Scheduling Pre-Operative Assessment Clinic with Answer Set Programming*

Simone Caruso[1][0000−0002−2724−4342], Giuseppe Galatà[2][0000−0002−1948−4469], Marco Maratea[1,3][0000−0002−9034−2527]**, Marco Mochi[1][0000−0002−5849−3667], and Ivan Porro[2][0000−0002−0601−8071]

[1] DIBRIS, University of Genoa, Genova, Italy, {name.surname}@unige.it
[2] SurgiQ srl, Genova, Italy, {name.surname}@surgiq.com
[3] DeMaCS, University of Calabria, Rende, Italy, maratea@mat.unical.it

Abstract. The problem of scheduling Pre-Operative Assessment Clinic (PAC) consists of assigning patients to a day for the exams needed before a surgical procedure, taking into account patients with different priority levels, due dates, and operators availability. Realizing a satisfying schedule is of upmost importance for a hospital, since delay in PAC can cause delay in the subsequent phases, thus lowering patients' satisfaction. In this paper, we propose a two-phase solution to the PAC problem: In the first phase, patients are assigned to a day taking into account a default list of exams; then, in the second phase, having the actual list of exams needed by each patient, we use the results of the first phase to assign a starting time to each exam. We first present a mathematical formulation for both problems. Further, we present a solution where modeling and solving are done via Answer Set Programming (ASP). We then introduce a rescheduling solution that may come into play when the scheduling solution can not be applied fully. Experiments employing synthetic benchmarks on both scheduling and rescheduling show that both solutions provide satisfying results in short time. We finally show the implementation and usage of a web application that allows to run our scheduling solution and analyze the results graphically in a transparent way.

Keywords: Answer Set Programming · Pre-Operative Assessment Clinic Scheduling · Digital Health

1 Introduction

The Pre-Operative Assessment Clinic (PAC) scheduling problem is the task of assigning patients to a day, in which the patient will be examined and prepared to a surgical operation, taking in account, e.g., patients with different priority levels, due dates, and operators availability. The PAC consists of several exams needed by patients to ensure they are well prepared for their operation. This allows patients to stay at home until the morning of the surgery, instead of being admitted to the hospital one or two days

* This is an extended and revision version of a paper appearing in the LNCS post-proceedings of the AI*IA 2021 conference [16].

** Corresponding author.

before the scheduled operation; moreover, this allows to reduce waiting time between the exams, thus increasing patients's satisfaction [35], and to avoid the cancellation of the surgery [27]. A proper solution to the PAC scheduling problem is vital to improve the degree of patients' satisfaction and to reduce surgical complications.

In our solution, the problem, whose specifications have been provided by a potential SurgiQ[4] client, is divided into two sub-problems [23]. In the first sub-problem, patients are assigned to a day taking into account a default list of exams, and the solution has to schedule patients before their due date and prioritize the assignments to patients with higher priority. In the second sub-problem, the scheduler assigns a starting time to each exam needed by the patients, considering the available operators and the duration of the exams. Our two-phase solution is not guaranteed to find the best possible overall solution, but has been designed in this way because: (i) it simplifies the overall encoding and its practical use, and (ii) it mimics how PAC schedules are often computed (manually) in hospitals. Complex combinatorial problems, possibly involving optimizations, such as the PAC problem, are usually the target applications of AI languages such as Answer Set Programming (ASP). Indeed, thanks to the readability of its specification and the availability of efficient solvers [2, 30], driven by the organization of a number of ASP Competitions (see, e.g., [13, 31, 32]), ASP has been successfully employed for solving hard combinatorial problems in several research areas, including Artificial Intelligence, Bioinformatics, Data Integration and Query Answering, Natural Language Processing and Understanding, and Hydroinformatics, and it has been also employed to solve many scheduling problems [40, 33, 1, 19, 20, 22], including industrial contexts (see, e.g., [24, 26, 42, 4] for detailed descriptions of some ASP applications).

In this paper, we first present a mathematical formulation of both sub-problems: In the solution of the first sub-problem, the scheduler minimizes the number of unassigned patients, while in the second sub-problem, using as input the result of the first sub-problem, the time each patient stays at the hospital is minimized. We then apply ASP to model and solve the PAC scheduling problem, by presenting two ASP encodings for both sub-problems. We also propose a rescheduling solution that may come into play when the scheduling solution can not be applied fully, due to, e.g., unavailable patients, or an operator who is suddenly missed. We run an experimental analysis on synthetic PAC benchmarks with realistic sizes and parameters inspired from data seen in literature, varying the number of scheduled patients and the available operators. Overall, results using the state-of-the-art ASP solver CLINGO [30] show that ASP is a suitable solving methodology also for the PAC scheduling and rescheduling problems. We finally present the implementation and show the usage of a web application that allows to insert the main parameters of the problem, run our scheduling solution and analyze the results graphically without the need of installing any ASP software locally.

The paper is structured as follows. Sections 2 and 3 present an informal description of the problem and a precise, mathematical formulation, respectively. Then, after the needed background on ASP in Section 4, Section 5 shows our ASP encodings for both phases, whose experimental evaluation is presented in Section 6. Rescheduling ASP solutions are introduced and evaluated in Section 7, and the web application is shown

[4] `https://surgiq.com/`.

in Section 8. The paper ends by discussing related works and conclusion in Section 9 and 10, respectively.

2 Problem Description

Pre-hospitalization is the phase in which the patient enters the facility to perform laboratory tests, radiological and cardiological examination, counseling for anesthesiological suitability for surgery, and any other examinations that the medical specialists deem appropriate for the definition of the pre-intervention diagnostic pathway (pneumological examination, nephrological examination, etc...). The management of the pre-intervention pathway is crucial within healthcare organizations as it involves the alignment of multiple facilities to ensure that services are delivered on time and in the correct manner. However, the high number of services involved in this pathway can create critical issues in the patient's journey in terms of service delivery time and alignment between the various providers (Cardiology, Pulmonology, Nephrology, etc...).

In many hospitals, including the one from which the specifications we employ have been taken, the date of the pre-operative procedures is set at a very early stage, before the exact number of necessary exams and visits is known. Later on, after a first contact with the patient, the number and nature of the exams and visits are settled, and can be scheduled precisely, assigning to each of them a starting time during the day set in the first phase. Often this problem is still solved as a manual process having critical issues such as the scheduling of appointments and the alignment between different providers. The automatic scheduling aims to ensure completion of the pathway in one day, to avoid repeated patient accesses and to allow the patients performing all services on time.

In the first of the two phases outlined above, we consider that each patient requires a default list of exams according to his specialty, i.e., the lists of exams are equal for patients requiring the same specialty but differ among specialties, and the scheduler assigns the day of PAC without considering the starting time of the exams, but just assigning a temporary starting time of the first and the last exam required by each patient to ensure that such patients can be assigned in the same day. Thus, in the first sub-problem the solution assigns patients overestimating the duration and the number of exams needed. In particular, all the optional exams, such as exams required by smokers or patients with diabetes, are assigned to all the patients in the first phase. The overestimation is important in order to guarantee the solvability of the second sub-problem. Then, when the operation day is closer, the hospital knows exactly the exams needed by each patient and can assign the starting time of each exam. Going in more details, the first sub-problem consists of scheduling appointments in a range of days for patients requiring a surgical operation. Each patient is linked to a due date, a target day, and a priority level: The due date is the maximum day in which (s)he can be assigned, the target day is the optimal day in which schedule the appointment, while the solution prioritizes patients with higher priority level. There are several exam areas, corresponding to the locations in which patients will be examined. Each exam area needs operators to be activated and has a limited time of usage. Each operator can activate three different exam areas, but they can be assigned to just one exam area for each day. The solution must assign the operators to the exam areas, to activate them, and the day of PAC to

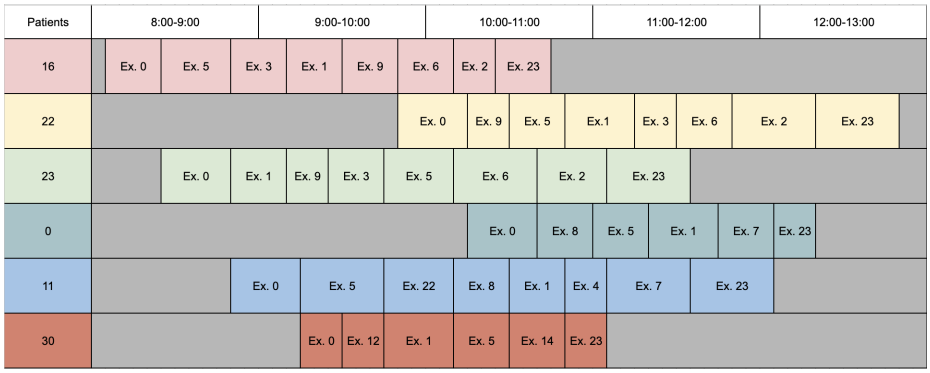

Fig. 1. Schedule example: assignments of the starting time of the different exam locations of each patient in a single day.

patients, ensuring that the total time of usage of each exam location is lower than its limit. Since in this first sub-problem the list of exams needed by patients is not the final list, i.e., just the first and the last exam are the same for every patient, and in the second sub-problem some exams could be added, the solution schedules patients leaving some unused time to each exam area. An optimal solution minimizes the number of unassigned patients, giving priority to patients with higher priority levels, and ties are broken by minimizing the difference between the day assigned and the target day of each patient, giving again precedence to patients with higher priority.

In the second sub-problem, patients are linked to their real exams, so the solution has to assign the starting time of each exam, having the first sub-problem already assigned the day. The input consists of registrations, exams needed by patients and the exam areas activated. Exams are ordered, so the solution must assign the starting time of each exam respecting their order and their duration, by considering that each exam area can be used by one patient at a time. Finally, the solution minimizes the difference between the starting time of the first exam and the last exam of each patient.

Figure 1 shows an example of the schedule that we want to compute, by assigning the starting times (x-axis) to each patient (y-axis) in a particular day. The patients that must be scheduled are the same that are set by the first sub-problem in this day. As can be seen in Figure 1, in this case the scheduler is able to assign all patients optimally; indeed, all patients have no waiting time among the exams and thus the time spent in the hospital is reduced to the minimum, while respecting the constraints of the problem.

3 Formalization of the PAC Scheduling problem

In this section, we provide a mathematical formulation of the two sub-problems.

