UNIVERSITY OF GENOVA

PHD PROGRAM IN COMPUTER SCIENCE AND SYSTEM ENGINEERING

Solving and Explaining Scheduling problems in Digital Health using Logic Programming

by

Marco Mochi

Thesis submitted for the degree of *Doctor of Philosophy* (37° cycle)

February 2025

Prof. Marco Maratea	Supervisor
Prof. Enrico Giunchiglia	Supervisor
Prof. Giorgio Delzanno	Head of the PhD program

Thesis Jury:

Mauro Vallati, *University of Huddersfield*	External examiner
Marco Gavanelli, *University of Ferrara*	External examiner
Giorgio Delzanno, *University of Genova*	Internal examiner

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are my own work and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university.

Marco Mochi
February 2025

Abstract

The importance of Digital Health has been increasing in recent years. This is due to the development of technologies and new challenges in the healthcare sector. The world's population is increasing and becoming older, moreover, according to the World Health Organization, in the near future there will be a shortage of medical staff of several million. In this thesis, we define and propose a solution to three healthcare scheduling problems using Answer Set Programming (ASP), a declarative language used in different complex scheduling problems. Having presented these solutions, which are representative of the way in which ASP could improve and help the Hospital organization, we present another important problem that arises with the usage of AI: Explainable Artificial Intelligence (XAI). XAI is a field of AI that tries to create human-understandable solutions. In the healthcare domain, proposing a black-box model as a solution will not be enough in the future, both because the patients and the operators need to know how and why a certain solution and a certain decision was made and because the European Commission has established, with the General Data Protection Regulation, that each person has the right to ask for an explanation of the decision taken by an AI. Without developing Explainability methodologies the usage of AI based solvers will be limited, thus, both for ethical and legal reasons the implementation of explainability techniques will be crucial in all the fields in which AI could be applied and even with more urgency in the healthcare domain. To these means, we present two tools with different objectives. The first one is an explainability tool, E-ASP, that is used to explain the reason why a solution has been given. E-ASP can help users receiving a solution to understand why certain results were obtained thus allowing the usage of ASP in Safety-critical domains, such as the healthcare sector. The last tool we present is CNL2ASP, which is a translation tool that translates from Controlled Natural Language sentences to an ASP encoding. The tool is used to ease the usage of ASP even for non-experts users and speed up the prototyping of ASP models. Finally, another important topic in AI and Digital Health is addressed, it is the problem of Fairness. In particular, in the thesis, we will propose a solution to overcome the Fairness problem in one of the presented scheduling problems.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Context and Motivation

One of the first times the term Digital Health was used was in 2000 by Seth Frank in Frank (2000), where he anticipated that the spread of technology such as Internet and the development of ad hoc applications would cause a greater impact on medical care than any previous information technology. Since the first time the term was coined, new information technologies have been developed and through the years the meaning of it became broader. Indeed, he saw Digital Health as a set of applications used to increase connectivity and the promotion of health. Whereas, in current years, the term has been used to encompass many different applications involving healthcare, such as genomics, artificial intelligence, mobile applications, and so on. A more modern and broader definition of what Digital Health means considers Digital Health as the use of information technologies, computer science, and data to help health workers, and health facilities in general, to make informed decisions that can increase the level of care, the organization, and to improve the resilience of the system. To certify the big landscape of modern Digital Health and its increasing importance, the World Health Organization (WHO) has produced many documents in recent years defining the different meanings of the term Digital Health and how it can improve the level of care (World Health Organization, 2018, 2023). As in other fields, there has been a rise in AI applications in the healthcare sector in recent years. Again, the WHO attested the increasing importance of AI in Digital Health by publishing different reports and articles (World Health Organization, 2024) and commented that "...Prioritizing AI for health is crucial, given its potential to enhance healthcare and address global health challenges...". Digital Health gained further increasing importance in recent years. This happened thanks to new technologies and also due to new challenges such as an aging society, the COVID-19

pandemic, and the need to reduce high costs. Moreover, according to the WHO, while the potential for digital technologies and innovation to enhance population health remains largely underutilized, it still can have an important role in achieving the Sustainable Development Goals (World Health Organization, 2021).

Indeed, as healthcare systems become more complex and patient demand continues to rise, AI offers promising solutions to manage and optimize healthcare processes, including scheduling. Scheduling in healthcare is particularly important as it affects many aspects from patient satisfaction to the effective use of critical resources such as operating rooms (ORs), beds, and medical staff.

