Nurse (Re)scheduling Via Answer Set Programming [1]

Mario Alviano [a], Carmine Dodaro [b,*] and Marco Maratea [b]

[a] *Department of Mathematics and Computer Science, University of Calabria, Ponte P. Bucci Cubo 30B, 87036, Arcavacata di Rende (CS), Italia*
E-mail: alviano@mat.unical.it

[b] *Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genova, Viale F. Causa 15, 16145, Genova (GE), Italia*
E-mail: {dodaro,marco}@dibris.unige.it

Abstract. The goal of the Nurse Scheduling Problem is to find an assignment of nurses to shifts according to specific requirements. Frequently, a computed schedule may become not usable because of sudden absences of some nurses. In this cases, Nurse Rescheduling amounts to the computation of a new schedule, which has to satisfy the original requirements and the new absences. Additionally, a good solution to the Nurse Rescheduling Problem must be as similar as possible to the original schedule, which practically means that the number of changes has to be minimized. This paper focuses on the requirements specified by an Italian hospital, and recently addressed by an approach based on Answer Set Programming (ASP). Even if promising results have been obtained with ASP, the original encoding presents some intrinsic weaknesses, which are identified and eventually circumvented in this paper. The new encoding is designed by taking into account both intrinsic properties of Nurse Scheduling and internal details of ASP solvers, such as cardinality and weight constraint propagators. The performance gain of CLINGO and WASP is empirically verified on instances from ASP literature. As an additional contribution, the performance of CLINGO and WASP is compared to other declarative frameworks, namely SAT and ILP; the best performance is obtained by CLINGO running the new ASP encoding. The advanced ASP encodings are then extended to solve Nurse Rescheduling, and an empirical evaluation is conducted with CLINGO and WASP.

1. Introduction

The Nurse Scheduling Problem (NSP) consists of generating a schedule of working and rest days for nurses employed in hospital units. The schedule should determine the shift assignments of nurses for a predetermined window of time, and must satisfy requirements imposed by the Rules of Procedure of hospitals. A proper solution to the NSP is crucial to guarantee the high level of quality of health care, to improve the degree of satisfaction of nurses, and the recruitment of qualified personnel. Given its practical relevance of the quality of hospital structures, NSP has been widely studied in the literature, and several variants have been considered [16,20]. Such variants are usually grouped according to several factors, as the planning period, the different types of shifts considered, and requirements on the preferences of hospitals and nurses.

The NSP variant considered in this paper concerns a planning period fixed to one year with three different types of shifts (morning, afternoon and night) and requirements on nurses and hospitals provided by an Italian hospital. Specifically, such requirements concern restrictions to the number of working hours per year and to the number of times nurses are assigned to a specific shift.

However, in hospital units that operate 24 hours a day, 7 days for week, it is frequent that one planned

[1]This paper includes parts and significantly extends our previous work [7].
[*]Corresponding author. E-mail: dodaro@dibris.unige.it.

schedule cannot be fulfilled due to sudden absences of nurses. The Nurse Rescheduling Problem (NRP), instead, occurs when one or more nurses notify their unavailability to attend one or more scheduled shifts, and aims at finding a new schedule taking into such unavailability, at the same time minimizing the differences with a previous computed schedule. It is important to emphasize here that rescheduling is usually independent from the quality of the schedule, and it is usually due to unpredictable events, e.g. illness of one or more nurses. When a sudden absence occurs, the planned schedule must be modified to ensure a good quality of the health care service. On the other hand, employees usually tend to organize their free time, thus modifications of a previously announced schedule may create personal inconveniences and may not be very well accepted by the workforce. Therefore, typically the schedule cannot be completely rebuilt from scratch. Instead, the goal should be to determine a new feasible scheduling minimizing the number of shift changes with respect to the previous schedule. Moreover, as argued in [21], rescheduling is a quite frequent activity and often requires reactive and immediate decisions. Thus, any approach aiming at solving the NRP should consider this additional requirement of being efficiently computed.

Complex combinatorial problems, such as NSP and NRP, are usually the target for the application of logic formalisms such as Answer Set Programming (ASP) [15]. Its simple syntax [17] and the intuitive semantics [31], combined with the availability of robust implementations (see, e.g. [3,28]), 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 [10,23], Bioinformatics [25,35], Hydroinformatics [26], Databases [38], and also employed in industrial applications [1,24]. Recently, the aforementioned variant of NSP has been modeled by means of an ASP encoding presented in [22]. The encoding resulted to be natural and intuitive, in the sense that it was designed by applying the standard modeling methodology, yet obtaining reasonable performance on solving the analyzed instances.

On the other hand, it turned out that the encoding presented in [22] shows some limitations and intrinsic weaknesses, mainly due to *aggregates* [4], i.e. operations on multi-sets of weighted literals that evaluate to some value. Indeed, the encoding exploits some aggregates with a quite large number of literals and few different weights, resulting to be counterproductive for the performance of modern ASP solvers [29], since they decrease their propagation power.

In this paper, we circumvented such limitations by taking into account only combinations of values that can lead to admissible schedules. Interestingly, the new encoding did not require to significantly sacrifice readability, as it remains intuitive and clear. Moreover, the new encoding is used as basis for an ASP-based solution for NRP.

The performance of the ASP solvers executed on the new encoding for NSP has been empirically evaluated on the same data and settings used in [22], showing a clear improvement in performance of state-of-the-art ASP systems CLINGO [28] and WASP [5]. As an additional contribution, the ASP-based approaches have been compared to other declarative frameworks, namely Propositional Logic Satisfiability (SAT) [8] and Integer Linear Programming (ILP) [32]. Results show that CLINGO and WASP executed on the new encoding outperform their counterparts executed on the original encoding. In particular, CLINGO executed on the new encoding is considerably faster than all other tested approaches.

The encoding for NRP has been tested by considering one previous schedule, and simulating sudden absences for nurses. Results on 150 random scenarios show that state-of-the-art ASP systems CLINGO and WASP can compute a solution for the NRP *in few seconds*.

The contributions of the paper can be summarized as follows:

1. We formalize the variant of NSP considered in [22] (Section 4.1).
2. We formalize the variant of NRP based on the variant of NSP considered in this paper (Section 4.2).
3. We propose a new ASP-based solution to the NSP overcoming some limitations of the encoding presented in [22] (Section 5.2).
4. We propose a new ASP-based solution to NRP (Section 6).
5. We present an experimental analysis comparing the basic and advanced ASP solutions as well as with SAT and ILP based solutions (Section 7.1). Results show a significant improvement of the performance of ASP solvers and, specifically, CLINGO performs better than all other alternatives.

6. We present an evaluation of rescheduling, which simulates sudden absences of nurses (Section 7.2). Results show that ASP systems can compute a solution for NRP in few seconds in our setting.

2. Preliminaries

We assume that the reader is familiar with basic knowledges on Answer Set Programming and ASP-Core-2 input language specification [17]. Minimal notions are anyhow introduced in this section in order to ease the understanding of the encodings presented in Sections 5–6.

The evaluation of an ASP program is usually made in two steps, called *grounding* and *solving*. First, the ASP program with variables is evaluated by the grounder, which is responsible to produce its variable-free (propositional) counterpart.

Example 2.1 (Grounding)**.** Consider as example the following rules:

```
     a(1..6).    b(1..10).    c(1..3).
p :  {out(X,Y, Z) : c(Z)} = 1 :- a(X), b(Y).
c :  :- b(Y), c(Z), #count{X : out(X,Y,Z)} <= 1.
w :   :~ out(X,Y,Z). [Z@1,X,Y]
```

The grounding of p consists of 60 (i.e. 6×10) propositional rules of the following form:

```
p1  :   {out(1,1,1); out(1,1,2); out(1,1,3)} = 1.
p2  :   {out(1,2,1); out(1,2,2); out(1,2,3)} = 1.
                          ⋮
p50 :   {out(6,10,1); out(6,10,2); out(6,10,3)} = 1.
```

Intuitively, *choice rule* p_1 enforces that exactly one atom between `out(1,1,1)`, `out(1,2,1)` and `out(1,3,1)` must be true in an answer set. Similar considerations hold for other ground rules generated.