Definition 1 (first PAC sub-problem). *Let*

- *R be a finite set of registrations;*
- *$E = \{e_1, \ldots, e_m\}$ be a set of m exams;*

- $EL = \{el_1, \ldots, el_n\}$ *be a set of n exam locations;*
- O *be a finite set of operators;*
- $D = \{d : d \in [1..14]\}$ *be the set of all days;*
- $T = \{t : t \in [1..ts]\}$ *be the set of all time slots;*
- $\delta : R \mapsto \{1,2,3,4\}$ *be a function associating a registration to a priority;*
- $\rho : R \times E \mapsto \mathbb{N}$ *be a function associating a registration and an exam to a duration such that for a registration r and an exam e if* $\rho(r,e) > 0$ *then the registration r requires the exam e;*
- $\omega : R \mapsto \mathbb{D}$ *be a function associating a registration to a due date;*
- $\lambda : R \mapsto \mathbb{D}$ *be a function associating a registration to a target day;*
- $\sigma : E \mapsto \mathbb{EL}$ *be a function associating an exam to the exam location;*
- $\mu : EL \times D \mapsto \mathbb{N}$ *be a function associating an exam location and a day to the maximum number of registrations that can be assigned concurrently, such that* $\mu(el,d) = n$ *if at most n registrations can be assigned concurrently to the exam location el in the day d.*
- $\tau : EL \times D \mapsto \mathbb{N}$ *be a function associating an exam location and a day to the required number of operators to be activated, such that* $\tau(el,d) = n$ *if the exam location el in the day d requires n operators to be activated;*
- $\theta : O \times EL \times D \mapsto \{0,1\}$ *be a function such that* $\theta(o,el,d) = 1$ *if the operator o is assigned to the exam location el in the day d, and* 0 *otherwise;*
- *ts be a constant that is equal to the number of time slots.*

Let $x : R \times D \times T \mapsto \{0,1\}$ *be a function such that* $x(r,d,t) = 1$ *if the registration r is assigned to the day d and the time slot t, and* 0 *otherwise. Moreover, for a given x, let* $A_x = \{(r,d,t) : r \in R, d \in D, t \in T, x(r,d,t) = 1\}$.

Then, given sets R, E, EL, O, D, T, and functions δ, ρ, ω, λ, σ, μ, τ, θ, *the first PAC sub-problem is defined as the problem of finding a schedule x such that*

(c_1) $|\{(d,t) : (r,d,t) \in A_x\}| \leq 1 \quad \forall r \in R, d \leq \omega(r)$;
(c_2) $|\{(d,t) : (r,d,t) \in A_x\}| = 0 \quad \forall r \in R, d > \omega(r)$;
(c_3) $t \leq ts - \sum_{e \in E} \rho(r,e) \quad \forall (r,d,t) \in A_x$;
(c_4) $|\{r : (r,d,t') \in A_x, t \geq t', t < t' + \rho(r,e_1)\}| \leq \mu(el) \quad \forall d \in D, t \in T, el = \sigma(e_1)$;
(c_5) $|\{r : (r,d,t') \in A_x, t \geq t' + \sum_{e \in E} \rho(r,e) - \rho(r,e_m), t < t' + \sum_{e \in E} \rho(r,e)\}| \leq \mu(el) \quad \forall d \in D, t \in T, el = \sigma(e_m)$;
(c_6) $|\{o : \theta(o,el,d) = 1\}| = \tau(el,d) \quad \forall (r,d,t) \in Ax, \rho(r,e) > 0, el = \sigma(e)$;
(c_7) $|\{el : \theta(o,el,d)\}| \leq 1 \quad \forall o \in O, \forall d \in D$.

Condition (c_1) ensures that each registration is assigned at most one time in the days before the due date associated with the registration. Condition (c_2) ensures that each registration is not assigned after the due date associated with the registration. Condition (c_3) ensures that for each registration the sum of the duration of the exams required plus the time slot assigned to the registration is less than the constant ts. Condition (c_4) ensures that the number of registrations having the first exam at a time slot t is less than or equal to the allowed value for each time slot. Condition (c_5) ensures that the number of registrations having the last exam at a time slot t is less then or equal to the allowed value for each time slot. Condition (c_6) ensures that for each exam location used by

at least one registration, the required number of operators is assigned. Condition (c_7) ensures that each operator is assigned to at most one exam location each day.

Definition 2 (Unassigned registrations). *Given a solution x, let* $U_x^{pr} = \{r : r \in R, \delta(r) = pr, r \notin A_x\}$. *Intuitively,* U_x^{pr} *represents the set of registrations of priority pr that were not assigned to any day.*

Definition 3 (Distance target day). *Given a solution x, let* $t_x^{pr} = \sum_{x(r,d,t)\in A_x, \delta(r)=pr} | d - \lambda(r) |$. *Intuitively,* t_x^{pr} *represents the sum of the distance between the day assigned to the registrations of priority pr and the target day associated.*

Definition 4. *A solution x is said to dominate a solution x' if* $|U_x^{pr}| < |U_{x'}^{pr}|$ *for the biggest pr for which* $|U_x^{pr}| \neq |U_{x'}^{pr}|$ *or if* $|U_x^{pr}| = |U_{x'}^{pr}|$ *for all the pr and* $|t_x^{pr}| < |t_{x'}^{pr}|$ *for the biggest pr for which* $|t_x^{pr}| \neq |t_{x'}^{pr}|$. *A solution is optimal if it is not dominated by any other solution.*

Definition 5 (second PAC sub-problem). *Let*

- $\beta : R \times E \mapsto \mathbb{N}$ *be a function associating a registration and an exam to a value corresponding to the order in which the exam must be assigned, such that* $\beta(r,e) = n$ *and* $\beta(r,e') = n'$ *and* $n > n'$ *if the registration r must do the exam e after the exam* e'*;*
- $\gamma : EL \mapsto \mathbb{N}$ *be a function associating an exam location to the starting time of the exam location, such that* $\gamma(el) = n$ *if n is the first time slot in which a registration requesting the exam location el can be assigned;*
- $\xi : EL \mapsto \mathbb{N}$ *be a function associating an exam location to the ending time of the exam location, such that* $\gamma(el) = n$ *if n is the last time slot in which a registration requesting the exam location el can be assigned.*

Let $x : R \times E \times EL \times D \times T \mapsto \{0,1\}$ *be a function such that* $x(r,e,el,d,t) = 1$ *if the registration r and the exam e are assigned to the exam location el in the day d and in the time slot t, and* 0 *otherwise. Moreover, for a given x let* $A_x = \{(r,e,el,d,t) : r \in R, e \in E, el \in EL, d \in D, t \in T, x(r,e,el,d,t) = 1\}$.

Then, given sets R, E, EL, D, T, and functions ρ, β, γ, ξ, μ, *the second PAC sub-problem is defined as the problem of finding a schedule x, such that*

(c_8) $|\{(d,t) : (r,e,el,d,t) \in A_x\}|=1 \quad \forall e \in E, \forall r \in R, \rho(r,e) > 0, \sigma(e) = el;$
(c_9) $\gamma(el) \leq t \leq \xi(el) - \beta(r,e) \quad \forall (r,e,el,d,t) \in A_x;$
(c_{10}) $(r,e,el,d,t) \notin A_x \quad \forall (r,e',el',d,t') \in A_x, \forall e \in E, \rho(r,e') = dt, \forall t \in T, t' \leq t < t' + dt;$
(c_{11}) $t > t' \quad \forall (r,e,el,d,t) \in A_x, (r,e',el,d,t') \in A_x, \beta(r,e) > \beta(r,e');$
(c_{12}) $|\{r : (r,e,el,d,t') \in A_x, \sigma(e) = el, t \geq t', t < t + \rho(r,e)\}| \leq \mu(el) \quad \forall el \in EL, t \in T.$

Condition (c_8) ensures that each exam is assigned exactly once. Condition (c_9) ensures that each exam is assigned after the starting time of the required exam location and before the closing time of the required exam location minus the duration of the exam. Condition (c_{10}) ensures that for each registration each exam is assigned after that the exam before is ended. Condition (c_{11}) ensures that each exam is assigned after the

exams lower in the ordering are assigned. Condition (c_{12}) ensures that the number of exams assigned to a location is lower than the maximum availability for each location in any time slot.

Definition 6 (Time in hospital). *Given a solution x, let*

$$m_x = \sum_{(r,e_1,el,d,t)\in A_x, x(r,e_m,el,d,t')\in A_x} (t' + \rho(r,e_m) - t - \sum_{e\in E} \rho(r,e)).$$

Intuitively, m_x represents the waiting time between the exams. It is evaluated as the sum of the differences between the time spent in the hospital (evaluated as the difference between the ending time of the last exam and the starting time of the first exam) and the sum of the durations of the exams required by each registration.

Definition 7. *A solution x is said to dominate a solution x' if $m_x < m_{x'}$. A solution is optimal if it is not dominated by any other solution.*

4 Background on ASP

Answer Set Programming (ASP) [10] is a programming paradigm developed in the field of non-monotonic 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 [10, 12]. 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 an 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, e.g., the function *#count* computes the number of terms.

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 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:

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

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 $a_1|\ldots|a_n$ 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, where a rule r is *safe* if 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 a positive atom in *Conj*.

A *weak constraint* [11] ω is of the form:

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

where w and l are the weight and level of ω, respectively. (Intuitively, $[w@l]$ is read as "weight w at level l", where the 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* ℓ) 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* [25] 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) [25].

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 $p \mid p'$, where p' is a fresh new atom not appearing elsewhere in the program, meaning that the atom p can be chosen as true.

5 ASP Encoding

In this section, starting from the specifications in Section 2 and the formalization of Section 3, we present the ASP encoding for the two sub-problems, in two separate subsections, based on the input language of CLINGO [28].

5.1 ASP encoding for the first PAC sub-problem

In the following, we present the ASP encoding for the first sub-problem.

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

- Instances of `reg(RID,PR,TARGET,TOTDUR,DUEDATE)` represent the registrations, characterized by an id (`RID`), the priority level (`PR`), the ideal day in which the registration should be assigned (`TARGET`), the sum of the durations of the exams needed by the registration (`TOTDUR`), and the due date (`DUEDATE`).
- Instances of `exam(RID,AREAID,DUR)` represent the exam linked to the registration identified by an id (`RID`), the exam area (`AREAID`), and the duration (`DUR`).
- Instances of `examLoc(AREAID,NOP,DAY,N)` represent the exam areas, characterized by an id (`AREAID`), which requires `NOP` operators to be activated in a day (`DAY`), and can be concurrently assigned up to `N` registrations.
- Instances of `operators(ID,AREAID,DAY)` represent the operators, characterized by an id (`ID`), that can be assigned to the exam area (`AREAID`) in a day `(DAY)`.
- Instances of `day(DAY)` represent the available days.
- The constant ts represents the number of time slots considered.
- The constants $first\_exam$ and $last\_exam$ correspond to e_1 and e_m in Section 3 (i.e., the first and the last exam required by every registration, respectively).
- The constants $totRegsP1$, $totRegsP2$, $totRegsP3$, and $totRegsP4$ represent the number of registrations with priority 1, 2, 3, and 4, respectively.

The output is an assignment represented by an atom of the form `x(RID,PR,ST,ET,DAY)`, where the intuitive meaning is that the exams of registration with id `RID` and priority level `PR` are assigned to the day `DAY`, the temporary starting time of the first exam is `ST` and the temporary ending time of the last exam is `ET`; and an atom of the form `operator(ID,AREAID,DAY)`, where the intuitive meaning is that the operator with id `ID` is assigned to the exam area `AREAID` on the day `DAY`.

Encoding. The related encoding is shown in Figure 2, and is described in the following. To simplify the description, we denote as r_i the rule appearing at line i of Figure 2.

