Dodaro[1], Giuseppe Galatà[2], Marco Maratea[1], and Ivan Porro[2]

[1] DIBRIS, University of Genova, Genova, Italy
{dodaro,marco}@dibris.unige.it,
[2] SurgiQ srl, Genova, Italy
{name.surname}@surgiq.com

Abstract. The Operating Room Scheduling (ORS) problem is the task of assigning patients to operating rooms, taking in account different specialties, the surgery and operating room shift 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.

1 Introduction

The Operating Room Scheduling (ORS) [1, 5, 18, 20] problem is the task of assigning patients to operating rooms, taking in account different specialties, surgery durations, and operating room shift 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 is 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 [9], which includes optimization statements as well as powerful database-inspired constructs such as aggregates, and its intuitive semantics, combined with the readability of specifications (always appreciated by users) and availability of efficient solvers (see, e.g., [3, 17, 19]), make ASP an ideal candidate for addressing such problems. Indeed, ASP has been already successfully used for solving hard combinatorial and application problems in several research areas, including Artificial Intelligence [6, 12], Bioinformatics [14], Hydroinformatics [16], and also employed in industrial applications (see, e.g., [2, 11]).

In this paper we first present an off-line solution schedule based on ASP for solving the ORS problem, where problem's 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 original ASP program. We have finally 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. Additionally, 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 in terms of room's occupation, can be achieved in short timings in line with the need of the application.

2 Background on ASP

Answer Set Programming (ASP) [7] 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 [7, 9]. 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 $p(t_1, \ldots, t_n)$, where p is a *predicate* of arity n and $t_1, \ldots, t_n$ are terms. An atom $p(t_1, \ldots, t_n)$ is ground if $t_1, \ldots, t_n$ 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, $\prec \in \{<, \leq, >, \geq\}$ 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:

$$a_1 \lor \ldots \lor a_n \coloneq b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m.$$

where $a_1, \ldots, a_n$ are standard atoms, $b_1, \ldots, b_k$ are atoms, $b_{k+1}, \ldots, b_m$ are standard atoms, and $n, k, m \geq 0$. A literal is either a standard atom a or its negation $not\ \ a$. The disjunction $a_1 \lor \ldots \lor a_n$ is the *head* of r, while the conjunction $b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m$ 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. 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; *(ii)* each local variable of r appearing in a symbolic set $\{Terms : Conj\}$ also appears in $Conj$.

A *weak constraint* [8] 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. (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 the rules of a program. Given a ground program G_P and an interpretation I, the *reduct* [15] 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) [15].

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* O 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.

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 shift does not exceed the length of the shift itself, this is referred in the following as *surgery requirement*. Moreover, registrations are not all equal: they can be related to different pathologies and they 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; we require 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 previous computed schedule. This problem is usually referred to as *rescheduling*. 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 can be split into two subproblems: (i) 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 4 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 canceled registrations is considered.

4 ASP Encoding

In this section the scheduling and rescheduling problems are described in the Answer Set Programming language, in particular following the ASP-CORE-2 input language specification [9], in two separate sub-sections.

4.1 OR scheduling

Data Model. The input data is specified by means of the predicates described in this paragraph. The predicates representing the facts of our encoding are the following:

- 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 shift (S) for each specialty and planning day (D) as established by the hospital Master Surgical Schedule (MSS), i.e., a cyclic timetable constructed to define the specific assignment of OR shifts to specialties. Note that every value of the variable S represents an unique period of time, called shift, and that each day contains two shifts in our choice of MSS. Thus, $S = 1$ denotes the Monday ($D = 1$) AM shift, $S = 2$ the Monday ($D = 1$) PM one, $S = 3$ the Tuesday ($D = 2$) AM one and so on.
- The OR shifts are represented by the instances of the predicate *duration(N,O,S)*, where N is the shift duration.
- Assignments are stored in the instances of *x(R,P,O,S,D)*, representing that the registration R with priority P is assigned to the operating room O during the shift S and the day D.

