

# Explaining ASP-based Operating Room Schedules

Riccardo Bertolucci<sup>1</sup>, Carmine Dodaro<sup>2</sup>, Giuseppe Galatà<sup>3</sup>, Marco Maratea<sup>1</sup>,  
Ivan Porro<sup>3</sup> and Francesco Ricca<sup>1</sup>

<sup>1</sup>University of Genoa, Genova, Italy

<sup>2</sup>University of Calabria, Arcavacata, Rende CS, Italy

<sup>3</sup>SurgiQ srl, Genova, Italy

## Abstract

The Operating Room Scheduling (ORS) problem refers to the task of assigning patients to operating rooms. An automated solution able to solve the ORS problem in real world scenarios should also be “explainable” to be fully acceptable. Answer Set Programming (ASP) has been successfully applied to solve the ORS problem. However, when the available resources are not enough to satisfy user’s requirements (e.g., insufficient number of free beds) the system cannot compute a schedule, and also cannot provide an explanation for that “negative” result.

In this work, we present an extension of the aforementioned ASP-based approach to the ORS problem that is also able to provide explanations (in terms of input facts) that caused the absence of solutions. The explanation computation technique builds on the ideas employed by the ASPIDE debugger for ASP programs. Preliminary experimental results show the viability of the approach.

## Keywords

Answer set programming, Explainability, Operating Room Scheduling

## 1. Introduction

The increasing use of Artificial Intelligence (AI) methods in applications is affecting all parts of our lives and the need for explainable methods is becoming even more important. Even though AI-driven systems have been shown to outperform humans in certain tasks, the lack of explainability features continues to spark criticisms ([1, 2]). For these reasons, the improvement of explainability techniques for transparent models is an important research topic, especially for those AI strategies used in healthcare applications.

Answer Set Programming (ASP; [3, 4]) is a popular declarative AI language that has been widely used for solving healthcare problems (we refer the reader to [5] for a recent survey of such applications). Despite the declarative nature of ASP and its intuitive semantics (see [6] for the standard language), and the availability of efficient ASP systems ([7, 8, 9, 10, 11]), the development of an explainability layer on top of ASP systems is still subject of research and debate ([12]).


---

*IPS-RCRA 2021: 9th Italian Workshop on Planning and Scheduling and 28th International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion.*

✉ riccardo.bertolucci@unige.it (R. Bertolucci); dodaro@mat.unical.it (C. Dodaro); giuseppe.galata@surgiq.com (G. Galatà); marco.maratea@unige.it (M. Maratea); ivan.porro@surgiq.com (I. Porro); ricca@mat.unical.it (F. Ricca)  
ORCID 0000-0001-7356-1579 (R. Bertolucci); 0000-0002-5617-5286 (C. Dodaro); 0000-0002-1948-4469 (G. Galatà); 0000-0002-9034-2527 (M. Maratea); 0000-0002-0601-8071 (I. Porro); 0000-0001-8218-3178 (F. Ricca)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

ASP has been proved to provide effective solutions to practical applications [13], in particular in the healthcare domain (see, e.g., [14, 15]). As an example, ASP has been recently applied to solve the Operating Room Scheduling (ORS) problem ([16]), that is the problem of computing an assignment of patients to beds in operating rooms, a task that become even more relevant task during the COVID19 pandemic. During the validation of such a tool we recognized the need to unfold the decision-making process to users not specialized in AI. In particular, the ORS tool presented in ([16]) resulted to be effective in practice in computing schedules, especially when there are enough resources. However, when the available resources are not enough to satisfy user's requirements (e.g., insufficient number of free beds) the system cannot compute a schedule, and also cannot provide an explanation for that "negative" result. This makes the tool not fully acceptable in practice, since neither explanation nor a hint on how to solve the problem is provided.