Rule r_1 assigns registrations to a day and to a time slot. The assignment is made assigning a day that is before the due date and to a temporary time slot such that the sum of the time slot and the duration of the exams required is less than the constant ts. Rule r_2 checks that every registration is assigned to a day with all the exams area needed to be activated. Rule r_3 is used to ensure that the number of registrations having the first exam is less or equal to the allowed value in every time slot. Rule r_4 is used to ensure that the number of registrations having the last exam is less or equal to the allowed value

```
{x(RID,PR,ST,ST+TOTDUR,DAY) : day(DAY),time(ST), DAY < DUEDATE, ST <= ts-TOTDUR} 1 :-
    reg(RID,PR,TARGET,TOTDUR,DUEDATE).
:- x(RID,_,_,_,DAY), exam(RID,AREAID,_), not examLoc(AREAID,_,_,DAY,_).
:- #count{RID: x(RID,_,ST,_,DAY),exam(RID,first_exam,DUR), T >= ST, T < ST+DUR} > N,
    examLoc(first_exam,_,_,DAY,N), day(DAY),time(S).
:- #count{RID: x(RID,_,_,ET,DAY),exam(RID,last_exam,DUR), T < ET, T >= ET-DUR} > N,
    examLoc(last_exam,_,_,DAY,N), day(DAY),time(T).
{operator(ID, AREAID, DAY) : operators(ID, AREAID, DAY)} == NOP :- examLoc(AREAID, NOP, _,
    DAY,_), x(RID, _, _, _,DAY), exam(RID, AREAID, _).
:- operator(ID,AREAID1,DAY), operator(ID,AREAID2,DAY), AREAID1 < AREAID2.
unassignedP1(N) :- M = #count {RID: x(RID,1,_,_,_)}, N = totRegsP1 - M.
unassignedP2(N) :- M = #count {RID: x(RID,2,_,_,_)}, N = totRegsP2 - M.
unassignedP3(N) :- M = #count {RID: x(RID,3,_,_,_)}, N = totRegsP3 - M.
unassignedP4(N) :- M = #count {RID: x(RID,4,_,_,_)}, N = totRegsP4 - M.
:~ unassignedP1(N). [N@8]
:~ unassignedP2(N). [N@7]
:~ unassignedP3(N). [N@6]
:~ unassignedP4(N). [N@5]
:~ x(RID,1,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@4,RID]
:~ x(RID,2,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@3,RID]
:~ x(RID,3,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@2,RID]
:~ x(RID,4,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@1,RID]
```

Fig. 2. ASP encoding of the first sub-problem

in every time slot. Rule r_5 assigns operators to the required exam areas, while rule r_6 checks that each operator is assigned to just one exam area in every day. Rules from r_7 to r_{10} derive intermediate atoms *unassignedP*1, *unassignedP*2, *unassignedP*3, and *unassignedP*4, respectively, that are used to count how many registrations with different priorities are not assigned to a day. Weak constraints from r_{11} to r_{14} use the atoms derived by rules from r_7 to r_{10} to minimize the number of unassigned registrations according to their priority. Finally, weak constraints from r_{15} and r_{18} minimize the difference between the assigned and target day of each registration, giving precedence to higher priorities.

5.2 ASP encoding for the second PAC sub-problem

We now move to the second sub-problem.

Data Model. The input data is the same of the first sub-problem for the atoms `exam` and `time`, while other atoms are changed:

- Instances of `reg(RID,DAY)` represent the registrations, characterized by an id (`RID`) assigned to a day (`DAY`).
- Instances of `examLoc(AREAID,DAY,AREAST,AREAET,N)` represent the exam areas, characterized by an id (`AREAID`), which in a day (`DAY`) has a starting time (`AREAST`) and a closing time (`AREAET`), and can provide the exam to `N` registrations.
- Instances of `phase(AREAID,ORD)` represent the order (`ORD`) of the exams provided by the exam area characterized by an id (`AREAID`).

The output is represented by an atom of the form `x(RID,AREAID,ST,ET,DAY)`, where the intuitive meaning is that the exam of the registration with id `RID` is in exam area `AREAID`, starts at time `ST` and ends at time `ET` on the day `DAY`.

```
1 {x(RID,AREAID,ST,ST+DUR,DAY) : examLoc(AREAID,DAY,AREAST,AREAET,_), time(ST), ST >= AREAST,
      ST <= AREAET-DUR} = 1 :- reg(RID,DAY), exam(RID,AREAID,DUR).
2 :- x(RID,AREAID1,ST1,_,_), x(RID,AREAID2,ST2,_,_), phase(AREAID1,ORD1), phase(AREAID2,ORD2),
      ORD2 < ORD1, ST1 < ST2.
3 :- #count{AREAID: x(RID,AREAID,ST,ET,DAY), T >= ST, T < ET} > 1, reg(RID,DAY), time(T).
4 :- #count{AREAID: x(RID,AREAID,ST,ET,DAY), T >= ST, T < ET} > N, examLoc(AREAID,DAY,_,_,N),
      time(T).
5 :~ reg(RID,_), x(RID,first_exam,ST,_,_), x(RID,last_exam,_,ET,_). [ET-ST@1, RID]
```

Fig. 3. ASP encoding of the second sub-problem

Encoding. The encoding consists of the rules reported in Figure 3. Rule r_1 assigns a starting and an ending time to each exam needed by every registration, checking that the time in which is assigned is within the opening time of the required exam area. Rule r_2 ensures that the order among the exams is respected. Rules r_3 checks that each registration is assigned to at most one exam for every time slot. Then, rule r_4 checks that each exam area provides the exam to at most `N` registrations for every time slot. Finally, rule r_5 minimizes the difference between the ending time of the last exam and the starting time of the first exam of each registration.

```
6 forbiddenAfter(RID,AREAID1,ST+DUR1) :- reg(RID,_,_), exam(RID,AREAID1,DUR1),
      phase(AREAID1,ORD1), #sum{DUR2: exam(RID,AREAID2,DUR2), phase(AREAID2,ORD2), ORD2 >
      ORD1 } = ST.
7 forbiddenBefore(RID,AREAID1,ST) :- reg(RID,_,_), exam(RID,AREAID1,DUR1),
      phase(AREAID1,ORD1), #sum{DUR2: exam(RID,AREAID2,DUR2), phase(AREAID2,ORD2), ORD2 <
      ORD1 } = ST.
8 {x(RID,AREAID,ST,ST+DUR,DAY): examLoc(AREAID,DAY,AREAST,AREAET,_),
      forbiddenAfter(RID,AREAID,FORB1), forbiddenBefore(RID,AREAID,FORB2), time(ST), ST >=
      AREAST, ST <= AREAET-DUR, ST <= ts-FORB1, ST > FORB2 } = 1 :- reg(RID,PRI,DAY),
      exam(RID,AREAID,DUR).
```

Fig. 4. Optimized encoding for pruning exams' starting time.

Domain specific optimizations. The encoding of the second sub-problem above is general, able to find optimal solutions, but the solver needs a large amount of time to prove optimality. However, some domain specific optimizations may be added to improve its performance. We present two domain specific optimizations, presented in the following two sub-paragraphs, which rely on the knowledge of the PAC domain for pruning impossible solutions.

Pruning of exams' starting time slots. As shown in Figure 3, in r_1 the starting time of the exams is guessed between all the available daily time slots, expressed by the atom `time`. Given that the exams must be assigned following an order, it is known the minimum number of time slots that each patient need to stay before and after each exam. Thus, the guess rule can be improved by reducing the number of possible starting time slots of each exam with the following constraints:

```
9  cost(RID, TOT) :- TOT = #sum{DUR,AREAID : exam(RID,AREAID,DUR)}, reg(RID,_,_).
10 :~ x(RID,first_exam,ST,_,_),x(RID,last_exam,_,ET,_), cost(RID,TOT), ET-ST-TOT >= 0. [
       ET-ST-TOT@1, RID]
```

Fig. 5. Optimized minimization rule

- an exam cannot start in a time slot if the remaining time slots are less than the minimum amount of time slots required to complete all the following exams;
- an exam cannot start in a time slot if the time slots before are less than the minimum amount of time slots required to complete the previous exams.

The encoding for pruning exams' starting time slots is obtained by substituting r_1 from Figure 3 with the rules reported in Figure 4, explained in the following. In rules r_6 and r_7 two new atoms `forbiddenAfter` and `forbiddenBefore` are defined as the minimum amount of time slots needed by each patient after (resp. before) each exam. obtained by computing the sum of the duration of the exams with the greater (resp. lower) phase value. In r_8 the two new atoms are used in the guess rule, so that the starting time is after the value computed by rule r_6, and after the difference between `lastTimeSlot`, that corresponds to the last time slot, and the value computed by r_7.

Minimization with lower bound. As it can be seen also in Figure 3, rule r_5 minimizes the time spent in the hospital by each patient, computed as the difference between the ending time of the last exam and the starting time of the first exam. However, the time spent in the hospital by each patient cannot be lower than the sum of the duration of all the required exams. Therefore, the minimization rule can be improved by computing the minimum time required by each patient and using it as a lower bound, so that candidate solutions below this value are pruned.

The enconding with the optimized weak constraint is obtained by substituting rule r_5 from Figure 3 with the rules shown in Figure 5. In rule r_9, the time to fully all the exams for each patient is computed as the sum of the duration of all the exams as the new auxiliary atom `cost`. The weak constraint in rule r_{10} then minimizes the difference between the planned total time (i.e., the difference between ending time and starting time of the last and first exam) and the lower bound previously computed, activating the weak constraint only when such difference is greater or equal than zero.

6 Experimental Results

In this section, we report the results of an empirical analysis of the PAC scheduling problem via ASP. For the first sub-problem (second subsection), data have been randomly generated using parameters inspired by literature and real world data (first subsection), then the results of the first sub-problem have been used as input for the second sub-problem (third subsection). The last subsection is then devoted to a comparison to alternative logic-based formalisms. The experiments were run on a AMD Ryzen 5 3600 CPU @ 3.60GHz with 16 GB of physical RAM. The ASP system used was CLINGO [28] 5.4.1, using parameters *--restart-on-model* for faster optimization

and *--parallel-mode 6* for parallel execution. Encodings and benchmarks employed in this section can be found at: `http://www.star.dist.unige.it/~marco/JLC2022/material.zip` .

6.1 PAC benchmarks

Data are based on the sizes and parameters of a typical middle sized hospital, with 24 different exam areas. The solution schedules patients in a range of 14 days, 5 hours per day, and for each day there are 60 time slots, thus the constant ts is set to 60, corresponding to 5 minute per time slot. To test scalability we generated 3 different benchmarks of different dimensions, having 40, 60, and 80 patients, respectively. Each benchmark was tested 10 times with different randomly generated input.

In particular, each registration is linked to a surgical specialty, and needs a number of exams between 5 and 13, according to the specialty, while the duration of each exam varies between 3 and 6 time slots. The priorities of the registrations have been generated from an even distribution of four possible values (with weights of 0.25 for registrations having priority 1, 2, 3, and 4, respectively). Moreover, a due date is randomly generated for each registration. For all the benchmarks, as said there are 24 exam areas and the operators, that are 35, can be assigned to 3 different exam areas. So, by increasing the number of patients while maintaining fixed the number of operators, we tested different scenarios with low, medium and high requests.