The grounding of c consists of 30 (i.e. 10×3) *(integrity) constraints* of the following form:

```
c1  :   :- #count{out(1,1,1); out(2,1,1);
                  out(3,1,1); out(4,1,1);
                  out(5,1,1); out(6,1,1)} <= 1.
c2  :   :- #count{out(1,2,1); out(2,2,1);
                  out(3,1,1); out(4,2,1);
                  out(5,2,1); out(6,2,1)} <= 1.
                          ⋮
c30 :   :- #count{out(1,10,3); out(2,10,3);
                  out(3,10,3); out(4,10,3);
                  out(5,10,3); out(6,10,3)} <= 1.
```

As example, constraint c_1 enforces that the count of true atoms among `out(1,1,1)`, `out(2,1,1)`, `out(3,1,1)`, `out(4,1,1)`, `out(5,1,1)`, and `out(6,1,1)` must be greater than or equal to 2.

Finally, the grounding of w consists of 180 (i.e. $6 \times 10 \times 3$) *weak constraints* of the following form:

```
w1   :   :~ out(1,1,1). [1@1]
w2   :   :~ out(1,1,2). [2@1]
                  ⋮
w180 :   :~ out(6,10,3). [3@1]
```

Weak constraints are used to express conditions that should be satisfied. Thus, they may be violated, and their semantics involves minimizing the number of violations. For instance, the informal meaning of w_1 is "out(1,1,1) should preferably be false". In addition, each weak constraint is associated to a weight and to a priority level (defined by the syntax [weight@level]). Optimal answer sets are those minimizing the sum of weights of the violated weak constraints at the highest priority level and, among them, those which minimize the sum of weights of the violated weak constraints in the next lower level. The process is applied recursively until the lowest level. ◁

The propositional program produced by the grounder is evaluated by the solver, whose role is to produce an answer set. Modern ASP solvers implement the algorithm CDCL [29], which is based on the pattern *choose-propagate-learn*. Intuitively, the idea is to build an answer set step-by-step by starting from an empty interpretation, i.e. all atoms are initially undefined. Then, the algorithm heuristically *chooses* an undefined atom to be true in the answer set, and the deterministic consequences of this choice are *propagated*, i.e. new atoms are derived true or false in the answer set candidate. The propagation may lead to a *conflict*, i.e. an atom is true and false at the same time. In this case, the conflict is analyzed and a new constraint is added to the propositional program (*learning*). The conflict is then repaired, i.e. choices leading to the conflict are retracted and a new undefined atom is heuristically selected. The algorithm then iterates until no undefined atoms are left, i.e. an answer set is produced, or the incoherence of the propositional program is proved, i.e. no answer sets are admitted.

Example 2.2 (Propagation)**.** Consider the propositional rule p_1 reported in Example 2.1 and assume that atoms `out(1,1,1)` and `out(1,1,2)` have been heuristically assigned to false. Then, the solver derives `out(1,1,3)` to true because it is the only way to satisfy rule p_1. ◁

In presence of weak constraints, modern solvers apply a strategy based on the so-called *unsatisfiable cores*. In particular, such an algorithm starts by searching an answer set satisfying all weak constraints, which would be therefore optimal. On the other hand, if there is no an answer set of this kind, a subset of the weak constraints that cannot be jointly satisfied is identified. Such a set is called unsatisfiable core, and essentially evidences that any optimal answer set must sacrifice at least one of the desiderata expressed by the weak constraints. Moreover, the program can be modified by replacing the weak constraints in the unsatisfiable core with new weak constraints that essentially express a preference for optimum answer sets satisfying all but one of the original weak constraints, and anyhow the largest number of them, so that the process can be reiterated.

3. Problems Description

We start this section by providing a description of the Nurse Scheduling Problem as posed by an Italian hospital (Section 3.1). In the description we identify elements that allow to reuse the proposed solution even if part of the specification given by the hospital will change. After that, we describe the Nurse Rescheduling Problem as posed by the same Italian hospital (Section 3.2).

3.1. Nurse Scheduling Problem

NSP amounts to the totalization of partial schedules assigning nurses to working and rest days over a predetermined period of time, which is fixed to one year in this paper. Usually, partial schedules to be totalized involve few data concerning already authorized vacations. Admissible schedules must satisfy a set of requirements dictated by the rules of the hospital units. In the following, we report the requirements specified by an Italian hospital.

Hospital requirements. For every working day, nurses can be assigned to exactly one of the following shifts: *morning* (7 A.M. – 2 P.M.), *afternoon* (2 P.M. – 9 P.M.), *night* (9 P.M. – 7 A.M.). Thus, the morning and the afternoon shifts last 7 hours, whereas the night shift lasts 10 hours. In order to ensure the best assistance program for patients, the number of nurses in every shift $x \in \{morning, afternoon, night\}$ must range from x^{nurse}_{min} to x^{nurse}_{max}.

Nurses requirements. In order to guarantee a fair workload, each nurse must work a number of hours ranging from $work_{min}$ to $work_{max}$. Additional requirements are also imposed to ensure an adequate rest period to each nurse: *(a)* nurses are legally guaranteed 30 days of paid vacation; *(b)* the starting time of a shift must be at least 24 hours later than the starting time of the previous shift; and *(c)* each nurse has at least two ordinary rest days for every window of fourteen days. In addition, nurses working on two consecutive nights deserve one special rest day in addition to the ordinary rest days.

Balance requirements. The number of morning, afternoon and night shifts assigned to every nurse should range over a set of acceptable values, that is, from x^{day}_{min} to x^{day}_{max} for each $x \in \{morning, afternoon, night\}$.

Optimal balance requirements. In addition to the above requirements, the hospital reported some further requirements to guarantee a balance in the assignment of shifts. Indeed, the number of morning, afternoon and night shifts assigned to every nurse should be *preferably* fixed to some desired values, that is, x^{day} for each $x \in \{morning, afternoon, night\}$.

3.2. Nurse Rescheduling Problem

NRP addresses situations where an already computed schedule is not usable because of sudden absences of some nurses. Stated differently, nurses are following a previously computed schedule, and some of them report impossibility to work on some future days, for example because of health problems or personal issues. In such cases, the previously computed schedule has to be changed starting from a future day that must not follow any reported absence. The new schedule must clearly satisfy all requirements described in Section 3.1, with the exception that any absence due to health problems must not be rescheduled. Additionally, the new schedule must minimize the differences with the previous schedule, and such a minimization has priority over any other optimality requirements.

4. Problems Formalization

The computational problems described in the previous section, namely the Nurse Scheduling Problem and the Nurse Rescheduling Problem, are formalized in Section 4.1 and 4.2, respectively. The formalization takes into account all parameters identified in the previous section, which are properly represented as part of the input.

4.1. Nurse Scheduling Problem

According to the requirements described in Section 3.1, we define the following decisional problem NSP^d: Given a set N of nurses, a set S=$\{morning$, $afternoon$, $night$, $special\text{-}rest$, $rest$, $vacation\}$ of shifts, a set $S^w = \{morning, afternoon, night\}$ of working shifts, a partial schedule

$$s' : N \times [1..365] \nrightarrow S, \tag{1}$$

natural numbers $work_{min}$, $work_{max}$, and x^{nurse}_{min}, x^{nurse}_{max}, x^{day}_{min}, x^{day}_{max} for $x \in S^w$, checks the existence of a schedule

$$s : N \times [1..365] \rightarrow S \tag{2}$$

extending s' and satisfying the following conditions:

$$|\{n \in N : s(n,d) = x\}| \in [x^{nurse}_{min}..x^{nurse}_{max}] \tag{3}$$

for all $x \in S^w$, and all $d \in [1..365]$;

$$\begin{aligned} 7\cdot \mid \{d \in [1..365] : s(n,d) \in \{&morning, \\ afternoon\}\} \mid +10\cdot \mid \{d \in [1..365] :& \\ s(n,d) = night\} \mid \ \in [work_{min}..work_{max}]& \end{aligned} \tag{4}$$

$$\mid \{d \in [1..365] : s(n,d) = vacation\} \mid = 30 \tag{5}$$

$$\begin{aligned} &\mid \{d \in [2..365] : s(n,d) = morning, \\ &s(n,d-1) \in \{afternoon, night\}\} \mid = 0 \\ &\mid \{d \in [2..365] : s(n,d) = afternoon, \\ &s(n,d-1) = night\} \mid = 0 \end{aligned} \tag{6}$$

for all $n \in N$;

$$\mid \{d' \in [d..d+13] : s(n,d') = rest\} \mid \geq 2 \tag{7}$$

for all $n \in N$, and all $d \in [1..352]$;

$$\begin{aligned} s(n,d) = special\text{-}rest \text{ if and only if} \\ s(n,d-1) = night \text{ and } s(n,d-2) = night \end{aligned} \tag{8}$$

for all $n \in N$, and all $d \in [3..365]$;

$$|\{d \in [1..365] : s(n,d) = x\}| \in [x^{day}_{min}..x^{day}_{max}] \tag{9}$$

for all $n \in N$, and $x \in S^w$.