In this work we present a framework for the generation of the needed explanations. In particular, our approach aims at generating explanations in case the instance modeling the ORS problem is incoherent, hence the system is not able to compute a solution. More in detail, our aim is to isolate the facts in the input that led to the incoherence: this procedure is called *facts checking*. Furthermore, we make this knowledge available in a readable fashion for inexperienced users such as medical staff. The explanation computation technique builds on the ideas employed by the ASPIDE debugger for ASP programs ([17]). Preliminary experimental results show the viability of the approach. In the following, we assume the reader is familiar with ASP.

## 2. Background on the ORS problem

In this paper, the elements of the waiting list are called *registrations*. Each registration links a particular surgical procedure, with a predicted surgery duration and length of stay in the ward and in the ICU, to a patient. The overall goal of the ORS problem is to assign the maximum number of registrations to the operating rooms (ORs), taking into account the availability of beds in the associated wards and in the ICU. Three requirements are respected to solve the ORS problem: (i) the assignments must guarantee that the sum of the predicted duration of surgeries assigned to a particular OR session does not exceed the length of the session itself. To distinguish between *registrations* referring to patients with different medical conditions a priority factor is used. (ii) It must be ensured that a registration can be assigned to an OR only if there is a bed available for the patient for the entire length of stay (LOS). (iii) Also the Intensive Care Unit (ICU) is considered: it is a particular type of ward that is accessible to patients from any specialty. It must be ensured that a registration can be assigned to an OR only if there is a bed available for the patient for the entire length of stay (LOS) in the ICU if necessary.

### 2.1. Data Model

Since our aim is to be able to identify facts that can lead to the inconsistency of the encoding, our focus will be in the input data given to the ORs scheduler. The input data is specified by means of the following atoms:

- Instances of *registration*( $R,P,SU,LOS,SP,ICU,A$ ) represent the registrations, characterized by an id ( $R$ ), a priority score ( $P$ ), a surgery duration ( $SU$ ) in minutes, the overall length of stay both in the ward and the ICU after the surgery ( $LOS$ ) in days, the id of the specialty ( $SP$ ) it belongs to, a length of stay in the ICU ( $ICU$ ) in days, and finally a parameter representing the number of days in advance ( $A$ ) the patient is admitted to the ward before the surgery.
- Instances of *mss*( $O,S,SP,D$ ) link each operating room ( $O$ ) to a session ( $S$ ) for each specialty( $SP$ ) and planning day ( $D$ ) as established by the hospital Master Surgical Schedule (*MSS*).
- The OR sessions are represented by the instances of the predicate *duration*( $N,O,S$ ), where  $N$  is the session duration.
- Instances of *beds*( $SP,AV,D$ ) represent the number of available beds ( $AV$ ) for the beds associated to the specialty  $SP$  in the day  $D$ . The *ICU* is represented by giving the value 0 to  $SP$ .

Details about the encoding can be found in [16].

### 3. Explainability layer

In this section, we show the integration of the ASP model for ORS and a debugging tool able to perform facts checking, with the goal of identifying the set of atoms modeling the input leading to the absence of a solution, i.e. to the inconsistency.

The idea of our approach is to implement a tool based on the work presented in [17] to identify the single or the set of facts atoms that led to the inconsistency of the encoding. Roughly, the tool described in ([17]) works by adding adornments atoms to the rules of the program, and then detecting a minimal set of adorned atoms that cause the inconsistency (reason for the inconsistency). We build on this idea and, instead of adorning all the rules of the program (which is provably correct), we apply the adornment only to input facts. Moreover, in order to improve the performances of the system, we perform some preprocessing on the atoms we are testing which consists on the selection of the atoms that are most likely to led to inconsistency. This selection was carried out under some assumptions: (i) The order in which the atoms are checked affects the performances. (ii) The faulty atoms typology can be: *beds*( $SP,AV,D$ ) or *duration*( $N,O,S$ ). When the atoms are identified, the system extracts the information from the atoms identified and use this information to give to the user the most complete answer on the cause of the impossibility to find a proper schedule.