Encoding. Here we describe the ASP rules used for solving the ORS problem. The encoding is based on the well-known *Guess&Check* programming methodology. In particular, the following rule guesses an assignment for the registrations to an operating room in a given day and shift among the ones permitted by the MSS for the particular specialty the registration belongs to.

$$\begin{aligned} x(R,P,O,S,D) \lor nx(R,P,O,S,D) \text{ :- } & \mathit{registration}(R,P,\_,SP), \\ & mss(O,S,SP,D). \end{aligned} \quad (1)$$

Note that $nx(R,P,O,S,D)$ is a fresh atom representing that a registration is not assigned to an operating room in a specific day. The same registration should not be assigned more than once, in different operating rooms or shifts. This is assured by the constraints:

$$\begin{aligned} \text{:- } x(R,P,O,S1,\_),\ x(R,P,O,S2,\_),\ S1 \mathrel{!=} S2. \\ \text{:- } x(R,P,O1,S,\_),\ x(R,P,O2,S,\_),\ O1 \mathrel{!=} O2. \\ \text{:- } x(R,P,O1,S1,\_),\ x(R,P,O2,S2,\_),\ O1 \mathrel{!=} O2,\ S1 \mathrel{!=} S2. \end{aligned} \quad (2)$$

Note that in our setting there is no requirement that every registration must actually be assigned.

Surgery requirement. With this constraint, we impose that the total length of surgery durations assigned to a shift is less than or equal to the shift duration.

$$\begin{aligned} \mathit{surgery}(R,SU,O,S) \text{ :- } x(R,\_,O,S,\_), \mathit{registration}(R,\_,SU,\_,\_,\_). \\ \text{:-} x(\_,\_,O,S,\_), \#sum\{SU,R : \mathit{surgery}(R,SU,O,S)\} > N, \mathit{duration}(N,O,S). \end{aligned} \quad (3)$$

Minimization. Registrations are characterized by the three priority levels P_1, P_2 and P_3. 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 procedure is accomplished through constraint (4) for the priority 1 and the weak constraints (5) and (6) for priority 2 and 3, respectively.

$$:\!-\ N = \mathit{totRegsP1} - \#\mathit{count}\{R : x(R, 1, \_, \_, \_)\},\ N > 0. \tag{4}$$

$$:\sim\ N = \mathit{totRegsP2} - \#\mathit{count}\{R : x(R, 2, \_, \_, \_)\}.\ [N@3] \tag{5}$$

$$:\sim\ N = \mathit{totRegsP3} - \#\mathit{count}\{R : x(R, 3, \_, \_, \_)\}.\ [N@2] \tag{6}$$

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 system.

4.2 Rescheduling

In the rescheduling problem, we start from an already-defined schedule that for some reasons could not be followed to the end and must be partially scheduled again. In particular, we took into account the case where some patients could not be operated in their assigned slot and must be reallocated in one of the slots in the remaining part of the original planning period.

Data Model. The predicates representing the facts of our encoding are the following:

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

Encoding. The new encoding includes only rules (1), (2) and (3) from the previous encoding, where atoms over the predicate x and nx are replaced with y and ny, respectively. Additionally, a constraint 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):

$$:\!-\ \mathit{not}\ y(R, P, \_, \_, \_),\ x(R, P, \_, \_, \_). \tag{7}$$

The main objective of the scheduling was to assign the largest possible number of registrations to the OR shifts. On the contrary, in the rescheduling problem the objective is to reassign all the previously allocated registrations and the reallocated ones with the least possible disruption to the old schedule. In order to do so, we compute and minimize the difference in days between the new and old assignments for each registration. This means that the rules (4), (5) and (6) are replaced by:

$$\begin{aligned} \mathit{difference}(DF, R) :\!-\ & y(R, \_, \_, \_, D),\ x(R, \_, \_, \_, \mathit{OldD}),\ DF = |D - \mathit{OldD}|. \\ :\sim\ & T = \#\mathit{sum}\{DF, R :\ \mathit{difference}(DF, R)\}.\ [T@1] \end{aligned} \tag{8}$$

5 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 [17], version 5.5.2.