Among the main problems related to modern hospitals is long waiting lists that reduce patients' satisfaction and the level of care offered to them. Moreover, surgery cancellations and resource overload negatively impact the level of patient satisfaction and the quality of care provided. Within every hospital, ORs are an important unit. As indicated in Meskens et al. (2013), the OR management accounts for approximately 33% of the total hospital budget because it includes high staff costs (e.g. surgeons, anesthetists, nurses) and material costs. In most modern hospitals, long surgical waiting lists are present because of inefficient planning. Therefore, it is of paramount importance to improve the efficiency of OR management, to enhance the survival rate and satisfaction of patients, thereby improving the overall quality of the healthcare system. In this regard, other problems in modern hospitals are related to the management of different resources, such as medical staff and medical equipment, involved in different problems such as the assignments of pre-operative exams or the assignments for patients needing CT scans. Optimizing the assignments for the patients under these resource constraints is very important. Indeed, a patient requiring surgery can need several exams, from different specialties, to ensure they are well-prepared for their operation. Optimizing the schedule of the exams required by the patients could prevent the patients from being admitted to the hospital one or two days before the scheduled operation and allow the patients to stay at home until the morning of the surgery. Moreover, this can reduce waiting time between the exams too, thus increasing patients' satisfaction (Harnett et al., 2010), and reducing the risk of the cancellation of the surgery (Ferschl et al., 2005). Finally, it will be more and more important to have solutions that can optimize the resources of the hospitals, indeed, the number of patients having chronic conditions is increasing, and life expectancy is getting higher. For these reasons, the WHO calculates a shortage of 10 million workers in the world's healthcare sector by 2030.

In this scenario, AI-based solutions could make a profound difference. AI-driven scheduling systems can optimize resource allocation, reduce waiting times, and enhance patient's

satisfaction. For instance, AI can take into account factors such as patient priority, surgeon availability, and bed capacity to generate optimized schedules that meet the demands of both patients and healthcare providers. Such systems are not only more efficient but also allow for greater flexibility and scalability, making them suitable for large healthcare networks.

However, the challenge of successfully integrating AI into Digital Health is not completed not only because many unsolved problems remain but also due to the difficulties the governments are facing in regulating these rapid technological changes. Indeed, while digital technologies have changed or are changing significantly thanks to AI, the health system is having some problems implementing the new technologies (Alami et al., 2017).

In recent years, there has been a rising demand for AI systems capable of not only providing answers but also explaining the reasoning behind their decisions. This transparency is essential for establishing trust, as people want to understand how and why AI systems work. Furthermore, in contexts such as finance and healthcare, where there are strict regulations, providing explanations for AI solutions is imperative for compliance. Moreover, the European Union released a new General Data Protection Regulation (GDPR) (Parliament and Council of the European Union 2016), stating how personal data can be processed. The GDPR implies that, in the future, anyone will have the right to reject a decision taken by an AI if no explanation is provided. This will be true regardless of the accuracy of the "black-box" model, which can often hide the reasons behind the outcome. Unlike black-box models, which prevent users from verifying or fully understanding the decision-making process, approaches that prioritize transparency offer significant advantages, particularly in critical domains where trust is vital. Explainability is crucial for ensuring that users, including non-technical users, can interpret and validate the solutions produced by AI systems. This interpretability builds confidence in the technology, as users can see not only what decisions are made but also why those decisions are reached. For these reasons, Knowledge Representation and Reasoning (KRR) languages that are explainable by default can be considered a viable tool for providing AI-based solutions. Unlike other AI methods, KRR systems rely on logical rules to make deductions, making their decision-making paths inherently provable. Indeed, many KRR systems, having the solving phase based on tree search, are explainable by design and thanks to this feature it is possible to reconstruct an explanation, but computing it is not easy. Indeed, different methods to reconstruct an explanation have been developed, each focusing on different aspects. Nevertheless, many works still highlight the lack of a clear definition of what it means to "explain" a solution and tried to define it (Miller, 2017; Mittelstadt et al., 2018). Thus, different techniques were developed. For example, some authors focused their work on "Contrastive explanations" (Miller, 2017) or "Justification" (Cabalar et al., 2014). In