### 4. Preliminary test

We tested our framework under the following assumptions: (i) the test were carried out with a single encoding with 5 different configuration of the data: 2 representing the data necessary for the schedule of 5 days of operations and 3 representing the data necessary for the schedule of 10 days of operations, (ii) for each configuration, we are measuring the computation time from the beginning to the moment in which the error is found, and (iii) the set of facts included in the search are subject to a *shuffle*, in which the atoms are shuffled each time a new search

begin, or to a *reverse*, in which the atoms are reversed once instead. Under these assumptions, we were able to identify two fact typologies that could lead to the inconsistency of the problem: (a) the lack of available beds in a certain specialty, given by facts of the type  $beds(SP,AV,D)$ , and (b) the duration of a certain operation is higher than the available time of each OR of each day in a certain specialty, given by facts of the type  $duration(N,O,S)$ .

As a sum up of our results, the reasons that can make a fact causing the incoherence are multiple. By analysing the knowledge represented by each fact, it is possible, for a domain expert, to categorize the source of the inconsistency: if the inconsistency is related to the number of beds available in a certain speciality  $SP$  in the day  $D$  its causes can be: (i) the lack of beds in the speciality in that specific day; or (ii) the number of patients to be scheduled in the speciality are too many; or (iii) the maximum time for the schedule is too short, or rather it is impossible to schedule all the patients in the amount of given days. If the inconsistency is related to the fact that duration  $N$  of a certain session  $S$  is higher than the available time of each OR of each day in a certain specialty, its causes can be: (i) the time that a patient must spend in an OR is too high, or (ii) the sum of the times that all patients, of a certain specialty ( $SP$ ), must spend in an OR is too high.

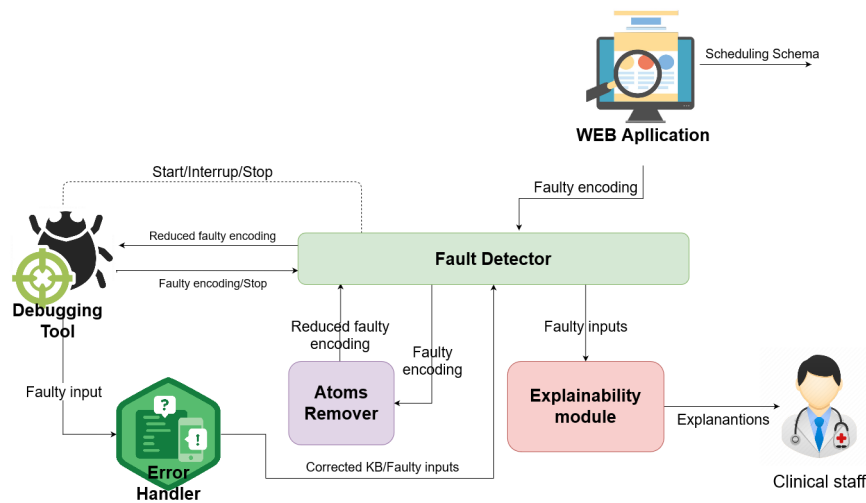
When all the faulty atoms are isolated, we are able to extract many useful information to retrieve to user in order to let her/him decide which solution fits best, such as: the specialty that is causing the fault; the list of patients with high priority in those specialities and all their data (surgery duration, length of the stay, etc); the day in which the fault occurs; and the number of beds available in that day in the given specialty.

With all these information, the user can choose the best way to solve the problem, e.g., increasing the number of beds, increasing the number of maximum days to schedule, decreasing the number of patients. The initial results gathered after the preliminary test shows that this approach is able to extract information from real-world scenarios data and generate explanations understandable by an inexperienced user.

#### 4.1. The explainability framework

In Figure 1, it is possible to observe the schema of our framework. The **Web Application** uses the encoding presented in [16] to compute a solution to the ORS problem. If a such a solution is found, then the optimal scheduling is reported as output to the user. Otherwise, if the given Knowledge Base ( $KB$ ) leads to the absence of a solution, then the faulty encoding is processed by our explainability framework.