5.1 ORS

The test case 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 respectively 15, 10, 7, 5, 3, 2 and 1 work days;
- 10 operating rooms unevenly distributed among the specialties;
- 5 hours long morning and afternoon shifts for each operating room, summing up to a total of respectively 1500, 1000, 700, 500, 300, 200 and 100 hours of OR 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 the beginning of 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 1. In

Table 1. Parameters for the random generation of the scheduler input.

Specialty	Registrations							ORs	Avg. Surgery Duration (min)	Coefficient of Variation
	15-day	10-day	7-day	5-day	3-day	2-day	1-day			
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		

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 2, as the average of 10 runs for each benchmark. Table 2 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 10 seconds.

A detailed analysis of the performance is reported in Table 3 for the target 5-day planning period. In particular, for each of the 10 runs executed, Table 3 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 than 95%, but for an instance having efficiency of 92%.

Table 2. 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 3. 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%

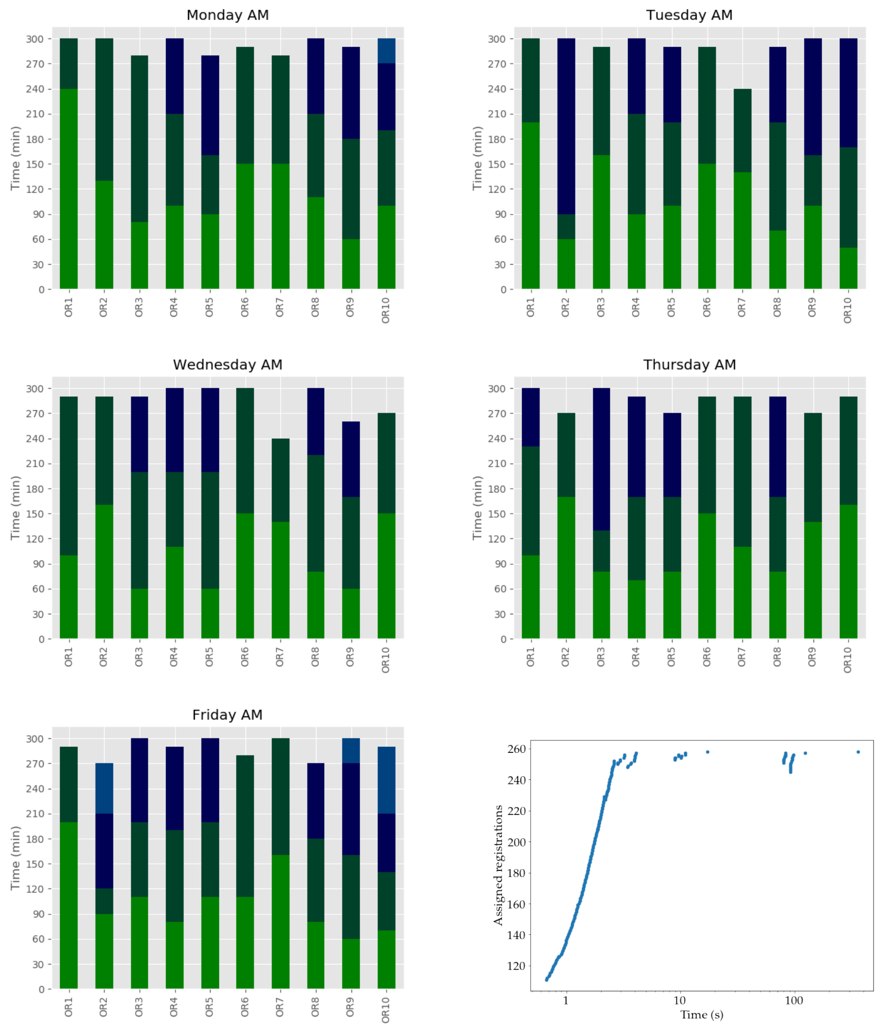

Fig. 1. Example of scheduling with 350 registrations for 5 days, and time scale (bottom-right).