particular, in the works focused on "Contrastive explanations", the explainee is interested in knowing why a certain event occurred instead of another one or in knowing why with the same model and with two similar inputs the occurred events were different, while, in the works that deal with Justification, the model explains an event by observing all the facts or events that caused the event to be true. Thus, the importance of working on explainable tools and providing new clear definition of what is an explanation is important for increasing the usage of such AI systems. Given the increasing adoption of KRR systems, it is also essential to explore tools that improve the accessibility of the solutions for a broader range of users. In this context, for certain types of users the use of a higher-level language that is closer to natural language for specifying and presenting programs could be preferable. Thus, the use of Controlled Natural Languages (CNLs) have gained significant attention. A CNL provides a bridge between formal logics and human language, making it possible to specify logic rules in controlled English. This approach makes KRR systems not only more approachable but also more interpretable for those unfamiliar with formal logic, such as domain experts or end-users who may not have a technical background in programming. For this reason, in the last decades, a number of attempts to convert English sentences expressed in a CNL into KRR formalisms emerged (Clark et al., 2005; Fuchs, 2005). Another main advantage of using a KRR approach is the intrinsic modularity. Allowing for flexible solutions and allowing the addition of further rules to a starting knowledge base. In the context of AI and critical domains such as Digital Health, this modularity can be very significant when dealing with Fairness problems. Indeed, modularity allows for the inclusion of fairness or ethical constraints without rewriting entire models and maintaining the core functionalities of the original model.

1.2 State of The Art

Logic Based Solutions for Scheduling Problems. Traditional approaches for complex scheduling problems in Digital Health include Mathematical Programming techniques, such as Mixed Integer Programming (Heshmat and Eltawil (2021); Huggins et al. (2014); Pérez et al. (2011); Zhang and Xie (2015)) and metaheuristic programmings such as Simulated Annealing (Edward et al. (2008)) or Genetic Algorithm (Xiao et al. (2018)). Other approaches include the usage of Stochastic Dynamic Program as in Akhavizadegan et al. (2017); Landa et al. (2016); Zhang et al. (2017) and Constraint Programming as in Alade and Amusat (2019); Di Gaspero and Urli (2014); Öztürkoğlu (2020). Differently from these Logic-based techniques, KRR systems offer some advantages such as the presence of high-performing

open source systems, that often have similar performances to the industrial tools, the ease of reason on multi-level optimizations, and a clear programming methodology. Moreover, some of such KRR approaches, such as Answer Set Programming (ASP) offer another advantage over traditional optimization methods due to the declarative nature, which allows for a higher readability of the models, even by non-experts, differently from the specifications employed by the other paradigms, e.g., SAT and CP.

Explainable AI, Clarity and Fairness. Despite the recent increase in interest towards Explainable Artificial Intelligence (XAI) many works still highlight the lack of a clear definition of what it means to "explain" a model and tried to define it (Marques-Silva, 2024; Miller, 2017; Mittelstadt et al., 2018). While many works have been done on studying explainability methods for Machine Learning and "black-box" models (e.g., Linardatos et al. (2020)), these works have to deal with the limitations of the systems, thus, an explainability is often given in the form of most "important" feature (Lundberg and Lee (2017)) and even the so-called interpretable models present some problems (Marques-Silva and Ignatiev (2023)), e.g., providing not always a sufficient explanation, and require a great amount of data. Moreover, other works highlighted the problems of using not rigorous methods to explain critical-domain applications, thus, advocating for the usage of logic-based formalism (Marques-Silva and Huang (2024)). Certainly, these methods do not come without any drawbacks, including the computational complexity and the scalability problems on large problems. However, a declarative language such as ASP, and its high-level syntax, can be more readable for non-expert users, allowing for a better clarity of the developed tools. Despite its clearer syntax, in certain domains, an even more high-level language could be preferred. Moreover, since some KRR language adds a declarative nature, which allows a natural mapping between their syntax and natural language, in recent years, different works proposed various attempts to convert English sentences expressed in a CNL into KRR formalisms (Clark et al., 2005; Fuchs, 2005). In the context of ASP, a CNL has been used for solving logic puzzles (Baral and Dzifcak, 2012), and for answering biomedical queries (Erdem and Yeniterzi, 2009). Finally, different works analyzed the importance of Fairness in contexts such as Digital Health and AI systems. Among the works addressing the problem in general, (Ferrara, 2023) provides a comprehensive survey on fairness and bias in AI, addressing their sources, impacts, and mitigation strategies. While, in Ueda et al. (2024) the authors examine the challenges of fairness in the clinical integration of AI in medicine. They provide a comprehensive review of concerns related to AI fairness and discuss various strategies to mitigate biases in AI-based healthcare systems.