The faulty encoding is sent to the **Fault Detector Module** which manages the data flow between the modules during the execution. Then, the **Atoms Remover module** removes all the low priority patients and all the weak constraints from the encoding since they cannot cause incoherence; therefore, they are not of our interest. After the preprocessing step, the faulty encoding is sent to the **Debugging Tool** module whose role is to identify the set of input facts leading to the inconsistency, i.e., it performs the facts checking. It is important to underline that we are searching the fault only inside the set of input facts given by the user since we assume that the encoding is properly written as we state in Section 3. The Debugging Tool operates within a given time limit, i.e., if the explanation is not found within such a threshold, then the debugging process is interrupted. In this case, the Faulty Detector Module performs a shuffle of



**Figure 1:** The system's architecture.

the *KB* atoms and starts a new facts checking process. This allows the tool to analyse always different atoms. This process is repeated until an explanation is found. It is important to notice that, during the preliminary test and analysis, we have noticed that shuffling the facts atoms before starting the facts checking process tangibly improves the performances. Moreover, we noticed that, under certain circumstances, reversing the *KB* could lead to great improvements of the performances.

When an explanation is found then faulty atoms are sent to the **Error Handler Module**, whose goal is to generate a new set of input atoms where faulty atoms are corrected. Since the facts checking process searches for the minimal set of facts leading to inconsistency, fixing one faulty atom is enough to solve the inconsistency. For this reason, the Error Handler Module forces the removal of one of the faulty facts. In this way, two scenarios are possible: (i) there are other sources of inconsistency, therefore the ORS problem admits no solution, or (ii) a solution can be found. In the first case, the new set of faulty atoms is sent back to the Fault Detector Module and the explanation process restarts with such a new set. In the second case, the inconsistency is fixed.

When the inconsistency is fixed, the **Explanability Module** extracts and processes all the information from the list of faulty atoms in order to make them available in a comprehensible form to the user. In our case, the explanations are text messages aimed at making various information available to the user. In particular, since the user is assumed to be totally inexperienced with ASP, the generated messages are designed to put all the extracted information in the right context so that the meaning of the given information is clear to the user.

**Message Example.** *The surgery time necessary to operate the patients of speciality 2 on day 10 exceeded the maximum time of operation at disposal of 30 hours for the operating room 5. Please increase the*

maximum operating time on some operating room or reduce the number of patients assigned to the faulty speciality.

The patients that might generate the error are:

- Patient id: 2001                      Speciality: 2  
Total stay: 12 days                  Days at the ICU: 0  
Days in ward before surgery: 2      Surgery duration: 10 hours
  
- Patient id: 2011                      Speciality: 2  
Total stay: 12 days                  Days at the ICU: 0  
Days in ward before surgery: 2      Surgery duration: 11 hours
  
- Patient id: 2014                      Speciality: 2  
Total stay: 6 days                    Days at the ICU: 0  
Days in ward before surgery: 2      Surgery duration: 10 hours

## 5. Results

In order to obtain an assessment of the capabilities of the developed solution, we generated three different sets of facts. Each set represent the available resources of an hospital for a 10 day scheduling with the list of the patients to which is necessary to assign an OR. Each set is composed by 600 atoms approximately and they are designed to be solvable. In our analysis, 1 to 7 sources of inconsistency were forced in each set and each one is tested on the architecture 10 times. Observe that for any iteration the inconsistencies are forced in different input atoms.

For each iteration, a time limit of 60 seconds and a memory limit of 16 GB was applied. All the experiments have been conducted on an Intel i7-4790 CPU and Linux OS. Clingo was used to solve the ASP-encoded instances.