Finally, in 5 plots of Figure 1 we (partially) present the results achieved on one instance (i.e., the first instance of Table 3) with 350 registrations for 5 days. Each colored block in the respective plots corresponds to a registration assigned to one of the 10 operating rooms. The remaining space up to the 300 minutes limit represent the idle time of the OR. Only the data about the morning assignments are showed; the ones for the afternoon are similar (qualitatively). The bottom-right plot shows, instead, the evolution of the solution quality when 600 seconds are granted to the same instance.

5.2 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.

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. 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 number of registrations present in the old schedule is 43, *(iii)* 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 4, 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 described in (8), showing the sum of all day displacements the old registrations were subject to in the resulting new schedule.

Table 4. 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

6 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 specified. 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. We have identified five different constraints that combined together cover most user needs, and are detailed in the following paragraph. Finally, note that such customizable constraints can be used for both scheduling and rescheduling.

Customizable constraints. The customizable constraints are not strictly required for the working of the program but allow the user to tweak as they prefer the final results. Each of these constraints can be activated at runtime for multiple registrations and can involve different selection of days, ORs, and shifts.

Given a set of n registrations, defined by the user through the app interface and characterized by the ids $r_i i = 1, .., n$, the first imposes that the set can be assigned only in a chosen period, defined as all the operating room shifts between the initial (i) and the final (f) days, where i and f are parameters provided by the user. For each registration we impose:

$$\begin{aligned} &:\!-\ x(r_i, \_, \_, S, \_),\ S < i. \\ &:\!-\ x(r_i, \_, \_, S, \_),\ S > f. \end{aligned} \tag{9}$$

The second constraint can be used to forbid a specific shift s to the chosen registrations:

$$:\!-\ x(r_i, \_, \_, s, \_). \tag{10}$$

The third and fourth customizable constraints regard forbidding or enforcing the use of a specific OR o for a set of registrations, respectively:

$$\begin{aligned} &:\!-\ x(r_i, \_, o, \_, \_), \mathit{registration}(r_i, \_, \_, \_), mss(o, \_, \_, \_). \\ &:\!-\ \mathit{not}\ x(r_i, \_, o, \_, \_), \mathit{registration}(r_i, \_, \_, \_), mss(o, \_, \_, \_). \end{aligned} \tag{11}$$

The last constraint can be used if the user wants to assign a set of registrations as temporally close as possible to a specific OR shift, 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 $prefS$) shifts and tries to minimize it:

$$\begin{aligned} &distance(N, r_i) :\!-\ x(r_i, \_, \_, S, \_),\ N = |S - prefS|. \\ &:\sim\ T = \#sum\{N, R :\ distance(N, R)\}.\ [T@4] \end{aligned} \tag{12}$$

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

7 Related Work

We are not aware of any previous attempt to solve the ORS problem using ASP algorithms, however an extensive literature approaching this problem with different techniques has been developed.

Aringhieri et al. [5] 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 large set of instances based on real data. In

[18], the same authors introduced a hybrid two-phase optimization algorithm which exploits the potentiality of 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] developed a bin packing model with a multi-step approach and a priority-type-duration (PTD) rule. The model maximizes utilization and minimizes the idle time, which consequently affects the cost at the planning phase and was programmed using MATLAB. Molina-Pariente et al. [20] 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 (BP) operators. They adapted existing heuristics to the problem and compared them to their own heuristics using a test bed based on the literature. The tests were performed with the software Gurobi.

The rescheduling problem was addressed by Shu et al. [22], using an extension of the Longest Processing Time (LPT) algorithm, which was used to solve the atomic job shop scheduling problem. Zhang et al. [23] 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 (MDP) with dead ends (fSSPDE), and solved using MATLAB by the method of asynchronous value iteration.

Finally, in the introduction we have already reported that ASP has been already successfully used for solving hard combinatorial and application problems in several research areas. ASP encodings were proposed for scheduling problems other than ORS, as examples *Incremental Scheduling Problem* [10], where the goal is to assign jobs to devices such that their executions do not overlap one another; *Team Building Problem* [21], where the goal is to allocate the available personnel of a seaport for serving the incoming ships; and *Nurse Scheduling Problem* [4, 13], where the goal is to create a scheduling for nurses working in hospital units.