1.3 Contributions

In this thesis, to solve the different problems we rely on the declarative programming language Answer Set Programming (ASP) (Baral, 2003; Brewka et al., 2011; Gelfond and Lifschitz, 1988; Janhunen and Niemelä, 2016). ASP has emerged as a powerful declarative programming paradigm, offering a flexible framework for addressing complex combinatorial problems. A more detail presentation of ASP and its advantages is in Chapter 2.

The contributions of this thesis regard two different but interconnected areas: scheduling in Digital Health with AI and explainability of AI and ease of the usage of AI tools to schedule problems. Our contributions presented in the thesis are:

- We propose a solution to the PRE-OPERATIVE ASSESSMENT CLINIC problem, which consists of assigning pre-operative exams to a list of patients while assigning the medical staff to the different medical areas. The complexity of the problem arises from the necessity to schedule the patients and the medical staff together, taking into account the availability of the different medical machines and the medical staff shifts, according to the exams needed by the patients. We tested the solution with synthetic data and compared it to other logic-based formalisms. Moreover, we proposed a solution to the rescheduling problem, testing our solution in different scenarios taking into account the unavailability of a doctor or a patient.

- We propose a solution to the OPERATING ROOM SCHEDULING problem, where the patients must be assigned to an operating room, to a date and an hour according to their required operation. We further proposed a solution that improved previous results and tested it using real data. Finally, we compared it to other logic-based formalisms.

- We propose a solution to the NUCLEAR MEDICINE SCHEDULING problem, concerning the assignments of patients requiring a CT-scan to a date and hour. Each patient has to follow a strict protocol composed of different phases of varying lengths. Moreover, in each phase, a resource, such as an injection chair or tomograph could be required. We defined the problem following the definition of a real hospital and tested the solution using real data.

- We define the theoretical aspects and present a tool that allows us to analyze the solutions of an ASP encoding. The tool, E-ASP, allows us to identify the set of rules justifying a solution. Moreover, E-ASP is able to provide explanations over aggregates via a stepping-through approach, enhancing its utility in complex scenarios.

- We propose CNL2ASP, that allows the translation of a Controlled Natural Language to an ASP encoding, allowing non-expert users to get an encoding, without being expert in the ASP syntax. Moreover, we present examples of how such a tool can be used with real-world problems and perform an analysis of the usability of the tool.
- Starting from the Nuclear Medicine Scheduling problem and exploiting the modularity of ASP, we tackled the Fairness problem and proposed an encoding that can be modularly added to the original one to reduce the biases of the AI solution in such a problem. We performed an experimental analysis to assess the quality of our solution.

1.4 Outline

After having introduced in this chapter the problem tackled in the thesis, in the following one, we will present the tool that we will use, presenting the syntax, semantics, and methodology of ASP. The thesis then is split into three parts.

Digital Health Scheduling Problems. The first part of the thesis presents the scheduling problems in Digital Health and is composed of Chapter 3, Chapter 4, and Chapter 5 that present the Pre-Operative Assessment Clinic problem, the Operating Room Scheduling problem, and the Nuclear Medicine Scheduling problem, respectively.

Increase Explainability and Fairness. The second part of the thesis is devoted to presenting two solutions to ease the usage of ASP through an explanation tool in Chapter 6 and an automated translator from natural language to ASP in Chapter 7, and to assess the problem of Fairness in Chapter 8.

Conclusions. Finally, the last part of the thesis concludes with a brief presentation of a selection of works, not concerning explicitly Digital Health, that were tackled during the thesis in Chapter 9, the presentation of the related works in Chapter 10, and the conclusion in Chapter 11.