For the second sub-problem, we used the results of the first sub-problem as input. Thus, the number of patients and the exam locations activated depend on the assignments made by the first sub-problem. Patients require all the same first and last exam, while the other exams required by each patient are linked to an order that is randomly assigned and that must be respected by the scheduler. In the second sub-problem, clinics know the actual list of exams needed by patients: To simulate this scenario, we randomly selected the optional exams assigned to patients in the first sub-problem. The number of such exams depend on the specialty, and on ”status” of the patient: For example, optional exams are needed by patients that are over 65 years old or smokers.

6.2 Results for the first sub-problem

The first optimization criteria in the PAC scheduling sub-problem is to assign as many patients as possible, starting from patients with higher priority. On all the tested instances, our solution is able to assign a day to 406 out of 437 patients with the highest priority; moreover, in 23 out of 30 instances the solution assigns a day to all or all but one patients with the highest priority.

Table 1 summarizes the results obtained in this first sub-problem, with a timeout set to 300 seconds for each instance, and a preliminary analysis on CLINGO done also with other parameters, e.g., `--opt-strategy=usc` that allows CLINGO to employ a different optimization algorithm: In particular, the table shows the average percentage of patients from the 10 instances assigned with 40, 60, and 80 patients according to their priority level. The test with 80 patients shows in particular how the solution behaves in a scenario with high demand and low resource availability, since the number of exam

Table 1. Percentage of assigned patients according to their priority level.

Total #Patients	%P1 ASSIGNED	%P2 ASSIGNED	%P3 ASSIGNED	%P4 ASSIGNED
40	92%	92%	91%	83%
60	95%	89%	82%	65%
80	91%	87%	60%	44%

areas and operators is fixed. From the table we can observe that the percentage of patients with priority 1 and 2 assigned is just slightly lower than in the other benchmarks, while the percentage of patients assigned with lower priorities decreases, due to the few resources available compared to the demand.

Compared to the previous version of the encoding [16] used for the first sub-problem, this new version is able to assign more patients without leading to unsatisfiable problems in the second sub-problem. What is changed is that, while in the old version of the encoding we used a formula to overestimate the time required by each patient, in this new version we assign a temporary starting time for the exams in the first sub-problem. The starting time is used to check that the first and the last exam (that are mandatory for every patient) are not concurrently required by more than the allowed number of patients. This change in the encoding leads to more patients assigned in the first sub-problem and, in particular, while the previous version assigned 80% of patients with the highest priority, with the new version we are able to assign approx. 93% of patients with the highest priority. Moreover, testing the previous encoding using the instances with 80 patients, the old encoding was able to assign a date just to 45% of the patients with priority 1 and 2, while the new version assigns a date to 89% of patients.

The second optimization criteria is to have an assigned day that is as much close as possible to the target day. This optimization criteria is able to assign some patients close to their target day, while for other patients quality decreases. This is due to two reasons: The first one is that there is another optimization criteria with higher priority; thus, the scheduler already tries to assign as many patients as possible, without taking into account the target days of the patients, and the second one is that some patients have a target day in a day without their exam locations available, so, even in an optimal solution the assigned day to some patients would not be in the target day.

To corroborate the general explanation above, Figure 6 reports the result obtained by the scheduler with one of the instances with 60 patients as input. What can be seen from the graph is that some patients are assigned in the last days, while in some days before there are no patients assigned. This can be explained by the fact that the scheduler tries to assign as many patients as possible and do not try to assign as soon as possible the patients. Moreover, in some days patients can not be assigned due to the unavailability of the exam locations. Other instances behave similarly.

6.3 Results for the second sub-problem

This subsection reports the results of the second sub-problem, with the same experimental setting of the first. In the second sub-problem, the solution assigns the starting time of each exam of the patients. The input is taken from the results obtained in the

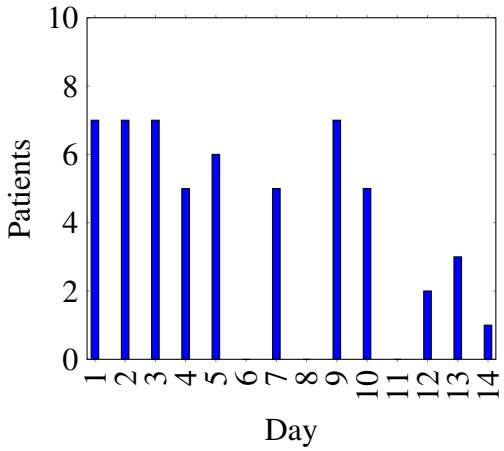

Fig. 6. Number of patients assigned to each day by the scheduler with 60 patients as input.

first sub-problem. The solution minimizes the difference between the ending time of the last exam and the starting time of the first exam. While minimizing this value the solution tries to minimize the time spent in the hospital by all patients. In this sub-problem, the scheduler is able to reach an optimal solution in all the instances tested. Moreover, in every instance tested there is at most one patient that has to wait for one time slot between two exams, while all the other patients have no waiting time between the exams. In particular, in all but three instances, patients are assigned without waiting time among the exams.

The timings required by the scheduler in the second sub-problem are shown in Figure 7, which represents the range of seconds required to reach the optimal solution in all the instances tested with the different number of patients, identified by the minimum and maximum times for solving the instances in the set, together with the mean and the median time. From the figure, it can be seen that it takes 16 seconds on average to find the optimal solution considering instances with 40 patients. While, considering instances with 60 and 80 patients, the scheduler finds the optimal solution on average in 33 seconds and 44 seconds, respectively. Moreover, even in the worst case with 80 patients the scheduler is able to find the optimal solution in less than 70 seconds.

To obtain these results we used the encoding defined in Section 5.2, included the domain specific optimizations. We are now interested in understanding the contribution that the two optimizations bring. Table 2 presents the mean time required to reach an optimal solution with the different version of the encoding. In particular, the second column of the table contains results for the encoding without optimizations, defined as BASIC, the third and fourth columns contain results with the two optimizations, defined as OPT1 and OPT2, respectively, while the fourth column, denoted ENC, reports the results with both optimizations. From Table 2, it can be noted that, while the plain encoder is able to reach the optimal solution on 16,7% of the instances (last row), the encoders featuring the OPT2 optimization are able to reach the optimal solution

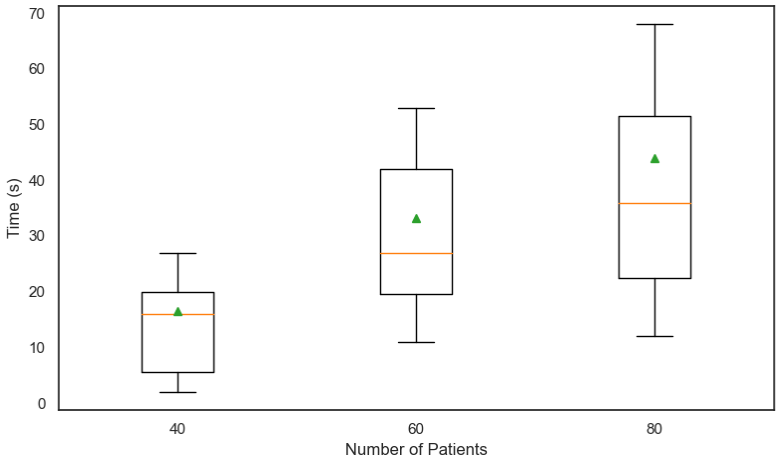

Fig. 7. Results obtained by solving 10 instances generated from the results of the first sub-problem considering 40, 60, and 80 patients. The box starts from the first quartile and ends at the third quartile. The mean time is represented by the (green) triangle, while the (orange) line represents the median value.

Table 2. Comparison of the mean time required to reach the optimal solution with the different versions of the encoding for the second sub-problem.

Total #Patients	CPU TIME PLAIN (S)	CPU TIME PLAIN+OPT1 (S)	CPU TIME PLAIN+OPT2 (S)	CPU TIME ENC (S)
40	260.7	268.1	30.4	16.6
60	283.5	295.5	57.1	33.3
80	295.3	300.0	61.9	44.1
Percentage Optimal	16,7%	16,7%	100%	100%

on all instances. Moreover, the performance increase due to the OPT1 optimization is not evident, at least with the current timeout, despite the advantages obtained in terms of number of rules generated as shown in Figure 8. The figure reports the number of rules generated by the PLAIN and by PLAIN+OPT1 options using the instances with 40 patients, and shows that the OPT1 optimization allows to decrease the number of rules generated by approx. one third. With PLAIN+OPT2, the performance increases noticeably, leading to the optimal solution in a few seconds. Finally, adding both OPT1 and OPT2 led to the better results, being able to reach an optimal solution in all the instances in less time than the other encodings. Moreover, in it can be noted that, paired with OPT2 optimization, OPT1 optimization helps further increasing the performance of the solutions.

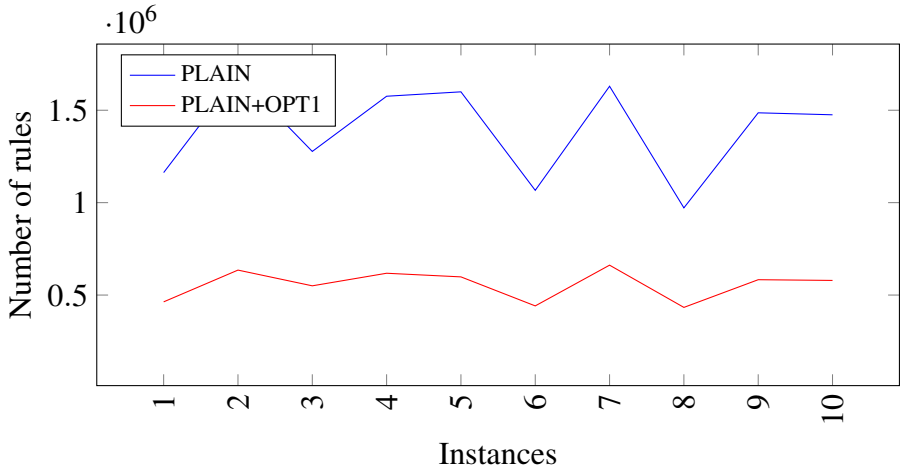

Fig. 8. Comparison between the number of rules generated using the plain encoding and using the plain encoding with the OPT1 optimization.

6.4 Comparison to alternative logic-based formalisms

In the following, we present an empirical comparison of our ASP-based solution with alternative logic-based approaches, obtained by applying automatic translations of ASP instances. In more details, we used the ASP solver WASP [3], with the option `--pre=wbo`, which converts ground ASP instances into pseudo-Boolean instances in the wbo format [39]. Then, we used the tool PYPBLIB [8] to encode wbo instances as MaxSAT instances. Moreover, in order to provide a fair comparison, given that the other formalisms/solvers can not deal with multi-levels optimizations, we also processed our ASP instances using WASP with the option `--pre=lparse`, which collapses all weak constraints levels into one single level using exponential weights. In this way, the costs found by the different approaches can be compared. We started from the instances with 40 patients for this analysis: Out of the 10 instances, we were able to effectively use 6 instances for the comparison, since on 4 instances WASP was not able to produce the corresponding single-level instance.