Actually, NSP^d does not take into account the optimal balance requirements. To cover such requirements, we define the following *optimization* problem NSP^o: Given a set N of nurses, a set $S = \{morning$, $afternoon$, $night$, $special\text{-}rest$, $rest$, $vacation\}$ of shifts, a set $S^w = \{morning, afternoon, night\}$ of working shifts, a partial schedule s' of the form (1), natural numbers $work_{min}$, $work_{max}$, and x^{nurse}_{min}, x^{nurse}_{max}, x^{day}, x^{day}_{min}, x^{day}_{max} for $x \in S^w$, compute a schedule s of the form (2) satisfying (3)–(9), and minimizing

$$\sum_{x \in S^w,\ n \in N} abs(x^{day} - \mid days^n_x \mid) \tag{10}$$

where

$$days^n_x := \{d \in [1..365] : s(n,d) = x\}. \tag{11}$$

4.2. Nurse Rescheduling Problem

According to the requirements described in Section 3.2, we define two optimization problems, namely NRP^d and NRP^o, where the second also takes into account the optimal balance requirements, which are instead not part of the first computational problem. Specifically, NRP^d is the following problem: Given a set N of nurses, a set $S = \{morning$, $afternoon$, $night$, $special\text{-}rest$, $rest$, $vacation$, $health\text{-}problem$, $personal\text{-}problem$, $morning^{hp}$, $afternoon^{hp}$, $night^{hp}\}$ of shifts, a set $S^w = \{morning, afternoon, night\}$ of working shifts, natural numbers $work_{min}$, $work_{max}$, and x^{nurse}_{min}, x^{nurse}_{max}, x^{day}_{min}, x^{day}_{max} for $x \in S^w$, a (previously computed) schedule s of the form (2), a partial schedule s' of the form (1), and a day $start\text{-}date \in [1..365]$, compute a schedule s'' satisfying (3), (5)–(9), and the following conditions:

$$\begin{aligned} 7\cdot \mid \{d \in [1..365] : s(n,d) \in \{&morning, \\ afternoon, morning^{hp}, afternoon^{hp}\}\}& \mid \\ + 10\cdot \mid \{d \in [1..365] : s(n,d) \in \{&night, \\ night^{hp}\}\} \mid \ \in [work_{min}..work_{max}]& \end{aligned} \tag{12}$$

for all $n \in N$;

$$\text{if } s'(n,d) \in S \setminus \{\mathit{health\text{-}problem}\} \quad \text{then } s''(n,d) = s'(n,d) \tag{13}$$

$$\text{if } s'(n,d) = \mathit{health\text{-}problem} \quad \text{then } s''(n,d) = s(n,d)^{hp} \tag{14}$$

for all $n \in N$, and for all $d \in [1..365]$; and minimizing

$$|\{(n,d) \in N \times [1..365] : s(n,d) \neq s'(n,d)\}| . \tag{15}$$

Similarly, NRP^o is the following computational problem: Given a set N of nurses, a set $S = \{morning, afternoon, night, special\text{-}rest, rest, vacation, personal\text{-}problem, health\text{-}problem, morning^{hp}, afternoon^{hp}, night^{hp}\}$ of shifts, a set $S^w = \{morning, afternoon, night\}$ of working shifts, natural numbers $work_{min}$, $work_{max}$, and x^{nurse}_{min}, x^{nurse}_{max}, x^{day}, x^{day}_{min}, x^{day}_{max} for $x \in S^w$, a (previously computed) schedule s of the form (2), a partial schedule s' of the form (1), and a day $start\text{-}date \in [1..365]$, compute a schedule s'' satisfying (3), (5)–(9), (12)–(14), and minimizing (15) and (10), in this order. Stated differently, quantity (15) is minimized first, and ties are possibly broken by minimizing (10).

5. Nurse Scheduling via ASP

In this section, after recalling an existing encoding from the literature [22] (Section 5.1), we present the new advanced encoding (Section 5.2).

5.1. Existing Encoding

Instances of NSP^d and NSP^o are represented by means of ASP facts and constants. Specifically, the interval [1..365] of days is encoded by facts of the form `day(`d`)`, for all $d \in [1..365]$, and the number of days is fixed by the fact `days(365)`. The nurses are encoded by facts of the form `nurse(`n`)`, for all $n \in N$. Available shifts are encoded by facts of the form `shift(`id_x, x, h`)`, where $id_x \in [1..6]$ is a numerical identifier of the shift $x \in \{morning, afternoon, night, special\text{-}rest, rest, vacation\}$, and h is the number of working hours associated to the shift. Natural numbers x^{nurse}_{min}, x^{nurse}_{max} for $x \in \{morning, afternoon, night\}$ are represented by facts of the form `nurseLimits(`id_x`,` x^{nurse}_{min}`,` x^{nurse}_{max}`)`, where id_x is the identifier of the shift x. Natural numbers $x^{day}, x^{day}_{min}, x^{day}_{max}$ for $x \in \{morning, afternoon, night\}$ are represented by `dayLimits(`id_x`,` x^{day}`,` x^{day}_{min}`,` x^{day}_{max}`)`, while numbers $work_{min}$ and $work_{max}$ are represented by instances of `workLimits(`$work_{min}$`,` $work_{max}$`)`. Hence, according to the specification given by the hospital, the following facts and constants are considered in our setting:

```
day(1..365). days(365). nurses(1..41).
shift(1,morning,7).
shift(2,afternoon,7).
shift(3,night,10).
shift(4,specialrest,0).
shift(5,rest,0).
shift(6,vacation,0).
nurseLimits(1,6,9).
nurseLimits(2,6,9).
nurseLimits(3,4,7).
dayLimits(1,78,74,82).
dayLimits(2,78,74,82).
dayLimits(3,60,58,61).
workLimits(1687,1692).
```

The computed schedule is encoded by atoms of the form `assign(`n, x, d`)`, representing that nurse n is assigned shift x on day d, that is, $s(n,d) = x$. The same predicate `assign` is used to specify the partial schedule s' in input.

The ASP encoding introduced in [22] is reported in Figure 1. It implements the *Guess&Check* programming methodology: Choice rule r_1 is used to guess the schedule $s : N \times [1..365] \to [1..6]$ extending s' and assigning each day of each nurse to exactly one shift, and rules r_2–r_{13} are used to discard schedules not satisfying some of the desired requirements. Specifically, hospital requirements, formalized as property (3), are enforced by the integrity constraints r_2 and r_3, which filter out assignments exceeding the limits. Regarding nurse requirements, property (4) is enforced by integrity constraints r_4 and r_5, property (5) by integrity constraint r_6, property (6) by integrity constraint r_7, property (7) by integrity constraint r_8, and property (8) by integrity constraint r_9–r_{11}. Note that r_7 takes advantage of the numerical identifiers associated with shifts, and in particular by the fact that morning has id 1, afternoon has id 2, and night has id 3. Concerning balance requirements, formalized as property (9), they are enforced by integrity constraints r_{12} and r_{14}. Rules r_1–r_{13} encode NSP^d, while for NSP^o we also need

```
      % Choose an assignment for each day and for each nurse.
r1 :  {assign(N,X,D) : shift(X,Name,H)} = 1 :- day(D), nurse(N).

      % Limits to nurses that must be present for each shift.
r2 :  :- day(D), #count{N : assign(N,X,D)} > Max, nurseLimits(X,Min,Max).
r3 :  :- day(D), #count{N : assign(N,X,D)} < Min, nurseLimits(X,Min,Max).

      % Each nurse works at least Min and at most Max hours per year.
r4 :  :- nurse(N), #sum{H,D : assign(N,X,D), shift(X,Na,H)} > Max, workLimits(Min,Max).
r5 :  :- nurse(N), #sum{H,D : assign(N,X,D), shift(X,Na,H)} < Min, workLimits(Min,Max).

      % Exactly 30 days of holidays. The ID 6 corresponds to the vacation.
r6 :  :- nurse(N), #count{D : assign(N,6,D)} != 30.

      % Each nurse cannot work twice in 24 hours (based on the order on the IDs).
r7 :  :- nurse(N), assign(N,X1,D), assign(N,X2,D+1), X2 < X1, X1 <= 3.

      % At least 2 rest days each 14 days. The ID 5 is associated to rest.
r8 :  :- nurse(N), day(D), days(DAYS), D <= DAYS-13,
          #count{D1 : assign(N,5,D1), D1 >= D, D1 <= D+13} < 2.

      % After two consecutive nights there is one rest day.
      % The ID 3 is associated to the shift night, while 4 is associated to special rest.
r9 :  :- not assign(N,4,D), assign(N,3,D-2), assign(N,3,D-1).
r10 : :- assign(N,4,D), not assign(N,3,D-2).
r11 : :- assign(N,4,D), not assign(N,3,D-1).

      % Balance requirements.
r12 : :- nurse(N), #count{D : assign(N,X,D)} > Max, dayLimits(X,T,Min,Max).
r13 : :- nurse(N), #count{D : assign(N,X,D)} < Min, dayLimits(X,T,Min,Max).

      % Added only in the optimization variant.
r14 : :~ nurse(N), V=#count{D : assign(N,X,D)}, dayLimits(X,T,Min,Max),
          V >= Min, V <= Max. [|V-T|@1,N]
```