Chapter 2

Background on Answer Set Programming

2.1 Answer Set Programming

In this section, we first overview the language of ASP, by presenting syntax and semantics. Then, we describe the ASP programming methodology. ASP is a declarative programming language based on the stable model semantics (Gelfond and Lifschitz, 1991). Its significance lies in its ability to effectively model a diverse range of real-world scenarios, making it particularly useful in various domains. Like other KRR systems, the appeal of ASP is related to a user-friendly syntax and an intuitive semantics, differently from the specifications employed by the other paradigms, e.g., SAT. This is important also when explainability features come into play. Moreover, it is fully modulable and explainable by design, and the advantages of ASP include the availability of many efficient systems, which have been refined through extensive research and development. These systems are free and open source (like Clingo Gebser et al. (2016), or Wasp Alviano et al. (2019a)), whose performances are often comparable to the ones of industrial tools for Integer Linear Programming like, e.g. Gurobi, or MaxSAT solvers. In recent years, ASP has become increasingly popular for solving complex combinatorial problems in both academic and industrial settings (Erdem et al., 2016; Falkner et al., 2018). (see, e.g., Alviano et al. (2019b); Alviano and Dodaro (2016); Gebser et al. (2016, 2018a); Maratea et al. (2014)). A solution based on ASP has been proposed for solving hard combinatorial problems in several research areas, including Artificial Intelligence (Amendola et al., 2016; Balduccini et al., 2001; Dodaro et al., 2015b), Bioinformatics, Data Integration and Query Answering, Natural Language Processing and Understanding, and Hydroinformatics (Gavanelli et al., 2015), and it has been also employed to solve many scheduling problems (Abels et al., 2019; Alviano et al., 2018; Cardellini et al., 2021; Dodaro et al., 2019a; Ricca et al., 2012), also in industrial contexts (see, e.g.,

Erdem et al. (2016); Falkner et al. (2018); Schüller (2018) for detailed descriptions of ASP applications). About the language, more detailed descriptions and a more formal account of ASP can be found in Brewka et al. (2011) and in Calimeri et al. (2020a).

2.1.1 Syntax and Semantics

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. The most common functions implemented in ASP systems are the following:

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

An *aggregate atom* is of the form $f(S) \prec T$, where $f(S)$ is an aggregate function, $\prec \in \{<, \leq, >, \geq, \neq, =\}$ is an 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:

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

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 $a_1 \mid \ldots \mid a_n$ 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* Buccafurri et al. (2000) ω is of the form:

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

where w and l are the weight and level of ω, respectively. (Intuitively, $[w@l]$ is read as "weight w at level l", where the weight is the "cost" of violating the condition in the body of ω, 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* of P, denoted as U_P, is the set of all constants appearing in P, while the *Herbrand base* of P, denoted as B_P, is the set of all ground atoms constructible from the predicate symbols appearing in P and the constants of U_P.

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) w.r.t. 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* Faber et al. (2011) 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) Faber et al. (2011).

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.

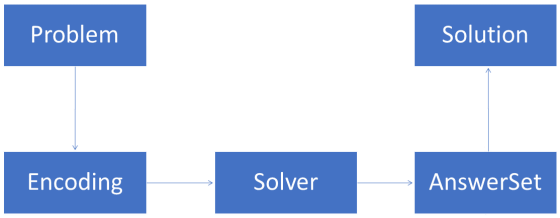

Figure 2.1 Programming methodology schema.

2.1.2 Programming Methodology

Figure 2.1 depicts a representation of a solution based on a logic-based declarative programming approach, as our ASP solution, consisting of five blocks as described above.

- Problem: this block represents the problem description or formulation to be modeled and solved.
- Encoding: this block involves the formal representation of the problem, using ASP in our case, based on the informal description of the problem or the precise mathematical formulation provided.
- Solver: this block takes the encoding of the problem as input and generates the answer sets.
- AnswerSet: this block represents the output of the solver and corresponds to the set of atoms that satisfy all the rules of the encoding, according to the semantics given above.
- Solution: this block is the solution of the problem, in which the answer sets are interpreted as solutions of the input problem.

The presence of a clear programming methodology is, arguably, one of the advantages that ASP offers with respect to other logic-based paradigms. Others include: (i) The ASP high-level specifications are declarative and often appreciated even by non-experts since they

found them readable, differently from the specifications employed by the other paradigms, e.g., SAT. (ii) There are free and open source systems (like the mentioned CLINGO, or WASP Alviano et al. (2019a)), whose performances are often comparable to the ones of industrial tools for ILP like, e.g. CPLEX, or to GUROBI, or SAT solvers (as shown in our experiments). (iii) ASP allows for easily expressing and reasoning on multi-objective and multi-level optimizations, which is not the case for, e.g., optimization variants of SAT such as Max-SAT (unless weights having exponential gaps are applied).