Then, we considered three state-of-the-art MaxSAT solvers, namely MAXHS [41], OPEN-WBO [37], and RC2 [36], and the industrial tool for solving optimization problems GUROBI [34], which is able to process instances in the wbo format. We re-run also CLINGO on the generated single-level instances, and used other than the option `restart-on-model` (CLINGO-ROM), the option `--opt-strategy=usc` (CLINGO-USC). The latter enables the usage of algorithm OLL [38], which is the same algorithm employed by the MaxSAT solver RC2, and differently from previous experiments is not dominated by CLINGO-ROM. The timeout was set to 60 seconds, given that the grounding phase and instances adaptation has been done offline.

Results are shown in Table 3, where for each solver and instance we report the time in seconds to reach the optimal solution and we write ”TIME” if it reaches the time limit with a solution, or a dash in case the solver outputs no solution within the time

Table 3. Comparison of the ASP solution to the first sub-problem with alternative logic-based solutions.

Instance	TIME (S) CLINGO-ROM	TIME (S) CLINGO-USC	TIME (S) MAXHS	TIME (S) OPEN-WBO	TIME (S) RC2	TIME (S) GUROBI
1	9.1	1.4	15.6	1.3	1.4	10.5
2	0.1	0.1	26.2	0.1	0.3	1.1
3	TIME	30.1	TIME	30.5	27.7	39.7
4	1.4	0.4	TIME	0.2	0.8	2.2
5	9.2	1.7	41.6	1.0	–	26.5
6	TIME	1.4	28.5	0.6	1.9	11.7

limit. As a general observation, CLINGO-USC, OPEN-WBO, and GUROBI are the only solvers that are able to find an optimal solution in all the tested instances. Moreover, while GUROBI finds the optimal solution on average in approx. 15 seconds, OPEN-WBO and CLING-USC obtain the best performance overall, since they are able to find the best solution on average in approx. 5 seconds. Concerning CLINGO-ROM and MAXHS, they are able to find an optimal solution in 4 instances. However, CLINGO-ROM is faster to find the optimal solutions on average. Finally, RC2 is able to find an optimal solution in 5 out of 6 instances but in the fifth instance RC2 is not able to output a solution within the time limit.

Then, we used the results obtained by the different solvers with the instances of the first sub-problem to generate 35 instances for testing the second sub-problem. The 35 instances correspond to the 31 optimal solutions we obtained testing the first sub-problem plus the 4 solutions that were anyway obtained reaching the time limit, though not (guaranteed to be) optimal. In particular, each solution obtained with the instances of the first sub-problem by the solvers represents a schedule in input for the second sub-problem. We repeated the same automatic translation we did to obtain the instances of the first sub-problem. In the following, we summarize the results. In the second sub-problem, CLINGO-USC, RC2, and OPEN-WBO are able to obtain an optimal solution within the time limit on each instance. Moreover, the average time required to solve the instances by these solvers is very similar: CLINGO-USC requires on average approx. 16 seconds, while RC2 and OPEN-WBO require on average approx. 13 seconds. Concerning the other solvers, MAXHS is able to find an optimal solution to all the instances but one while CLINGO-ROM cannot output the optimal solution to 4 out of the 35 instances tested. Finally, GUROBI cannot find the optimal solution to any of the instances tested within the time limit. We have analyzed its behavior, and we noticed that GUROBI spends, on these instances, a lot of time in the pre-solving phase: In order to find optimal solutions, the timeout should be moved to the order of tens of minutes.

Finally, we discuss advantages and reasons for the ASP approach compared to the other formalisms, to complement the analysis focused on performance above. Arguably, ASP offers a number of advantages, including: (i) The ASP specifications are often appreciated even by non-experts since they found them readable. This is important when the solution has to be used in real applications; (ii) ASP allows expressing and solving multi-level optimizations; (iii) There are free and open source systems (like CLINGO),

whose performances are often comparable (or even better) to the solvers for the other formalisms (as confirmed by our experiments).

7 Rescheduling

In this section, we present a solution to the PAC rescheduling problem, considering different scenarios in which a rescheduling could be required. In the hospital context is vital to be able to react to problems with a scheduled solution; in fact, it can happen that, due to some unpredictable events, it is not possible to implement the computed schedule. Therefore, it is necessary to find a new schedule that guarantees the proper execution of the pre-operative assessment for every patient assigned in the original scheduling. We consider three different scenarios in which rescheduling can be required:

1. Some patients do not show up on the assigned day.
2. Some operators are missing for some days.
3. Exam locations are unavailable for some days.

Thus, given a scheduling and the information of why it is not possible to implement it, the PAC rescheduling problem consists in moving patients and operators in order to find a new assignment that takes into account the new information and the constraints of the initial problem. In particular, we considered the rescheduling after the second sub-problem: This is because either we want to reschedule, in case, the result of the whole process, and, given the first sub-problem is usually done well in advance, it can be more appropriate to perform another scheduling from scratch. The optimization of the rescheduling problem consists in minimizing the changes done to the initial scheduling, according to some criteria: The priority is to maintain the day on which patients were assigned and, in case it is not possible, to minimize the distance between the new and the old assigned day. Then, if a patient is rescheduled on the same day we want to minimize the distance between the new and the old exams starting time, otherwise, since the day is changed, it is not important to maintain the same exams starting time, thus it is minimized the patient's total length of the pre-operative assessment as it was done in the second sub-problem of the PAC scheduling problem (Section 5.2). Finally, if possible, operators are assigned in the same exam areas they were initially assigned; but, while a solution that does not change the assigned day to the operators is preferred, a proper solution must still activate all the exam areas required by the patients. Thus, operators can be moved from a day to another, in order to activate the required exam areas. Moreover, in case of unavailable operators, if it is possible to replace the operator so that the exam area in which (s)he was assigned can still be activated then no changes are needed; otherwise, patients requiring that exam area must be rescheduled in a new day.

7.1 ASP encoding of the rescheduling

Data Model. The input data is the same of the PAC scheduling problem plus the following atoms:

```
1  allowedDay(RID,DAY):- reg(RID,_,DUEDATE), time(DAY,_), not forbidden(RID,DAY), DAY < DUEDATE.
2  allowedTime(RID,AREAID,ST..60-ET) :-
       forbiddenBefore(RID,AREAID,ST),forbiddenAfter(RID,AREAID,ET).
3  changedDay(DAY) :- y(RID,_,_,_,DAY), x(RID,_,_,_,DAY1), DAY != DAY1.
4  changedDay(DAY) :- changedExamLoc(_,DAY).
5  changedDay(DAY) :- changedOperator(_,DAY).
6  1{y(RID,AREAID,ST,ST+DUR,DAY): allowedTime(RID,AREAID,ST), allowedDay(RID,DAY)}1 :-
       reg(RID,_,DUEDATE), exam(RID,AREAID,DUR), forbidden(RID,_).
7  1{y(RID,AREAID,ST,ST+DUR,DAY): allowedTime(RID,AREAID,ST), allowedDay(RID,DAY)}1 :-
       reg(RID,_,DUEDATE), exam(RID,AREAID,DUR), not forbidden(RID,_), x(RID,_,_,_,DAY1),
       changedDay(DAY1).
8  y(RID,AREAID,ST,ET,DAY) :- x(RID,AREAID,ST,ET,DAY), not changedDay(DAY), not
       forbidden(RID,_).
9  :- y(RID,_,_,_,DAY1), y(RID,_,_,_,DAY2), DAY1 > DAY2.
10 :~ y(RID,_,_,_,DAY),x(RID,_,_,_,DAYX), changedDay(DAY). [|DAYX-DAY|@4,RID]
11 :~ y(RID,AREAID,STy,_,DAY),x(RID,AREAID,STx,_,DAY). [|STx-STy|@3,RID,AREAID]
12 :~ y(RID,first_exam,ST,_,_), y(RID,last_exam,_,ET,_), changedDay(DAY), cost(RID,TOT),
       ET-ST-TOT >= 0. [ET-ST-TOT@2,RID]
13 :~ operatorY(ID,AREAID1,DAY), operators(ID,AREAID2,DAY), AREAID1 != AREAID2. [1@1,ID,DAY]
```

Fig. 9. ASP encoding of the rescheduling problem.

- Instances of `reg(RID,PR,DUEDATE)` represent the registration of a patient, characterized by an id (`RID`), the priority level (`PR`), and the duedate (`DUEDATE`) (i.e., the input of the scheduling without TARGET, which is not anymore useful, and TOT_DUR, which is not anymore valid).
- Instances of `x(RID,AREAID,ST,ET,DAY)` represent the previous scheduling, i.e., the output of the encoding in Section 5.2, without the priority field.
- Instances of `operator(ID,AREAID,DAY)` represent the previous scheduling, i.e., the output of the encoding in Section 5.1.
- Instances of `forbidden(RID,DAY)` represent the day (`DAY`) in which a patient (`RID`) can not show up.
- Instances of `notAvailableExamLoc(AREAID,DAY)` represent the day (`DAY`) in which an exam location (`AREAID`) is not available.
- Instances of `notAvailableOperator(ID,DAY)` represent the day (`DAY`) in which an operator (`ID`) can not show up.

Output. The output is similar to the output of Section 5.2, where the new schedule is represented by two atoms, `y(RID,AREAID,ST,ET,DAY)` and `operatorY(ID,AREAID,DAY)`, having the same meaning of atoms `x` and `operator` of Section 5.2.

Encoding. The ASP encoding of the PAC rescheduling problem we realized is made by the rules in Figure 9, where we denote with r_i rule appearing at the line i, plus rules r_5 and r_6 from Figure 2, where atom `operatorY` replaces atom `operator`, rules r_2-r_4 from Figure 3 (corresponding to the constraints of the second sub-problem), rules r_6 and r_7 from Figure 4 (for computing atoms `forbiddenAfter` and `forbiddenBefore` in the first optimization), and rule r_9 in Figure 5 (corresponding to the second optimization), where atom y replaces atom x.

In Figure 9, rules r_1 and r_2 define two auxiliary atoms that represent the possible days in which a registration can be assigned and the possible starting times of the

exams, respectively. Rules from r_3 to r_5 define an auxiliary atom (`changeDay`) that represents the days in which there are changes due to patients or operators (un)availability or changes in the availability of the exam locations. Rule r_6 assigns a new day and a new starting time to all the exams required by the registrations that can not show up in the previously assigned day. Rule r_7 assigns a new day and new starting time to all the exams required by the registrations that were assigned in a day involved in the changes. Rule r_8 reassigns the same day and starting time to all exams of the still available registrations that were assigned in a day without changes. Then, rule r_9 ensures that, for each registration, all his/her exams are assigned in the same day. Finally, weak constraints r_{10}-r_{13} scecify the optimization in a prioritized way. Weak constrains r_{10} and r_{11} minimize the distance between the old and the new date assigned to each registration, and between the old and the new starting time assigned to every exam, respectively. Whereas, weak constraint r_{12} is used to minimize the time spent in the hospital by patients, while weak constraint r_{13}, which has the lowest priority level, minimizes the number of operators assigned to different exam locations than in the previous scheduling.