Fig. 1. ASP encoding introduced in [22] for NSP^o (and for NSP^d if r_{14} is removed).

weak constraint r_{14}: It assigns a cost to each admissible schedule measured according to function (10). Optimum schedules are those minimizing such a cost.

5.2. *Advanced Encoding*

The aim of this section is to introduce a new encoding, shown in Figure 2, which improves the encoding reported in the previous section. First of all, it has to be noted that many constraints of the encoding in Figure 1 only involve assignments to working shifts, that is, morning, afternoon and night. The *Guess* part of the encoding (i.e. rule r_1) can thus be replaced by two different choice rules, r'_{1a} and r'_{1b}, where r'_{1a} guesses among one of the working shifts or otherwise marks nurses as *not working*, and r'_{1b} eventually guesses among rest, special-rest and vacation for each nurse marked as not working. To achieve such a behavior, an additional *meta-shift* is added to the set of facts, namely `shift(7,notworking,0)`.

A second improvement is obtained by combining the knowledge represented by (4) and (9) with some observations on how rules r_4 and r_5 are evaluated. In fact, during the solving phase, rules obtained by instantiating r_4 and r_5 comprise aggregates with relatively large aggregation sets and few different weights. Specifically to our setting, where morning and afternoon shifts are fixed to 7 hours, and night shifts to 10 hours, each aggregation set contains 365 elements with weight 7, and 365 elements with weight 10. It turns out that several schedules result into exactly the same sum value. The question is now how many of these schedules actually satisfy both (4) and (9). Restricting to the specification given by the hospital, that is, $morning^{day}_{min} = afternoon^{day}_{min} = 74$, $morning^{day}_{max} = afternoon^{day}_{max} = 82$, $night^{day}_{min} = 58$, and $night^{day}_{max} = 61$, the possible sum values are those reported in Table 1, where we also highlight admissible values in the interval $[work_{min}..work_{max}] = [1687..1692]$. The new encoding therefore determines the admissible pairs of the form $(N, M + A)$, where M, A, N are the number of morning, afternoon and nights assigned to a given nurse, by means of rule r'_{15}. These pairs are then used to check whether the assignment of working shifts is valid for each nurse by means of rules r'_4 and r'_5.

```
       % Choose an assignment for each day and for each nurse.
r'1a : {assign(N,X,D):shift(X,Name,H), X != 4, X != 5, X != 6} = 1 :- day(D), nurse(N).
r'1b : {assign(N,X,D):shift(X,Name,H), X >= 4, X <= 6} = 1 :- day(D), nurse(N), assign(N,7,D).

       % Limits to nurses that must be present for each shift.
r'2  : :- day(D), #count{N : assign(N,X,D)} > Max, nurseLimits(X,Min,Max).
r'3  : :- day(D), #count{N : assign(N,X,D)} < Min, nurseLimits(X,Min,Max).

       % Nurses requirements.
r'4  : valid(Nu) :- nurse(Nu), admissible(N,M+A),
            countGE(1,Nu,M), not countGE(1,Nu,M+1),
            countGE(2,Nu,A), not countGE(2,Nu,A+1),
            countGE(3,Nu,N), not countGE(3,Nu,N+1).
r'5  : :- nurse(N), not valid(N).

       % Exactly 30 days of holidays. The id 6 corresponds to the vacation.
r'6  : :- nurse(N), #count{D : assign(N,6,D)} != 30.

       % Each nurse cannot work twice in 24 hours (based on the order on the IDs).
r'7  : :- nurse(N), assign(N,X1,D), assign(N,X2,D+1), X2 < X1, X1 <= 3.

       % At least 2 rest days each 14 days. The id 5 is associated to rest.
r'8  : :- nurse(N), day(D), days(DAYS), D <= DAYS-13,
            #count{D1 : assign(N,5,D1), D1 >= D, D1 <= D+13} < 2.

       % After two consecutive nights (ID 3) there is one rest day (ID 4).
r'9  : :- not assign(N,4,D), assign(N,3,D-2), assign(N,3,D-1).
r'10 : :- assign(N,4,D), not assign(N,3,D-2).
r'11 : :- assign(N,4,D), not assign(N,3,D-1).

       % A nurse should be assigned to the shift X at least Min and at most Max days.
r'12 : :- dayLimits(X,T,Min,Max), nurse(N), not countGE(X,N,Min).
r'13 : :- dayLimits(X,T,Min,Max), nurse(N), countGE(X,N,Max+1).

       % Added only in the optimization variant. The ID 7 corresponds to notworking.
r'14 : :~ nurse(N), countGE(X,N,V), not countGE(X,N,V+1),
            dayLimits(X,T,Min,Max), X != 7, D= |V-T|. [D@1,N,X]

       % Admissible pairs of nights and morning+afternoon shifts, and limits for nonworking days.
r'15 : admissible(N,M+A) :-
            dayLimits(1,T1,MinM,MaxM), dayLimits(2,T2,MinA,MaxA), dayLimits(3,T3,MinN,MaxN),
            M = MinM..MaxM, A = MinA..MaxA, N = MinN..MaxN,
            V = 7*(M+A) + 10*N, workLimits(MinW, MaxW), MinW <= V, V <= MaxW.

r'16 : dayLimits(7,null,Min,Max) :- days(DAYS), Min = #min{DAYS-MA-N : admissible(N,MA)},
            Max = #max{DAYS-MA-N : admissible(N,MA)}.

       % countGE(S,N,V) is derived when a nurse N is assigned to the shift X at least V days.
r'17 : countGE(X,N,V) :- nurse(N), dayLimits(X,T,Min,Max), V = Min..Max+1,
            #count{D : assign(N,X,D)} >= V.

       % The derivation of countGE(X,N,V) implies the derivation of countGE(X,N,V-1).
r'18 : :- dayLimits(X,T,Min,Max), V > Min, countGE(X,N,V), not countGE(X,N,V-1).
```

Fig. 2. Advanced ASP encoding for NSP^o (and for NSP^d if r'_{14} is removed).