8 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 re-scheduling 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 re-scheduling is specified by modularly adding ASP rules to (part of) the 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.
All benchmarks and encodings employed in this work can be found at: `http://www.star.dist.unige.it/~marco/AIIA2018/material.zip`.

References

1. Abedini, A., Ye, H., Li, W.: Operating Room Planning under Surgery Type and Priority Constraints. Procedia Manufacturing 5, 15–25 (2016)
2. Abseher, M., Gebser, M., Musliu, N., Schaub, T., Woltran, S.: Shift design with answer set programming. Fundam. Inform. 147(1), 1–25 (2016)
3. Alviano, M., Dodaro, C.: Anytime answer set optimization via unsatisfiable core shrinking. TPLP 16(5-6), 533–551 (2016)
4. Alviano, M., Dodaro, C., Maratea, M.: An advanced answer set programming encoding for nurse scheduling. In: AI*IA. LNCS, vol. 10640, pp. 468–482. Springer (2017)
5. Aringhieri, R., Landa, P., Soriano, P., Tnfani, E., Testi, A.: A two level metaheuristic for the operating room scheduling and assignment problem. Computers & Operations Research 54, 21–34 (2015)
6. Balduccini, M., Gelfond, M., Watson, R., Nogueira, M.: The USA-Advisor: A case study in answer set planning. In: LPNMR. LNCS, vol. 2173, pp. 439–442. Springer (2001)
7. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM 54(12), 92–103 (2011)
8. Buccafurri, F., Leone, N., Rullo, P.: Enhancing Disjunctive Datalog by Constraints. IEEE Trans. Knowl. Data Eng. 12(5), 845–860 (2000)
9. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: ASP-Core-2 Input Language Format (2013), `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.01c.pdf`
10. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: Design and results of the Fifth Answer Set Programming Competition. Artif. Intell. 231, 151–181 (2016)
11. Dodaro, C., Gasteiger, P., Leone, N., Musitsch, B., Ricca, F., Schekotihin, K.: Combining answer set programming and domain heuristics for solving hard industrial problems (application paper). TPLP 16(5-6), 653–669 (2016)
12. Dodaro, C., Leone, N., Nardi, B., Ricca, F.: Allotment problem in travel industry: A solution based on ASP. In: RR. LNCS, vol. 9209, pp. 77–92. Springer (2015)
13. Dodaro, C., Maratea, M.: Nurse scheduling via answer set programming. In: LPNMR. LNCS, vol. 10377, pp. 301–307. Springer (2017)
14. Erdem, E., Öztok, U.: Generating explanations for biomedical queries. TPLP 15(1), 35–78 (2015)
15. Faber, W., Pfeifer, G., Leone, N.: Semantics and complexity of recursive aggregates in answer set programming. Artif. Intell. 175(1), 278–298 (2011)
16. Gavanelli, M., Nonato, M., Peano, A.: An ASP approach for the valves positioning optimization in a water distribution system. J. Log. Comput. 25(6), 1351–1369 (2015)
17. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Wanko, P.: Theory solving made easy with clingo 5. In: ICLP (Technical Communications). OASICS, vol. 52, pp. 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
18. Landa, P., Aringhieri, R., Soriano, P., Tnfani, E., Testi, A.: A hybrid optimization algorithm for surgeries scheduling. Operations Research for Health Care 8, 103–114 (2016)
19. Maratea, M., Pulina, L., Ricca, F.: A multi-engine approach to answer-set programming. TPLP 14(6), 841–868 (2014)
20. Molina-Pariente, J.M., Hans, E.W., Framinan, J.M., Gomez-Cia, T.: New heuristics for planning operating rooms. Computers & Industrial Engineering 90, 429–443 (2015)
21. Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., Leone, N.: Team-building with answer set programming in the gioia-tauro seaport. TPLP 12(3), 361–381 (2012)
22. Shu, A.C., Subbaraj, I., Phan, L.: Operating Room Rescheduler (2015)
23. Zhang, J., Dridi, M., El Moudni, A.: A stochastic shortest-path MDP model with dead ends for operating rooms planning. pp. 1–6 (Sep 2017)