7.2 Results of the rescheduling

Here, the results of an empirical analysis of the PAC rescheduling problem are presented. As for the PAC scheduling problem, the ASP system used was CLINGO [28], ver. 5.4.0, using parameters *--restart-on-model* for faster optimization and *--parallel-mode 6* for parallel execution. The time limit was set to 60 seconds.

Since the rescheduling problem consists of, given a scheduling that cannot be implemented, moving operators and patients to find a new assignment, the initial scheduling was taken from all the results of the second sub-problem run with CLINGO considering the benchmarks with 40 patients. We started from the results obtained in the second sub-problem with 40 patients. Then, according to the different scenarios, we added the unavailability of patients, exam locations, or operators using the atoms `forbidden(RID,DAY)`, `notAvailableExamLoc(AREAID,DAY)`, and `notAvailableOperator(ID,DAY)` as presented in Section 7.1. In particular, an unavailable operator will be set unavailable for 5 consecutive days, while an unavailable exam location will be set unavailable for 3 consecutive days. To test the scalability of the solution, we considered for each scenario 1, 2, and 4 among exam locations, operators or patients unavailable. The unavailability were randomly selected between the assigned one in the original scheduling.

Results for the three identified scenarios are described in the next three paragraphs, while a fourth paragraph is devoted to present an example of the rescheduling.

First scenario: Patients require a new day. In this scenario, it is not possible for one (or more) patient(s) to have the pre-operative assessment on his/her planned day of the initial scheduling.

We tested the 10 instances with 1, 2, and 4 patients to be rescheduled. With 1 patient unavailable, the solution is able to find the optimal solution in 7 out of 10 instances tested; while, with 2 and 4 patients unavailable, the solution finds the optimal solution in 2 out of 10 instances. About the timings, with 1 patient to reschedule, the solution

Table 4. Average time required to obtain the optimal solution in the different scenarios.

Resource Unavailable	TIME (S) SCENARIO 1	TIME (S) SCENARIO 2	TIME (S) SCENARIO 3
1	31.0	27.1	19.0
2	49.5	31.2	26.2
4	51.6	49.6	48.6

was able to reach the optimal solution on average in 31 seconds, while, with 2 and 4 patients, the results are obtained on average in approx. 49 and 51 seconds, respectively.

Second scenario: Operators are not available. In this scenario, it is considered the case in which an operator (or more) is (are) not available.

We tested the 10 instances with 1, 2, and 4 operators not available in days in which they were scheduled. With both 1 and 2 operators unavailable, the rescheduler is able to find the optimal solution in 7 out of 10 instances and to find such solution in approx. 27 and 31 seconds on average, respectively. Increasing to 4 the number of unavailable operators leads to more patients involved in the rescheduling, thus, the number of instances in which the rescheduler finds the optimal solution decreases to 2, with an average time required to find such solutions of approx. 49 seconds.

Third scenario: Exam areas are not available. In this scenario, it is supposed that an (or more) exam area(s) is (are) not available.

We tested this scenario with the 10 instances and considering the unavailability of 1, 2, and 4 exam areas. With 1 exam area unavailable, the solution obtains the optimal solution in 7 out of 10 instances and the solution is obtained on average in 19 seconds. Increasing the unavailable exam areas and testing the solution with 2 unavailable exam areas the rescheduler still gets the optimal solution in 6 out of 10 instances, while, with 4 unavailable exam areas it finds the optimal solution in just 2 out of the 10 instances. The average CPU times for computing optimal solutions for these last two cases are around 26 and 48 seconds, respectively.

Example of rescheduling. We performed an analysis starting from the schedule in Figure 1 in which three patients have been rescheduled in the day shown due to the unavailability of the exam location with id equal to 3 required by them, whose result is shown in Figure 10. The three patients were scheduled originally the day before Figure 1, thus, they are rescheduled on the first available day to minimize the distance between the original and the new schedule. Moreover, the image shows that there is a time slot of waiting time for patient 18. This waiting time cannot be reduced because the patients wouldn't be able to fully complete all the exams without overlapping and the rescheduler prefers to reassign this patient with one time slot of waiting time rather than move one patient to another day, i.e., minimizing the distance between the original day and the new one has higher priority than minimizing the waiting times.

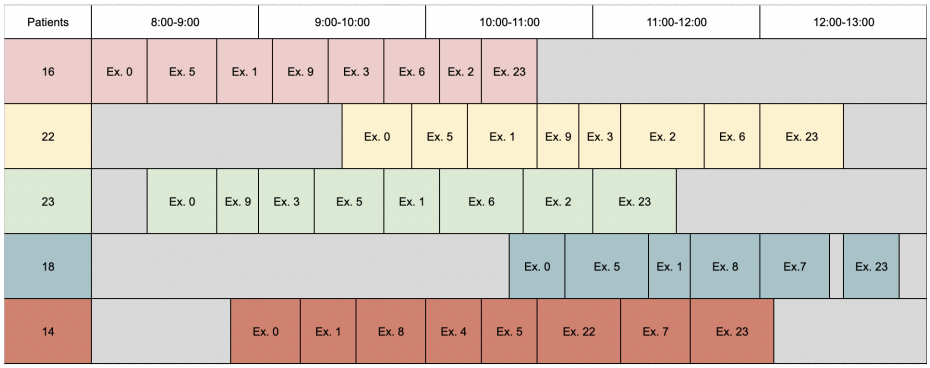

Fig. 10. Rescheduling example related to registrations 16, 22, and 23 of Figure 1 rescheduled to a new day because of unavailability of exam location 3.

8 Web Application

In view of a practical application, the solution provided in this article should be accessible to users with no or low knowledge of ASP. Thus, we developed a web application that allows any hospital operators to interact with our solution and understand the results through a graphical interface.

The system architecture is reported in Figure 11. The graphical interface has been developed using the frameworks Angular for the frontend, Node.js for the backend, and GraphQL for data queries. We also used a wrapper inside the backend which calls the solver CLINGO. Planning results are then stored in a database. The workflow of the architecture is the following:

1. the user selects a period to plan through the frontend; data are shown according to the selection, and then (s)he can start the first planning. The result of the first planning represents the solution of the first sub-problem of the PAC scheduling problem.
2. Once the first planning is completed, the result is stored and shown. Then, the user has to select, from a default list of exams, all the exams necessary for each patient. Now, the detailed planning can start. This corresponds to the second sub-problem of the PAC problem.
3. The final scheduling is stored and shown with the precise starting time of each exam.

These three steps are further elaborated in the next three paragraphs.

Data preparation and first sub-problem. In Figure 12 the first page is shown: Here the user can select the period to plan and the solver timeout. In this screen, data are read-only because in a real-world application data should be fed through direct integration with the hospital system and its database. Three tabs then allow the user to check the elements involved in the planning:

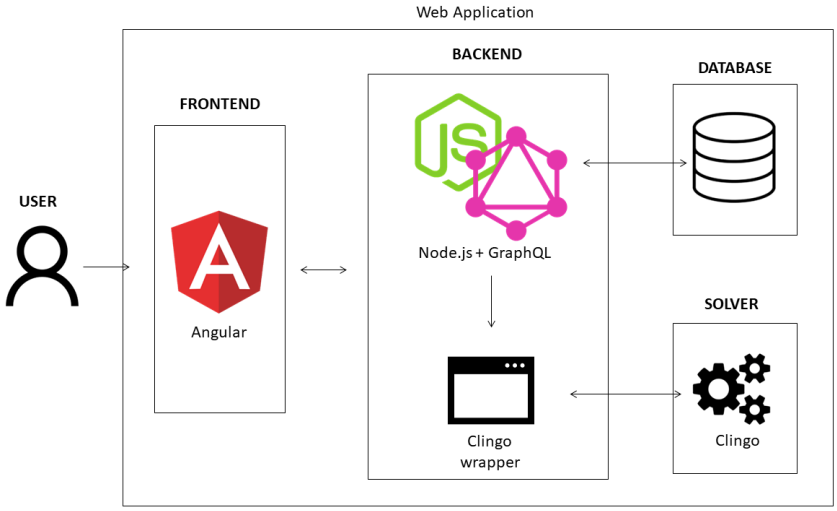

Fig. 11. Representation of the web application architecture, together with the employed technologies.

- Patients tab, to monitor all the patients with a target date included in the planning period (Figure 12).
- Personnel tab, to monitor the operators available in the selected period (Figure 13). A drop-down menu allows to select which operator to inspect.
- Providers tab, to check all the exam areas enabled in the selected time period (Figure 14). A drop-down menu allows to select the exam area to inspect.

Then, by clicking the ”START PLANNING” button, the solver is called through the wrapper, and the solution of the first PAC sub-problem is calculated. A loading screen will appear and once the solver finishes the planning (optimum found or timeout reached), the user is redirected to the next page in which the results are shown.

Second sub-problem. Here the user can look at the results of the first sub-problem, each schedule can be selected through a drop down menu. At this point, the user has to select the final list of exams required by each patient. Thus, by clicking on the edit icon a menu will appear listing all the patient’s exams where a check box allows to select which exams are necessary, as shown in Figure 15.

Moreover, a second tab called ”Graphics” displays the results obtained through two kinds of graphs: A bar chart (Figure 16) in which the bars represent the number of patients assigned to each day of the scheduled period, and four doughnuts charts (Figure 17) that represent the percentage of patients scheduled over the total number of patients, grouped by the patients’ priority. Finally, by clicking on the ”START PLANNING” button, again the solver is called on the second sub-problem and the detailed planning with the precise starting time of all the exams is computed. Once the solver terminates, the user is redirected to the final page.

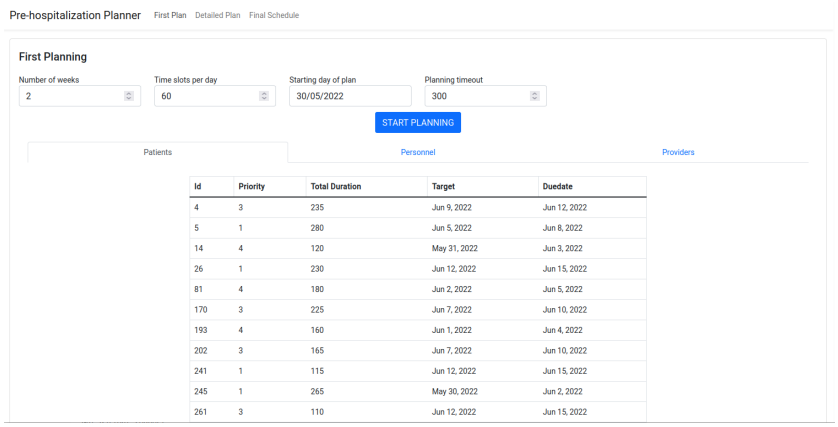

Fig. 12. Data preparation.