N	$M+A$																
	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164
58	1616	1623	1630	1637	1644	1651	1658	1665	1672	1679	1686	1693	1700	1707	1714	1721	1728
59	1626	1633	1640	1647	1654	1661	1668	1675	1682	**1689**	1696	1703	1710	1717	1724	1731	1738
60	1636	1643	1650	1657	1664	1671	1678	1685	**1692**	1699	1706	1713	1720	1727	1734	1741	1748
61	1646	1653	1660	1667	1674	1681	**1688**	1695	1702	1709	1716	1723	1730	1737	1744	1751	1758

Table 1

Number of working hours assigned to nurse n, that is, $7 \cdot (M + A) + 10 \cdot N$, where $M = \mid \{d \in [1..365] : s(n, d) = morning\} \mid$, $A = \mid \{d \in [1..365] : s(n, d) = afternoon\} \mid$, and $N = \mid \{d \in [1..365] : s(n, d) = night\} \mid$. Admissible values, that is, those in the interval [1687..1692], are emphasized in bold

Actually, rules r'_4 and r'_5 take advantage from a third improvement of the advanced encoding. The number of morning, afternoon and night shifts that can be assigned to a nurse must adhere to (9), and are therefore limited to a few different values. The possible values of these aggregations are therefore encoded by means of atoms of the form $\texttt{countGE}(x, n, v)$, being true whenever $| \ days^n_x \ | \geq v$, where $days^n_x$ is (11). It turns out that any answer set satisfies the following property: For each shift x and for each nurse n, there is exactly one value v such that $\texttt{countGE}(x, n, v), \texttt{not countGE}(x, n, v+1)$ is true. In the advanced encoding, predicate `countGE` is defined by rule r'_{17}. Moreover, rule r'_{18} is used to enforce truth of $\texttt{countGE}(x, n, v-1)$ whenever $\texttt{countGE}(x, n, v)$ is true; it is not required for correctness, but convenient to prune the search space in case $\texttt{countGE}(x, n, v)$ is assigned to true during the computation even if $| \ days^n_x \ | \geq v$ does not yet hold (for example, in case $\texttt{countGE}(x, n, v)$ is selected as a branching literal). In addition, since rules r_{12}–r_{14} aggregate on sets $days^n_x$ for $x \in \{morning, \ afternoon, \ night\}$, it is convenient to rewrite these rules in terms of predicate `countGE`, hence obtaining rules r'_{12}–r'_{14}.

In the previous optimization, note that conjunction $\texttt{countGE}(x, \ n, \ v)$, $\texttt{not countGE}(x, \ n, \ v+1)$ in rule r'_4 represents the aggregate `#count{D : assign(n,x,D)} = v`. A predicate `countEQ` has not been used because predicate `countGE` is also used in rules r'_{12} and r'_{13}, where the encoded aggregates are respectively `#count{D : assign(n,x,D)}` $\leq v_{min}$, and `#count{D : assign(n,x,D)}` $\geq v_{max}$. Moreover, the knowledge encoded by predicate `countGE` allows for the introduction of rule r'_{18}, which as already explained further prunes the search space; such a rule is similar to the *order encoding* used by SMT solvers to handle integer constants [**?**].

Finally, a further improvement is obtained by checking the number of nonworking days assigned to each nurse. For the specification given by the hospital it must range between 149 and 150, and in general the admitted range can be determined by rule r'_{16}. The check itself is then performed by rules r'_{12} and r'_{13} (for S being 7). Note that also this last check is not required to guarantee correctness of the encoding, but instead to further prune the search space.

6. Nurse Rescheduling via ASP

The aim of this section is to introduce an encoding, shown on Figure 3, for updating a given schedule due to a sudden absence of one or more nurses. In particular, the absence of nurses is encoded by atoms of the form $\texttt{unavailable}(n, d_1, d_2, t)$, representing that nurse n is absent from the day d_1 to day d_2 ($d_1 \leq d_2$) due to reason t, where t can be $health\_problem$ or $personal\_problem$. A solution of NSP^d (NSP^o), i.e. the schedule to update, is represented by atoms of the form $\texttt{previous\_assign}(n, x, d)$, representing that nurse n was assigned shift x on day d.

The idea is to compute a new schedule, taking into account the unavailability of some nurses, such that differences with the previous schedule are minimized. To this end, rules s_2 and s_3 are used to compute the days for which the schedule of an unavailable nurse must be changed, corresponding to all working days between the range of unavailability provided by the nurse. Schedules for the days before starting date are not changed (rule s_4). Then, nurses that are absent due to health problems cannot be assigned to other shifts (rules s_5–s_7), whereas nurses that are absent for personal problems are assigned to the shift rest (rules s_8 and s_9). The differences between the new schedule and the previous one are minimized by means of the weak constraint s_{10}. The correctness of the new schedule is then checked using a slight different variant of the encoding reported in Figure 2, where the only differences are in rule r'_{1a}, r'_{1b}, and r'_{17}, which are substituted with r''_{1a}, r''_{1b}, and r''_{17}. The new version takes also into account the days in which a nurse is unavailable due to health problems.

7. Empirical Evaluation

In this section the results of the empirical evaluation is reported. The experiments take into account both the problem of generating a schedule, and the problem of repair a schedule to sudden absences. Both experiments were executed on the same computer equipped with four core Intel Xeon CPU X3430 2.4 GHz and 16 GB of RAM, running Debian Linux. Time and memory were limited to 1 hour and 8 GB, respectively. All ASP material can be found at http://www.star.dist.unige.it/~marco/Data/material.zip.

7.1. Nurse Scheduling

The experiments consider real data provided by the Italian hospital unit, which comprises a set of 41 nurses and holidays selected using the preferences of nurses of the year 2015. Moreover, the scalability of

```
       % A nurse is unavailable in a day if he/she was working.
s1 :   working(N,X,D) :- previous_assign(N,X,D), X=1..3.
s2 :   health_problem(N,X,D) :- unavailable(N,D1,D2,health_problem), D >= D1, D <= D2, working(N, X, D).
s3 :   personal_problem(N,D) :- unavailable(N,D1,D2,personal_problem), D >= D1, D <= D2, working(N,X,D).


       % Assignments of previous days are not modified.
s4 :   assign(N,X,D) :- previous_assign(N,X,D), D < D1, start_date(D1).

       % Assign nurses with health problems.
s5 :   assign(N,health_problem(X),D) :- health_problem(N,X,D).
s6 :   assign(N,health_problem(X),D) :- previous_assign(N,health_problem(X),D).

       % Nurses with health problems cannot be assigned to other shifts.
s7 :   unavailable(N,D) :- assign(N,health_problem(X),D), X=1..3.

       % Nurses with personal problems are assigned to shift rest.
s8 :   assign(N,5,D) :- personal_problem(N,D).
s9 :   assign(N,7,D) :- personal_problem(N,D).

       % Minimize the differences between the new schedule and the previous one.
s10 :   :~ assign(N,X,D), not previous_assign(N,X,D). [1@2, N,X,D]

        % Guess an assignment for all the days after the first day.
r''1a : {assign(N,X,D) : shift(X,Name,H), X != 4, X != 5, X != 6} = 1 :- day(D), nurse(N),
             start_date(D1), D >= D1, not unavailable(N,D).
r''1b : {assign(N,X,D) : shift(X,Name,H), X >= 4, X <= 6} = 1 :- day(D), nurse(N),
             start_date(D1), D >= D1, assign(N,7,D).

        % countGE(X,N,V) is derived when a nurse N is assigned to the shift X at least V days.
r''17 : countGE(X,N,V) :- nurse(N), dayLimits(X,T,Min,Max), V = Min..Max+1,
             #count{D : assign(N,X,D); D1: assign(N,health_problem(X),D1)} >= V.

        % Rules from r'2 to r'16, and r'18 are included
```

Fig. 3. ASP encoding for rescheduling NRP^o (and for NRP^d if r'_{14} is removed).

the approach has been evaluated by considering different number of nurses. In particular, an additional experiment was run by considering 10, 20, 41, 82 and 164 nurses without fixed holidays. Both the decisional (NSP^d) and the optimization (NSP^o) variants of NSP were considered. Concerning the decisional variant, the ASP-based approaches are compared to solutions based on SAT and on ILP.

The ASP encodings have been tested using the system CLINGO (version 5.1.0) [30] and the system WASP (version 996bfb3) [5] combined with the grounder GRINGO [27], both configured with the core-based algorithms [3] for NSP^o. Solvers LINGELING (version bbc-9230380-160707) [14], GLUCOSE (version 4.1) [8] and CLASP (v. 3.2.2) have been executed on the SAT encoding, while the commercial tool GUROBI (version 7.0.2) [32] on the ILP encoding. Concerning the optimization variant, the same tools for ASP and ILP have been used, whereas LINGELING and GLUCOSE have been replaced by the MaxSAT tools MSCG [40] and MAXINO [6], both binaries taken from MaxSAT Competition 2016.

In order to test SAT and ILP solutions, a pseudo-Boolean formula based on the ideas of the advanced ASP encoding is created. The pseudo-Boolean formula was represented using the OPB format, which is parsed by the tool GUROBI. Concerning the SAT-based solutions, the state-of-the-art tool PBLIB [45] has been used to convert the pseudoBoolean formula into a CNF. The running time of PBLIB has not been included in the analysis.