We compared the performance of the considered encodings using *coverage*, i.e., the percentage of solved instances, and the Penalised Average Run-time (PAR10) score. The latter is a metric usually exploited in machine learning and algorithm configuration techniques. It trades off coverage and run-time for solved problems: if an encoding  $e$  allows the solver to solve an instance  $\Pi$  in time  $t \leq T$  ( $T = 60$  s in our case), then  $PAR10(e, \Pi) = t$ , otherwise  $PAR10(e, \Pi) = 10 \times T$  (i.e., 600s in our case). It is important to point out that the errors identified by the architecture may vary from the errors forced in the input set, i.e., if we remove beds from a speciality SP in a day D, then the architecture could suggest us to add beds for the speciality SP but in a day D'. However, as it is possible to observe from the results (shown in table 1), our system is able to identify the sources of the unsatisfiability under the 60 seconds timeout only if in the input sets are forced 6 or less errors. It is important to underline that in this paper we are not going to evaluate the understandability of the generated explanations; for a detailed study in the field of the social sciences we refer to ([1]).

## 6. Conclusions

In this paper, we have presented an approach to explaining the outcome of an ASP-based solution to the problem of operating room scheduling. The objective is to explain why, in

#	PAR10			Coverage		
	Set 1	Set 2	Set 3	Set 1	Set 2	Set 3
1	0.57	0.51	0.60	100.0	100.0	100.0
2	0.81	0.9	0.67	100.0	100.0	100.0
3	14.71	17.12	16.45	100.0	100.0	100.0
4	25.43	26.77	22.96	100.0	100.0	100.0
5	31.14	30.67	93.15	100.0	100.0	90.0
6	164.04	50.34	110.47	80.0	100.0	90.0
7	600.0	600.0	600.0	0.0	0.0	0.0

**Table 1**

Results, in terms of PAR10 and coverage, achieved by the considered encodings on the tested instances. Instances are grouped according to the source set (rows) and the number errors forced in the facts (source set, number of errors).

a certain situation, no appropriate schedule could be found, in other words, why no answer set could be computed. The preliminary tests show that the presented approach is able to identify the facts which led to the inconsistency of the solution. However, the generation of the explanations is highly dependent on the treated domain: in order to be able to present an explanation comprehensible to inexperienced users, the knowledge represented by the faulty facts must be analysed by a domain expert with a deep understanding of the encoding. As we stated in Section 3, the presented approach is able to find the minimal set of adorned atoms. Since a single faulty fact is enough to lead to the inconsistency, our strategy will be able to find one faulty fact at the time. To overcome this problem and being able to find the complete set of faulty facts, we included in our architecture a module that handles the inconsistency: the process used in this model is highly dependent by the domain. However, the process of finding a single faulty fact is applicable to any domain. Moreover, the explanation generated by our approach takes under consideration only the input facts: to generate exhaustive explications it is necessary to include this approach into a platform of solutions, which consider also rules, similarly to, e.g., [18] in other contexts. We are also currently working on extending our preliminary experiments. On the side of ASP computation, we would like to implement and test other solving procedures, e.g., [19, 20, 21, 22], considering the relation between ASP and SAT procedures [23, 24], whose goal would be to improve the current performance.

## References

- [1] T. Miller, Explanation in artificial intelligence: Insights from the social sciences, *Artificial intelligence* 267 (2019) 1–38.
- [2] A. Abedini, H. Ye, W. Li, Operating room planning under surgery type and priority constraints, *Procedia Manufacturing* 5 (2016) 15–25.
- [3] G. Brewka, T. Eiter, M. Truszczyński, Answer set programming at a glance, *Communications of the ACM* 54 (2011) 92–103.
- [4] F. Van Harmelen, V. Lifschitz, B. Porter, *Handbook of knowledge representation*, Elsevier, 2008.