On the other hand: (a) Being a declarative approach, there is less control on the solving, which is delegated to an ASP solver. (b) Some CP constraints, such as *alldifferent*, that may be useful in applications, are not part of the language, and can not be expressed in a compact way; CPLEX and GUROBI allow for expressing quadratic, non-linear, functions in the optimization statement, which are not part of the ASP standard and ASP solver can not solve (at least in a direct way).

Part I

Digital Health Scheduling Problems

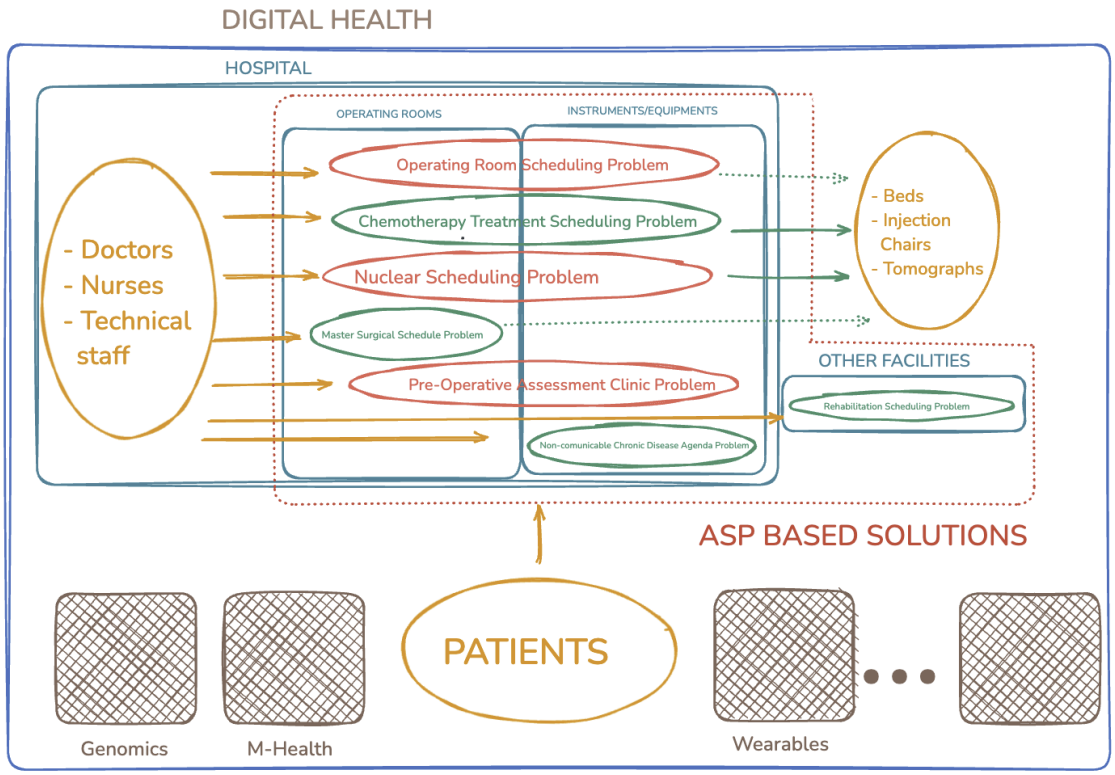

Figure 2.2 The image presents a schema representing how the scheduling problems interact with the hospital staff, the hospital resources and the patients.

Figure 2.2 presents a schematic representation of how the different Digital Health problems, with the problems presented in this thesis colored in red, interact with the different components of the Digital Health landscape. The figure highlights that scheduling problems are part of the whole landscape, that is formed by many other fields such as Genomics, M-Health, and many others. All these fields work taking into account different problems and aspects of the healthcare sector. In this case, the involved components are (i) the doctors and nurses, who must be taken into account in many problems, thus, in the schema, the arrows starting from the medical staff mean that they are part of the input of the problem; (ii) the resources of the hospital such as the ORs and the instruments of a hospital or other resources, depending on the problem and its variation. These resources can be the injection chairs, beds, and tomographs. In the schema the arrows start from the problems and end with these resources: this is done to highlight that these problems have to assign these resources and that they are part of the output; (iii) the patients, which are in may problems the input to assign.

To better appreciate the importance and the contribution that an AI-based solution could bring, here we briefly present a description of how the problems interact with the different components and resources and the usage of the ASP solutions.