Results. The results of the instance with parameters provided by the Italian hospital are reported in Table 2. The best result overall is obtained by CLINGO exe-

Table 2

Results of the experiment with 41 nurses and fixed holidays

NSP^d		NSP^o	
Solver	**Time (s)**	**Solver**	**Time (s)**
CLINGO (ORIG ENC)	1352	CLINGO (ORIG ENC)	431
CLINGO (ADV ENC)	43	CLINGO (ADV ENC)	70
WASP (ORIG ENC)	-	WASP (ORIG ENC)	-
WASP (ADV ENC)	-	WASP (ADV ENC)	-
GLUCOSE (SAT ENC)	-	MSCG (MAXSAT ENC)	-
LINGELING (SAT ENC)	-	MAXINO (MAXSAT ENC)	-
CLASP (SAT ENC)	-	CLASP (MAXSAT ENC)	-
GUROBI (ILP ENC)	1018	GUROBI (ILP ENC)	1073

cuted on the advanced encoding for both NSP^d and NSP^o, which is able to find a schedule in 42 and 70 seconds, respectively. This is a clear improvement with respect to the original encoding. Indeed, CLINGO executed on the original encoding was able to find a schedule in 22 and 7 minutes for the decisional and optimization variant, respectively. However, the advanced encoding does not help the ASP solver WASP: Its bad performance seems related to the branching heuristic, which is not effective on this particular domain. Indeed, WASP often selects atoms which do not appear in the aggregates, and this strategy seems to be ineffective for solving such instances. SAT-based (and MaxSAT-based) approaches are also not able to find a schedule within the allotted time and memory. In this case their performance can be explained by looking at the large size of the formula to evaluate (approximately 65 millions of clauses), which makes the solvers exceed the allotted memory. The tool GUROBI obtained good performance on both NSP^d and NSP^o instances. In particular for NSP^d GUROBI is faster than CLINGO executed on the original encoding. On the contrary, GUROBI is slower than CLINGO on the NSP^o instance.

Scalability. A further analysis about the scalability of the encoding, considering different numbers of nurses, is reported in the following. In particular, for both NSP^d and NSP^o, five instances are considered, containing 10, 20, 41, 82 and 164 nurses, respectively. For each instance, the number of working nurses during each shift is proportionally scaled and holidays are randomly generated, whereas other requirements are not modified. Results are reported in Table 3.

Table 3
Scalability of the approach. Solving time (s) for each solver

	Solver	**Nurses**				
		10	**20**	**41**	**82**	**164**
NSP^d	CLINGO (ORIG ENC)	155	117	738	1486	2987
	CLINGO (ADV ENC)	4	9	70	351	1291
	WASP (ORIG ENC)	-	-	-	-	-
	WASP (ADV ENC)	5	20	-	-	-
	GLUCOSE (SAT ENC)	-	-	-	-	-
	LINGELING (SAT ENC)	-	-	-	-	-
	CLASP (SAT ENC)	-	-	-	-	-
	GUROBI (ILP ENC)	62	172	1018	-	-
NSP^o	CLINGO (ORIG ENC)	37	94	339	798	1689
	CLINGO (ADV ENC)	4	13	72	482	1590
	WASP (ORIG ENC)	-	-	-	-	-
	WASP (ADV ENC)	4	-	-	-	-
	MSCG (MAXSAT ENC)	-	-	-	-	-
	MAXINO (MAXSAT ENC)	-	-	-	-	-
	CLASP (MAXSAT ENC)	-	-	-	-	-
	GUROBI (ILP ENC)	113	411	2004	-	-

The best results overall is obtained again by CLINGO executed on the advanced encoding, which outperforms all other tested approaches. Concerning ASP-based approaches, their performance is much better when they are executed on the advanced encoding. Indeed, the running time of CLINGO considerably decreases for all tested instances in both NSP^d and NSP^o. As a possible explanation for this behavior, Figure 4 shows a comparison among the number of conflicts found by CLINGO executed on the original and on the advanced encodings. The new encoding takes advantage of the better propagations, thus it is able to find a solution with a smaller number of conflicts. Concerning NSP^o, it can also be observed that the performance of the two versions of CLINGO are comparable on the instance with 164 nurses, even if the number of conflicts are much lower when CLINGO is executed on the advanced encoding. To explain this discrepancy we analyzed the number of branching choices performed by CLINGO, which are around 128 millions for the original encoding, and around 300 millions for the advanced encoding. Thus, for this specific instance, the branching heuristic of CLINGO seems to be more effective when the original encoding is considered. Moreover, WASP executed on the advanced encoding is able to find a schedule for NSP^d when 10 and 20 nurses are considered, whereas WASP executed on the original one does not terminate the computation in 1 hour. The performance of SAT (and MaxSAT) solvers are also in this case not satisfactory since they cannot solve any of the tested instances. In this case, when 10 nurses are considered the size of the formula is not prohibitive (approximately 15 millions of clauses), however all the SAT solvers were not able to find a solution within the allotted time. GUROBI can solve instances up to 41 nurses, whereas it is not able to find a schedule when 82 and 164 nurses are considered.

7.2. Nurse Rescheduling

The experiments for rescheduling consider the schedule computed by CLINGO when 41 nurses. Then, absences of nurses were randomly generated, considering three different sets of instances, i.e. short, mid and long term absences. For each set, 50 instances were generated.

Absences ranging from 1 to 7 days are considered short; from 8 to 30 days are considered mid and from 30 days to one year are considered long. For each test, a random number of absent nurses is generated and for

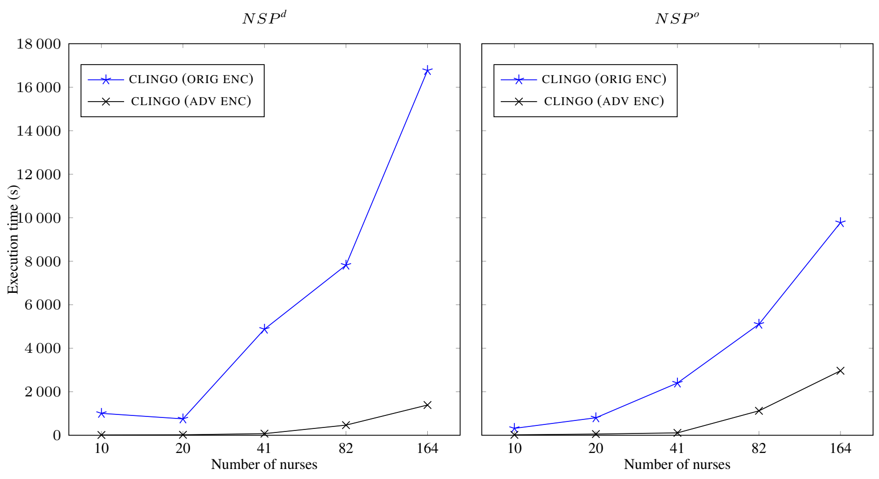

Fig. 4. Comparison of the number of conflicts (in thousands) of CLINGO executed on the original and on the advanced encodings for both NSP^d and NSP^o with different number of nurses.

each nurse a range of unavailability is also randomly generated. In short term instances, the number of absent nurses is limited to 3, while for mid term and long term instances it is limited to 2 and 1, respectively. Moreover, short term instances include both health and personal problems, while mid term and long term include only health problems. An example of a short term instance is the following:

```
unavailable(12,303,305,personal_problem).
unavailable(9,304,306,health_problem).
start_date(303).
```

representing that nurses with ids 12 and 9 are not available for 3 days due to personal problems and health problems, respectively.

Since ASP has been shown to be the best solution for NSP, here we consider only this approach. The encodings have been tested using the system CLINGO (version 5.1.0) [30] and the solver WASP [5] combined with the grounder GRINGO [27], both configured with the core-based algorithms [3].

Results. The results of the experiments are reported in Table 4. First of all, both CLINGO and WASP are able to solve all the instances within the time limit. Concerning CLINGO, its average running time is approximately 7 and 26 seconds for NRP^d and NRP^o, respectively. Concerning WASP, its running time is also satisfactory since it is on average 48 and 65 seconds for NRP^d and NRP^o, respectively. The running time does not depend on the length of the absence, instead we observed that both solvers needed more time to find a solution on instances including absences due to personal problems. As example, short term instance number 14 comprises only absences related to personal problems and it is solved in approximately 4 minutes by CLASP and in approximately 6 minutes by WASP. Moreover, we report that CLINGO is able to find a solution within *10 seconds* for the 97% of the considered NRP^d and NRP^o instances.