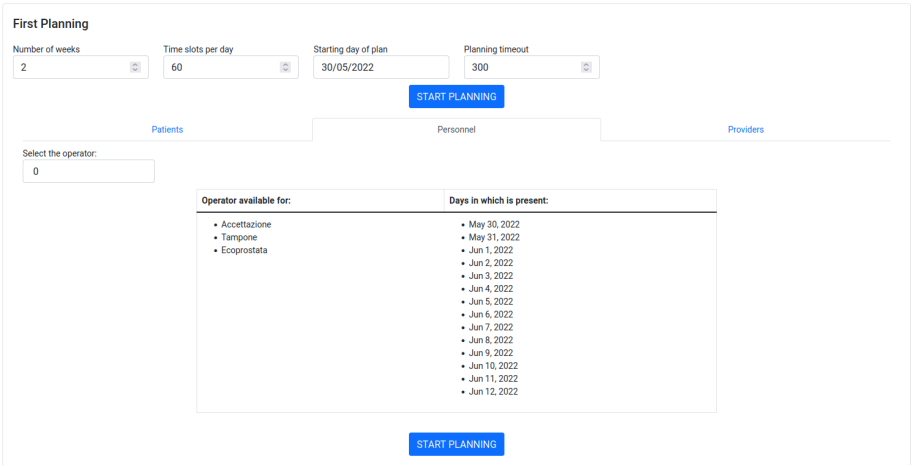

Fig. 13. Personnel tab.

Final schedule. As shown in Figure 18, the final schedule page is made up of a drop down menu, from which it can be selected the scheduled days, and two tabs: The patients info tab and the graphics tab. In the patients info tab, patients are sorted in a table and by clicking on the icon in the ”Show Exams” column a table will appear listing all the exams and their starting time. Figure 19, instead, reports a chart shown in the graphics tab, in which on the x-axis there is the timeline and on the y-axis the patients, and each segment corresponds to an exam. A similar chart in the graphics tab contains the exam locations on the y-axis.

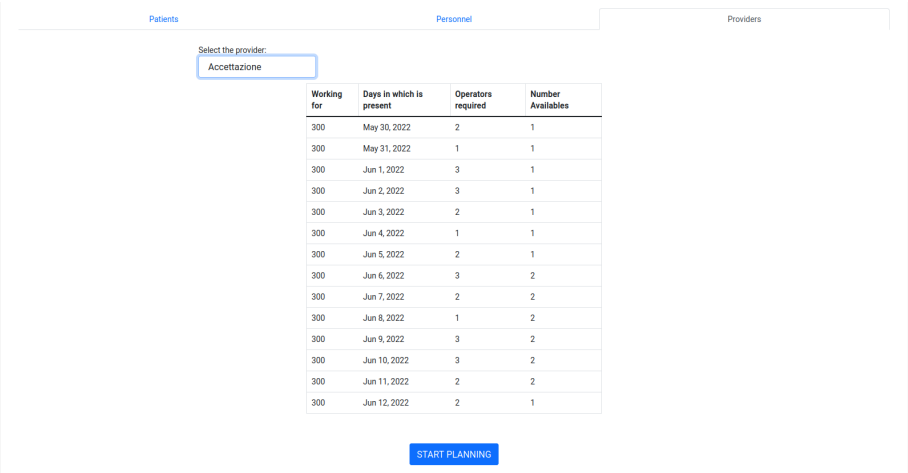

Fig. 14. Providers tab

9 Related Work

This paper is an extended and revised version of a paper appearing in the LNCS post-proceedings of the 20th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2021) [16], having the following main improvements: (i) an improved encoding for the first sub-problem, with more patients assigned (Section 5.1), (ii) a comparison to alternative logic-based formalisms (Section 6.4), (iii) a rescheduling solution and related experimental analysis (Section 7), and (iv) a web application to easy the usage of our scheduling solution (Section 8).

The rest of the section is organized in two paragraphs: The first is focused on alternative methods for solving the PAC problem, while the second mentions works in which ASP has been employed in scheduling problems.

Solving the PAC problem. Edward at al. [23] used two simulation models to analyse the difficulties of planning in the context of PAC and to determine the resources needed to reduce waiting times and long access times. The models were tested in a large university hospital and the results were validated by measuring the level of patients's satisfaction. Stark et al. [43] used a Lean quality improvement process changing the process and the standard routine. For example, patients were not asked to move from one room to another for the visits, but patients were placed in a room, and remained there for the whole duration of their assessment. This and other changes to the processes led to a decrease in the average lead time for patients and to the number of patients required to return the next day to complete the visits. The authors of [35, 27, 17, 44, 45] studied the importance of implementing the PAC problem and the positive results obtained by having less waiting time between the exams and for the visit to the hospital. In particular, while different clinics follow different guidelines, implementing PAC has proved to be

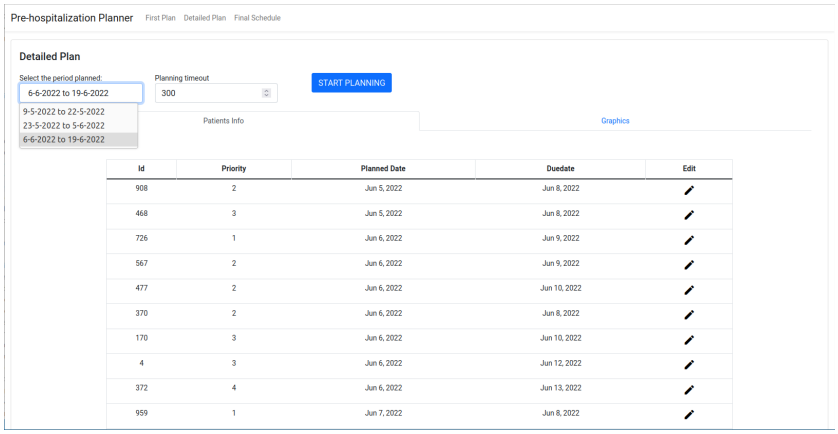

Fig. 15. Edit patient exams.

Fig. 16. Bar chart.

an important tool to avoid the cancellation of surgeries and to significantly reduce the risk associated with the surgery.

Differently from the works above mentioned, the solution we proposed in this paper focuses on an actual scheduling of the PAC with real requirements, making a practical system able to solve the PAC scheduling and rescheduling problems. Moreover, our solution guarantees a certain level of usability; in fact, through the two-phase approach and the web application that we developed, we can support the users in computing the scheduling.

As it can be seen, all works mentioned are about the scheduling problem. To the best of our knowledge, there is no paper dealing with the rescheduling PAC problem: Our work is thus the first one which proposes and implements a rescheduling PAC problem and related solution.

Fig. 17. Doughnut chart.

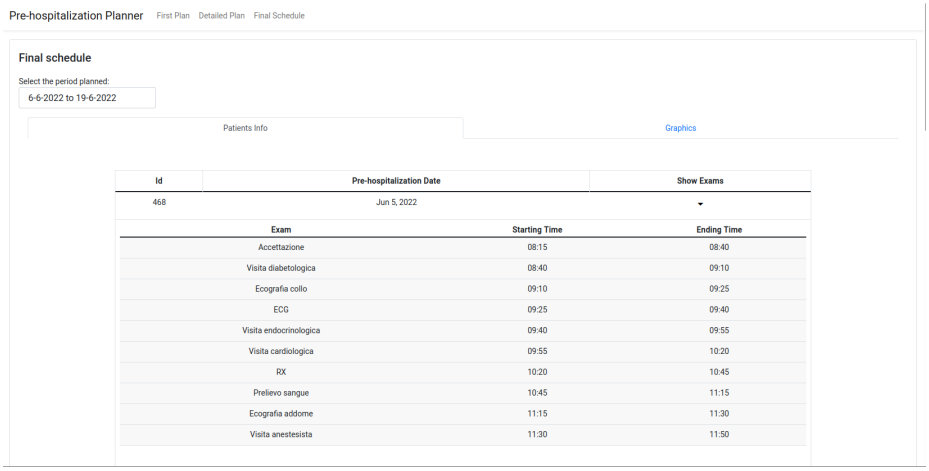

Fig. 18. Final schedule page.

Solving scheduling problems with ASP. ASP has been successfully used for solving hard combinatorial and application scheduling problems in several research areas. In the Healthcare domain (see, e.g., [4] for a recent survey), the first solved problem was the *Nurse Scheduling Problem* [5, 22, 6], where the goal is to create a scheduling for nurses working in hospital units. Then, the problem of assigning operating rooms to patients, denoted as *Operating Room Scheduling* [20, 21], has been treated, and further extended to include bed management [19]. More recent problems include the *Chemotherapy Treatment Scheduling* problem [18], in which patients are assigned a chair or a bed for their treatments, and the *Rehabilitation Scheduling Problem* [15], which assigns patients to operators in rehabilitation sessions. The current paper is the only one which deals with the pre-operative phase; moreover, it employs a two-phase approach. In [14], it is proposed a solution using ASP to the problem of assigning a date to visit or ther-

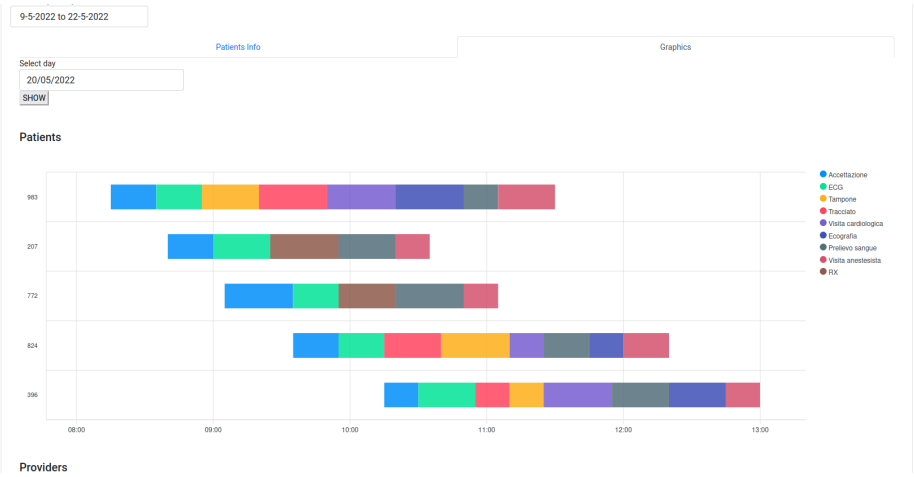

Fig. 19. Final schedule page, graphics tab, patients chart

apy for multiple recurrent exams to chronic patients. As in our work, the problem is split into two sub-problems. The main difference between the two works is that in our the two sub-problems are solved at different times, while in [14] the problem is split into two sub-problems to increase the performance of the solution using the Benders's decomposition method.

Concerning scheduling problems beyond the Healthcare domain, ASP encoding were proposed for the following problems: *Incremental Scheduling Problem* [9], where the goal is to assign jobs to devices such that their executions do not overlap one another; *Team Building Problem* [40], where the goal is to allocate the available personnel of a seaport for serving the incoming ships; and the *Conference Paper Assignment Problem* [7], which deals with the problem of assigning reviewers in the Program Committee to submitted conference papers. Other relevant papers are Gebser et. al [33], where, in the context of routing driverless transport vehicles, the setup problem of routes such that a collection of transport tasks is accomplished in case of multiple vehicles sharing the same operation area is solved via ASP, in the context of car assembly at Mercedes-Benz Ludwigsfelde GmbH, and the recent survey paper by Falkner et al. [26], where industrial applications dealt with ASP are presented, including those involving scheduling problems.