- [5] M. Alviano, R. Bertolucci, M. Cardellini, C. Dodaro, G. Galatà, M. K. Khan, M. Maratea, M. Mochi, V. Morozan, I. Porro, et al., Answer set programming in healthcare: Extended overview., in: IPS-RCRA@ AI\* IA, 2020.
- [6] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, T. Schaub, ASP-Core-2 input language format, *Theory and Practice of Logic Programming* 20 (2020) 294–309.
- [7] M. Gebser, B. Kaufmann, T. Schaub, Conflict-driven answer set solving: From theory to practice, *Artificial Intelligence* 187 (2012) 52–89.
- [8] M. Alviano, G. Amendola, C. Dodaro, N. Leone, M. Maratea, F. Ricca, Evaluation of disjunctive programs in WASP, in: M. Balduccini, Y. Lierler, S. Woltran (Eds.), LPNMR, volume 11481 of *LNCS*, Springer, 2019, pp. 241–255.
- [9] F. Calimeri, M. Gebser, M. Maratea, F. Ricca, The design of the fifth answer set programming competition, CoRR abs/1405.3710 (2014). URL: <http://arxiv.org/abs/1405.3710>. arXiv:1405.3710.
- [10] M. Gebser, M. Maratea, F. Ricca, The design of the seventh answer set programming competition, in: M. Balduccini, T. Janhunen (Eds.), LPNMR, volume 10377 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 3–9.
- [11] M. Gebser, M. Maratea, F. Ricca, The seventh answer set programming competition: Design and results, *Theory and Practice of Logic Programming* 20 (2020) 176–204.
- [12] J. Fandinno, C. Schulz, Answering the “why” in answer set programming—a survey of explanation approaches, *Theory and Practice of Logic Programming* 19 (2019) 114–203.
- [13] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, *AI Magazine* 37 (2016) 53–68.
- [14] C. Dodaro, M. Maratea, Nurse scheduling via answer set programming, in: LPNMR, volume 10377 of *LNCS*, Springer, 2017, pp. 301–307.
- [15] M. Alviano, C. Dodaro, M. Maratea, An advanced answer set programming encoding for nurse scheduling, in: AI\*IA, volume 10640 of *LNCS*, Springer, 2017, pp. 468–482.
- [16] C. Dodaro, G. Galatà, M. K. Khan, M. Maratea, I. Porro, An ASP-based solution for operating room scheduling with beds management, in: International Joint Conference on Rules and Reasoning, Springer, 2019, pp. 67–81.
- [17] C. Dodaro, P. Gasteiger, K. Reale, F. Ricca, K. Schekotihin, Debugging non-ground asp programs: Technique and graphical tools, *Theory and Practice of Logic Programming* 19 (2019) 290–316.
- [18] A. Armando, C. Castellini, E. Giunchiglia, M. Idini, M. Maratea, TSAT++: an open platform for satisfiability modulo theories, *Electronic Notes in Theoretical Computer Science* 125 (2005) 25–36.
- [19] E. Giunchiglia, M. Maratea, A. Tacchella, D. Zambonin, Evaluating search heuristics and optimization techniques in propositional satisfiability, in: R. Goré, A. Leitsch, T. Nipkow (Eds.), International Joint Conference on Automated Reasoning (IJCAR 2001), volume 2083 of *Lecture Notes in Computer Science*, Springer, 2001, pp. 347–363.
- [20] E. Giunchiglia, M. Maratea, A. Tacchella, Dependent and independent variables in propositional satisfiability, in: S. Flesca, S. Greco, N. Leone, G. Ianni (Eds.), JELIA, volume 2424 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 296–307.
- [21] E. Giunchiglia, M. Maratea, A. Tacchella, (In)Effectiveness of look-ahead techniques in



- a modern SAT solver, in: F. Rossi (Ed.), CP, volume 2833 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 842–846.
- [22] E. D. Rosa, E. Giunchiglia, M. Maratea, A new approach for solving satisfiability problems with qualitative preferences, in: M. Ghallab, C. D. Spyropoulos, N. Fakotakis, N. M. Avouris (Eds.), ECAI, volume 178 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2008, pp. 510–514.
- [23] E. Giunchiglia, M. Maratea, On the Relation Between Answer Set and SAT Procedures (or, Between cmodels and smodels), in: ICLP, volume 3668 of *LNCS*, Springer, 2005, pp. 37–51.
- [24] E. Giunchiglia, N. Leone, M. Maratea, On the relation among answer set solvers, *Ann. Math. Artif. Intell.* 53 (2008) 169–204.