8. Related Work

In recent years, several approaches to solve NSP have been proposed. The main differences concern (i) the planning periods; (ii) the different type of shifts; (iii) the requirements related to the coverage of shifts, i.e. the number of personnel needed for every shift; and (iv) other restrictions on the rules of nurses (see [16] for more detailed information). In this paper a one-year window of time has been considered as in [19], where however the same requirements on nurses and hospitals were not reported. Concerning the shifts, we considered three different shifts (morning, afternoon and night) with no overlapping among shifts, whereas in the literature other approaches are based on one single shift only (see e.g. [39]). Other requirements depend

Table 4
Average solving time (s) for CLASP and WASP

Absences	NRP^d		NRP^o	
	CLINGO	**WASP**	**CLINGO**	**WASP**
Short term	14	61	69	87
Mid term	3	10	3	10
Long term	5	73	5	100

on the different policies of the considered hospitals. Thus, this makes the different strategies not directly comparable with each other.

Concerning other solving technologies reported in the literature, they range from mathematical to meta-heuristics approaches, including solutions based on integer programming [9,11], genetic algorithms [2], fuzzy approaches [47], and ant colony optimization algorithms [33], to mention a few. Detailed and comprehensive surveys on NSP can be found in [16,20].

As far as the relation between the basic and advanced encodings, the two encodings mainly differ with respect to how the constraints related to hospital and balance requirements are modeled. Indeed, the new encoding takes into account only combinations of parameters values that can lead to a valid schedule.

As argued in two recent surveys [21,43], NRP is considered strategic in many hospitals, since rescheduling is an almost daily activity and any change to previous schedule usually face unexpected resistance. Thus, most of the approaches are based on minimizing the changes with the previous schedule as also considered in this paper.

Many solutions have been proposed in the literature [21,43], including heuristic approaches [34,36, 44], genetic algorithms [42], parallel algorithms [13], artificial immune systems [37], and integer multicommodity flow formulations [41].

In [12], authors considered the possibility of hiring temporary staff, called *travelers*, for dealing with sudden absences of nurses. It turns out that this approach is needed in hospitals facing a chronic nursing shortage, i.e. when permanent nurses cannot fulfill the requirements imposed by the hospital, which is not the case of the hospital considered here.

Finally, ASP encodings have been proposed for scheduling problems other than NSP and NRP: *Incremental Scheduling Problem* [18], where the goal is to assign jobs to devices such that their executions do not overlap one another; and *Team Building Problem* [46], where the goal is to allocate the available personnel of a seaport for serving the incoming ships. However, to the best of our knowledge, the only ASP encodings for NSP and NRP are those shown in Sections 5.1, 5.2 and 6.

9. Conclusion

In this paper an advanced ASP encoding for addressing a variant of NSP has been proposed. The new encoding overcomes the limitations of the one proposed in [22] by taking into account intrinsic properties of NSP and internal details of ASP solvers. The new NSP encoding has been used as basis for an ASP-based solution for solving NRP. The ASP-based approach to NSP has been compared with the basic one [22] and with other declarative approaches on real setting provided by an Italian hospital. Results clearly show that CLINGO executed on the new encoding outperforms all alternatives, being able to solve all instances within 30 minutes, even with more than 100 nurses. Concerning NRP solution, it has been tested by simulating sudden absences for nurses along the year. Results on 150 random scenarios are quite good: CLINGO and WASP are able to compute a solution for NRP within few seconds.

Acknowledgments.

We would like to thank Nextage srl for providing support for this work. Mario Alviano has been partially supported by the Italian Ministry for Economic Development (MISE) under project "PIUCultura – Paradigmi Innovativi per l'Utilizzo della Cultura" (n. F/020016/01-02/X27), and under project "Smarter Solutions in the Big Data World (S2BDW)" (n. F/050389/01-03/X32) funded within the call "HORIZON2020" PON I&C 2014-2020, and by Gruppo Nazionale per il Calcolo Scientifico (GNCS-INdAM).

References

[1] Michael Abseher, Martin Gebser, Nysret Musliu, Torsten Schaub, and Stefan Woltran. Shift design with answer set programming. *Fundam. Inform.*, 147(1):1–25, 2016. doi: 10.3233/FI-2016-1396. URL https://doi.org/10.3233/FI-2016-1396.

[2] Uwe Aickelin and Kathryn A. Dowsland. An indirect genetic algorithm for a nurse-scheduling problem. *Computers & OR*, 31(5):761–778, 2004. doi: 10.1016/S0305-0548(03)00034-0. URL https://doi.org/10.1016/S0305-0548(03)00034-0.

[3] Mario Alviano and Carmine Dodaro. Anytime answer set optimization via unsatisfiable core shrinking. *TPLP*, 16(5-6): 533–551, 2016. doi: 10.1017/S147106841600020X. URL https://doi.org/10.1017/S147106841600020X.

[4] Mario Alviano and Wolfgang Faber. The complexity boundary of answer set programming with generalized atoms under the FLP semantics. In *LPNMR*, volume 8148 of *LNCS*, pages 67–72. Springer, 2013. doi: 10.1007/978-3-642-40564-8_7. URL http://dx.doi.org/10.1007/978-3-642-40564-8_7.

[5] Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca. Advances in WASP. In *LPNMR*, volume 9345 of *LNCS*, pages 40–54. Springer, 2015. doi: 10.1007/978-3-319-23264-5_5. URL https://doi.org/10.1007/978-3-319-23264-5_5.

[6] Mario Alviano, Carmine Dodaro, and Francesco Ricca. A MaxSAT Algorithm Using Cardinality Constraints of Bounded Size. In *IJCAI 2015*, pages 2677–2683. AAAI Press, 2015.

[7] Mario Alviano, Carmine Dodaro, and Marco Maratea. An advanced answer set programming encoding for nurse scheduling. In *AI*IA*, volume to appear, 2017.

[8] Gilles Audemard and Laurent Simon. Extreme cases in SAT problems. In Nadia Creignou and Daniel Le Berre, editors, *SAT*, volume 9710 of *LNCS*, pages 87–103. Springer, 2016. doi: 10.1007/978-3-319-40970-2_7. URL https://doi.org/10.1007/978-3-319-40970-2_7.

[9] M. Naceur Azaiez and S. S. Al Sharif. A 0-1 goal programming model for nurse scheduling. *Computers & OR*, 32:491–507, 2005. doi: 10.1016/S0305-0548(03)00249-1. URL https://doi.org/10.1016/S0305-0548(03)00249-1.

[10] Marcello Balduccini, Michael Gelfond, Richard Watson, and Monica Nogueira. The USA-advisor: A case study in answer set planning. In *LPNMR*, volume 2173 of *LNCS*, pages 439–442. Springer, 2001. doi: 10.1007/3-540-45402-0_39. URL https://doi.org/10.1007/3-540-45402-0_39.

[11] Jonathan F. Bard and Hadi W. Purnomo. Preference scheduling for nurses using column generation. *European Journal of Operational Research*, 164(2):510–534, 2005. doi: 10.1016/j.ejor.2003.06.046. URL https://doi.org/10.1016/j.ejor.2003.06.046.

[12] Jonathan F. Bard and Hadi W. Purnomo. Incremental changes in the workforce to accommodate changes in demand. *Health Care Management Science*, 9(1):71–85, 2006. doi: 10.1007/s10729-006-6281-y. URL https://doi.org/10.1007/s10729-006-6281-y.

[13] Zdenek Bäumelt, Jan Dvorák, Premysl Sucha, and Zdenek Hanzálek. A novel approach for nurse rerostering based on a parallel algorithm. *European Journal of Operational Research*, 251(2):624–639, 2016. doi: 10.1016/j.ejor.2015.11.022. URL https://doi.org/10.1016/j.ejor.2015.11.022.

[14] Armin Biere and Andreas Fröhlich. Evaluating CDCL variable scoring schemes. In *SAT*, volume 9340 of *LNCS*, pages 405–422. Springer, 2015. doi: 10.1007/978-3-319-24318-4_29. URL https://doi.org/10.1007/978-3-319-24318-4_29.

[15] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12): 92–103, 2011. doi: 10.1145/2043174.2043195. URL http://doi.acm.org/10.1145/2043174.2043195.

[16] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *J. Scheduling*, 7(6):441–499, 2004. doi: 10.1023/B:JOSH.0000046076.75950.0b. URL https://doi.org/10.1023/B:JOSH.0000046076.75950.0b.