10 Conclusion

In this paper, we have presented an analysis of the PAC scheduling problem modeled and solved with ASP. We started from a mathematical formulation of the problem, whose specifications come from a real scenario, and then presented our ASP solution. Results on synthetic data shows that the solution is able to assign a high number of patients with high priority, and compares well to other logic-based formalisms. We have then modeled and solved the PAC rescheduling problem, which comes into play when

the computed scheduling can not be implemented due to some unavailability. We finally presented a web application for easy usage of our scheduling solution. Possible topics for future research include testing our solution with real data (we remind that, through the specifications used are real, data have been randomly generated), when available, and with other ASP solvers such as WASP [2]. Another option would be to test the CLINGO multi-shot approach [29] on the first sub-problem.

References

1. Abels, D., Jordi, J., Ostrowski, M., Schaub, T., Toletti, A., Wanko, P.: Train scheduling with hybrid ASP. In: LPNMR. Lecture Notes in Computer Science, vol. 11481, pp. 3–17. Springer (2019)
2. Alviano, M., Amendola, G., Dodaro, C., Leone, N., Maratea, M., Ricca, F.: Evaluation of disjunctive programs in WASP. In: Balduccini, M., Lierler, Y., Woltran, S. (eds.) LPNMR. LNCS, vol. 11481, pp. 241–255. Springer (2019)
3. Alviano, M., Amendola, G., Dodaro, C., Leone, N., Maratea, M., Ricca, F.: Evaluation of disjunctive programs in WASP. In: LPNMR 2019. LNCS, vol. 11481, pp. 241–255. Springer (2019). https://doi.org/10.1007/978-3-030-20528-7_18, `https://doi.org/10.1007/978-3-030-20528-7\_18`
4. Alviano, M., Bertolucci, R., Cardellini, M., Dodaro, C., Galatà, G., Khan, M.K., Maratea, M., Mochi, M., Morozan, V., Porro, I., Schouten, M.: Answer set programming in healthcare: Extended overview. In: IPS and RCRA 2020. CEUR Workshop Proceedings, vol. 2745. CEUR-WS.org (2020), `http://ceur-ws.org/Vol-2745/paper7.pdf`
5. 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)
6. Alviano, M., Dodaro, C., Maratea, M.: Nurse (re)scheduling via answer set programming. Intelligenza Artificiale **12**(2), 109–124 (2018)
7. Amendola, G., Dodaro, C., Leone, N., Ricca, F.: On the application of answer set programming to the conference paper assignment problem. In: AI*IA. Lecture Notes in Computer Science, vol. 10037, pp. 164–178. Springer (2016)
8. Ansótegui, C., Pacheco, T., Pon, J.: Pypblib (2019), `https://pypi.org/project/pypblib/`
9. Balduccini, M.: Industrial-size scheduling with ASP+CP. In: Delgrande, J.P., Faber, W. (eds.) Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6645, pp. 284–296. Springer (2011). https://doi.org/10.1007/978-3-642-20895-9_33, `https://doi.org/10.1007/978-3-642-20895-9\_33`
10. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011)
11. Buccafurri, F., Leone, N., Rullo, P.: Enhancing Disjunctive Datalog by Constraints. IEEE Transactions on Knowledge and Data Engineering **12**(5), 845–860 (2000)
12. 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`
13. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: The design of the fifth answer set programming competition. CoRR **abs/1405.3710** (2014), `http://arxiv.org/abs/1405.3710`
14. Cappanera, P., Gavanelli, M., Nonato, M., Roma, M.: A decomposition approach to the clinical pathway deployment for chronic outpatients with comorbidities. In: Amorosi, L., Dell'Olmo, P., Lari, I. (eds.) Optimization in Artificial Intelligence and Data Sciences. pp. 213–226. Springer International Publishing, Cham (2022)

15. Cardellini, M., Nardi, P.D., Dodaro, C., Galatà, G., Giardini, A., Maratea, M., Porro, I.: A two-phase ASP encoding for solving rehabilitation scheduling. In: Moschoyiannis, S., Peñaloza, R., Vanthienen, J., Soylu, A., Roman, D. (eds.) Proceedings of the 5th International Joint Conference on Rules and Reasoning (RuleML+RR 2021). Lecture Notes in Computer Science, vol. 12851, pp. 111–125. Springer (2021)
16. Caruso, S., Galatà, G., Maratea, M., Mochi, M., Porro, I.: An ASP-based approach to scheduling pre-operative assessment clinic. In: Bandini, S., Gasparini, F., Mascardi, V., Palmonari, M., Vizzari, G. (eds.) Proc. of AIxIA 2021 - Advances in Artificial Intelligence - 20th International Conference of the Italian Association for Artificial Intelligence, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13196, pp. 671–688. Springer (2021)
17. Correll, D., Bader, A., Hull, M., Hsu, C., Tsen, L., Hepner, D.: Value of Preoperative Clinic Visits in Identifying Issues with Potential Impact on Operating Room Efficiency. Anesthesiology **105**(6), 1254–1259 (12 2006). https://doi.org/10.1097/00000542-200612000-00026, `https://doi.org/10.1097/00000542-200612000-00026`
18. Dodaro, C., Galatà, G., Grioni, A., Maratea, M., Mochi, M., Porro, I.: An ASP-based solution to the chemotherapy treatment scheduling problem. Theory and Practice of Logic Programming **21**(6), 835–851 (2021)
19. Dodaro, C., Galatà, G., Khan, M.K., Maratea, M., Porro, I.: An ASP-based solution for operating room scheduling with beds management. In: Fodor, P., Montali, M., Calvanese, D., Roman, D. (eds.) Proceedings of the Third International Joint Conference on Rules and Reasoning (RuleML+RR 2019). Lecture Notes in Computer Science, vol. 11784, pp. 67–81. Springer (2019)
20. Dodaro, C., Galatà, G., Maratea, M., Porro, I.: Operating room scheduling via answer set programming. In: AI*IA. LNCS, vol. 11298, pp. 445–459. Springer (2018)
21. Dodaro, C., Galatà, G., Maratea, M., Porro, I.: An ASP-based framework for operating room scheduling. Intelligenza Artificiale **13**(1), 63–77 (2019)
22. Dodaro, C., Maratea, M.: Nurse scheduling via answer set programming. In: LPNMR. LNCS, vol. 10377, pp. 301–307. Springer (2017)
23. Edward, G.M., Das, S.F., Elkhuizen, S.G., Bakker, P.J.M., Hontelez, J.A.M., Hollmann, M.W., Preckel, B., Lemaire, L.C.: Simulation to analyse planning difficulties at the preoperative assessment clinic. BJA: British Journal of Anaesthesia **100**(2), 195–202 (02 2008)
24. Erdem, E., Gelfond, M., Leone, N.: Applications of answer set programming. AI Magazine **37**(3), 53–68 (2016)
25. Faber, W., Pfeifer, G., Leone, N.: Semantics and complexity of recursive aggregates in answer set programming. Artificial Intelligence **175**(1), 278–298 (2011)
26. Falkner, A.A., Friedrich, G., Schekotihin, K., Taupe, R., Teppan, E.C.: Industrial applications of answer set programming. Künstliche Intelligenz **32**(2-3), 165–176 (2018)
27. Ferschl, M., Tung, A., Sweitzer, B., Huo, D., Glick, D.: Preoperative Clinic Visits Reduce Operating Room Cancellations and Delays. Anesthesiology **103**(4), 855–859 (10 2005)
28. 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)
29. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. Theory Pract. Log. Program. **19**(1), 27–82 (2019)
30. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artificial Intelligence **187**, 52–89 (2012)
31. Gebser, M., Maratea, M., Ricca, F.: The design of the seventh answer set programming competition. In: Balduccini, M., Janhunen, T. (eds.) LPNMR. Lecture Notes in Computer Science, vol. 10377, pp. 3–9. Springer (2017)

32. Gebser, M., Maratea, M., Ricca, F.: The seventh answer set programming competition: Design and results. Theory and Practice of Logic Programming **20**(2), 176–204 (2020)
33. Gebser, M., Obermeier, P., Schaub, T., Ratsch-Heitmann, M., Runge, M.: Routing driverless transport vehicles in car assembly with answer set programming. Theory and Practice of Logic Programming **18**(3-4), 520–534 (2018). https://doi.org/10.1017/S1471068418000182, https://doi.org/10.1017/S1471068418000182
34. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2021), https://www.gurobi.com
35. Harnett, M.P., Correll, D., Hurwitz, S., Bader, A., Hepner, D.: Improving Efficiency and Patient Satisfaction in a Tertiary Teaching Hospital Preoperative Clinic. Anesthesiology **112**(1), 66–72 (01 2010)
36. Ignatiev, A., Morgado, A., Marques-Silva, J.: RC2: an efficient maxsat solver. J. Satisf. Boolean Model. Comput. **11**(1), 53–64 (2019). https://doi.org/10.3233/SAT190116, https://doi.org/10.3233/SAT190116
37. Martins, R., Manquinho, V.M., Lynce, I.: Open-wbo: A modular maxsat solver,. In: SAT 2014. LNCS, vol. 8561, pp. 438–445. Springer (2014). https://doi.org/10.1007/978-3-319-09284-3_33, https://doi.org/10.1007/978-3-319-09284-3_33
38. Morgado, A., Dodaro, C., Marques-Silva, J.: Core-Guided MaxSAT with Soft Cardinality Constraints. In: CP 2014. pp. 564–573. Springer, Lyon, France (September 2014)
39. Olivier Roussel and Vasco Manquinho: Input/Output Format and Solver Requirements for the Competitions of Pseudo-Boolean Solvers (2012), https://www.cril.univ-artois.fr/PB12/format.pdf
40. 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. Theory and Practice of Logic Programming **12**(3), 361–381 (2012)
41. Saikko, P., Berg, J., Järvisalo, M.: LMHS: A SAT-IP hybrid maxsat solver. In: SAT 2016. LNCS, vol. 9710, pp. 539–546. Springer (2016). https://doi.org/10.1007/978-3-319-40970-2_34, https://doi.org/10.1007/978-3-319-40970-2_34
42. Schüller, P.: Answer set programming in linguistics. Künstliche Intelligence **32**(2-3), 151–155 (2018). https://doi.org/10.1007/s13218-018-0542-z, https://doi.org/10.1007/s13218-018-0542-z
43. Stark, C., Gent, A., Kirkland, L.: Improving patient flow in pre-operative assessment. BMJ Open Quality **4**(1) (2015). https://doi.org/10.1136/bmjquality.u201341.w1226
44. Tariq, H., Ahmed, R., Kulkarni, S., Hanif, S., Toolsie, O., Abbas, H., Chilimuri, S.: Development, functioning, and effectiveness of a preoperative risk assessment clinic. Health Services Insights **2016**, 1 (10 2016). https://doi.org/10.4137/HSI.S40540
45. Woodrum, C.L., Wisniewski, M., Triulzi, D.J., Waters, J.H., Alarcon, L.H., Yazer, M.H.: The effects of a data driven maximum surgical blood ordering schedule on preoperative blood ordering practices. Hematology **22**(9), 571–577 (2017)