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

[18] Francesco Calimeri, Martin Gebser, Marco Maratea, and Francesco Ricca. Design and results of the Fifth Answer Set Programming Competition. *Artif. Intell.*, 231:151–181, 2016. doi: 10.1016/j.artint.2015.09.008. URL https://doi.org/10.1016/j.artint.2015.09.008.

[19] Peter Chan and Georges Weil. Cyclical staff scheduling using constraint logic programming. In *PATAT*, volume 2079 of *LNCS*, pages 159–175. Springer, 2000. doi: 10.1007/3-540-44629-X_10. URL https://doi.org/10.1007/3-540-44629-X_10.

[20] Brenda Cheang, Haibing Li, Andrew Lim, and Brian Rodrigues. Nurse rostering problems - a bibliographic survey. *European Journal of Operational Research*, 151(3):447–460, 2003. doi: 10.1016/S0377-2217(03)00021-3. URL https://doi.org/10.1016/S0377-2217(03)00021-3.

[21] Alistair Clark, Pam Moule, Annie Topping, and Martin Serpell. Rescheduling nursing shifts: scoping the challenge and examining the potential of mathematical model based tools. *Journal of Nursing Management*, 23(4):411–420, 2015. doi: 10.1111/jonm.12158. URL http://dx.doi.org/10.1111/jonm.12158.

[22] Carmine Dodaro and Marco Maratea. Nurse scheduling via answer set programming. In *LPNMR*, volume 10377 of *LNCS*, pages 301–307. Springer, 2017. doi: 10.1007/978-3-319-61660-5_27. URL https://doi.org/10.1007/978-3-319-61660-5_27.

[23] Carmine Dodaro, Nicola Leone, Barbara Nardi, and Francesco Ricca. Allotment problem in travel industry: A solution based on ASP. In *RR*, volume 9209 of *LNCS*, pages 77–92. Springer, 2015. doi: 10.1007/978-3-319-22002-4_7. URL https://doi.org/10.1007/978-3-319-22002-4_7.

[24] Carmine Dodaro, Philip Gasteiger, Nicola Leone, Benjamin Musitsch, Francesco Ricca, and Konstantin Schekotihin. Combining answer set programming and domain heuristics for solving hard industrial problems (application paper). *TPLP*, 16(5-6):653–669, 2016. doi: 10.1017/S1471068416000284. URL https://doi.org/10.1017/S1471068416000284.

[25] Esra Erdem and Umut Öztok. Generating explanations for biomedical queries. *TPLP*, 15(1):35–78, 2015. doi: 10.1017/S1471068413000598. URL https://doi.org/10.1017/S1471068413000598.

[26] Marco Gavanelli, Maddalena Nonato, and Andrea Peano. An ASP approach for the valves positioning optimization in a water distribution system. *J. Log. Comput.*, 25(6):1351–1369, 2015. doi: 10.1093/logcom/ext065. URL https://doi.org/10.1093/logcom/ext065.

[27] Martin Gebser, Roland Kaminski, Arne König, and Torsten Schaub. Advances in *gringo* series 3. In *LPNMR*, volume 6645 of *LNCS*, pages 345–351. Springer, 2011. doi: 10.1007/978-3-642-20895-9_39. URL https://doi.org/10.1007/978-3-642-20895-9_39.

[28] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider. Potassco: The potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011. doi: 10.3233/AIC-2011-0491. URL https://doi.org/10.3233/AIC-2011-0491.

[29] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187:52–89, 2012. doi: 10.1016/j.artint.2012.04.001. URL https://doi.org/10.1016/j.artint.2012.04.001.

[30] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory solving made easy with clingo 5. In *ICLP TCs*, volume 52 of *OASICS*, pages 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi: 10.4230/OASIcs.ICLP.2016.2. URL https://doi.org/10.4230/OASIcs.ICLP.2016.2.

[31] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[32] Gurobi. The website of gurobi, 2017. URL http://www.gurobi.com.

[33] Walter J. Gutjahr and Marion S. Rauner. An ACO algorithm for a dynamic regional nurse-scheduling problem in austria. *Computers & OR*, 34(3):642–666, 2007. doi: 10.1016/j.cor.2005.03.018. URL https://doi.org/10.1016/j.cor.2005.03.018.

[34] Manabu Kitada and Kazuko Morizawa. A Heuristic Method for Nurse Rerostering Problem with a Sudden Absence for Several Consecutive Days. *International Journal of Emerging Technology and Advanced Engineering*, 3(11):353–361, 2013.

[35] Laura Koponen, Emilia Oikarinen, Tomi Janhunen, and Laura Säilä. Optimizing phylogenetic supertrees using answer set programming. *TPLP*, 15(4-5):604–619, 2015. doi: 10.1017/S1471068415000265. URL https://doi.org/10.1017/S1471068415000265.

[36] Broos Maenhout and Mario Vanhoucke. An evolutionary approach for the nurse rerostering problem. *Computers & OR*, 38 (10):1400–1411, 2011. doi: 10.1016/j.cor.2010.12.012. URL https://doi.org/10.1016/j.cor.2010.12.012.

[37] Broos Maenhout and Mario Vanhoucke. An artificial immune system based approach for solving the nurse re-rostering problem. In *EvoCOP 2013*, volume 7832 of *LNCS*, pages 97–108. Springer, 2013. doi: 10.1007/978-3-642-37198-1_9. URL https://doi.org/10.1007/978-3-642-37198-1_9.

[38] Mónica Caniupán Marileo and Leopoldo E. Bertossi. The consistency extractor system: Answer set programs for consistent query answering in databases. *Data Knowl. Eng.*, 69 (6):545–572, 2010. doi: 10.1016/j.datak.2010.01.005. URL https://doi.org/10.1016/j.datak.2010.01.005.

[39] Holmes E. Miller, William P. Pierskalla, and Gustave J. Rath. Nurse scheduling using mathematical programming. *Operations Research*, 24(5):857–870, 1976. doi: 10.1287/opre.24.5.857. URL https://doi.org/10.1287/opre.24.5.857.

[40] António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-Guided MaxSAT with Soft Cardinality Constraints. In *CP*, volume 8656 of *LNCS*, pages 564–573. Springer, 2014. doi: 10.1007/978-3-319-10428-7_41. URL https://doi.org/10.1007/978-3-319-10428-7_41.

[41] Margarida Moz and Margarida Vaz Pato. Solving the problem of rerostering nurse schedules with hard constraints: New multicommodity flow models. *Annals OR*, 128(1-4):179–197, 2004. doi: 10.1023/B:ANOR.0000019104.39239.ed. URL https://doi.org/10.1023/B:ANOR.0000019104.39239.ed.

[42] Margarida Moz and Margarida Vaz Pato. A genetic algorithm approach to a nurse rerostering problem. *Computers & OR*, 34(3):667–691, 2007. doi: 10.1016/j.cor.2005.03.019. URL https://doi.org/10.1016/j.cor.2005.03.019.

[43] Michael Mutingi and Charles Mbohwa. The nurse rerostering problem: An explorative study. In *International Conference on Industrial Engineering and Operations Management (IEOM)*, 2017. URL http://ieomsociety.org/ieom2017/papers/563.pdf.

[44] Margarida Vaz Pato and Margarida Moz. Solving a bi-objective nurse rerostering problem by using a utopic pareto genetic heuristic. *J. Heuristics*, 14(4):359–374, 2008. doi: 10.1007/s10732-007-9040-4. URL https://doi.org/10.1007/s10732-007-9040-4.

[45] Tobias Philipp and Peter Steinke. Pblib - A library for encoding pseudo-boolean constraints into CNF. In *SAT*, volume 9340 of *LNCS*, pages 9–16. Springer, 2015. doi: 10.1007/978-3-319-24318-4_2. URL https://doi.org/10.1007/978-3-319-24318-4_2.

[46] Francesco Ricca, Giovanni Grasso, Mario Alviano, Marco Manna, Vincenzino Lio, Salvatore Iiritano, and Nicola Leone. Team-building with answer set programming in the gioia-tauro seaport. *TPLP*, 12(3):361–381, 2012. doi: 10.1017/S147106841100007X. URL https://doi.org/10.1017/S147106841100007X.

[47] Seyda Topaloglu and Hasan Selim. Nurse scheduling using fuzzy modeling approach. *Fuzzy Sets and Systems*, 161(11): 1543–1563, 2010. doi: 10.1016/j.fss.2009.10.003. URL http://dx.doi.org/10.1016/j.fss.2009.10.003.