In the schema, all the presented problems have to deal with the doctors and the medical staff in general. Some of the problems, such as a variation of the Operating Room Scheduling Problem (Dodaro et al., 2020), the Rehabilitation Scheduling Problem (Cardellini et al., 2024), and the Master Surgical Schedule Problem (Mochi et al., 2023), consider the medical staff explicitly, by scheduling the different objectives taking into account the explicit medical staff. However, all these problems, consider, even implicitly, the medical staff and their availability (Gavanelli et al., 2015). Without an automated solution, the scheduler should consider the problem by looking at the availability of the medical staff or the available slots (if the medical staff is considered implicitly). Meanwhile, a human scheduler should take into account, for many problems, the resources involved, such as the beds or the injection chairs. These resources are the main core of many problems and, thus, a manual scheduler consider often just the availability of these resources. But, not taking into account other constraints and limitations could lead to sub-optimal schedules or, even worse, solutions that are not feasible in practice. Indeed, in many of the analyzed problems such as Dodaro et al. (2021), we found that the human-based schedule had to resort to virtual resources or similarly overlook some resources, leaving the doctors and the nurses to handle the problem, further adding to the already heavy burden carried by the medical staff. Finally, at the core of all these problems is the patient, whose needs and well-being drive the resolution of each problem and each sub-optimal solution or infeasible solution is translated to lower care of the treatments and higher costs.

Thus, an AI-based solution using a language such as ASP can drastically help the hospitals to manage the schedule of these problems by taking into account all the different aspects of the problems, from the doctors and the technical staff to the resources involved in the problems. Moreover, using a proper solution to the problems is not only helpful due to the better solutions obtained, but also due to the fact that can leave more time to the doctors and nurses to actually treat the patients, without forcing them to schedule the patients.

In this first part of the thesis, we will present three problems we tackled regarding the scheduling of Digital Health problems. These problems represent different kinds of problems, concerning different resources and having different kinds of specifications and structures, allowing us to highlight the potentiality and the versatility of an AI-based solution. The first problem is the "Scheduling Pre-Operative Assessment Clinic Problem" (PAC), where a list of patients requiring some pre-operative exams should be assigned to a day to perform all the

exams. Moreover, these exams can not be done without properly managing the schedule of the doctors in the different days and specialties of the hospital. The second one is a variation of the "Operating Room Scheduling" (ORS) problem, as described by the hospitals of ASL1 in the Liguria region, Italy. The primary goal of the ORS problem is to assign as many patients as possible from a waiting list to the appropriate Operating Rooms. In this way, we can ensure the most efficient use of OR time, which is a very valuable resource. The last one we present is the "Nuclear Medicine Scheduling" (NMS) problem, as described by Medipass[1], leading provider of technological innovation across cancer care and diagnostic imaging in Italy. The problem consists of assigning a starting time for the CT-scan of the patients requiring it, considering all the different phases required by the patients before it, e.g. the admission and the injection of a drug. Indeed, some patients could require the injection of a drug before the actual scan, and the infusion can be performed in an injection chair or a tomograph, depending on the kind of patient. The objective of the problem is to assign the highest number of patients and reduce the waiting times between the different phases. All the problems are related to one of the most important problems of modern hospitals: long waiting lists. Indeed, in these problems, while in different contexts, the main aim of the schedule is to assign the highest number of patients, while satisfying all the constraints. For all the problems, an ASP-based solution is proposed and it is analyzed through extensive experimental analysis. The analysis is done utilizing realistic data from a middle Italian hospital for the first presented problem, whereas, in the second and the third one, it was possible to test our solution using real data. Finally, to confront ASP to other approaches, we compared our solutions to the ORS and the PAC problems to different logic-based formalisms, such as MaxSAT and the industrial tool GUROBI.

[1] https://ergeagroup.com/it/

Chapter 3

Scheduling Pre-Operative Assessment Clinic Problem

The Pre-Operative Assessment Clinic (PAC) scheduling problem involves assigning patients to a specific day for their pre-surgical examinations and preparations. This task must account for factors such as varying patient priority levels, due dates, and the availability of medical staff. The PAC process includes a series of essential tests to ensure patients are adequately prepared for their surgery. By conducting these exams beforehand, patients can remain at home until the morning of their surgery, avoiding the need for hospital admission one or two days in advance; moreover, this allows to reduce waiting time between the exams, thus increasing patients's satisfaction Harnett et al. (2010), and to avoid the cancellation of the surgery Ferschl et al. (2005). 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 were provided by a potential SurgiQ[1] client, is divided into two sub-problems Edward et al. (2008). In the first sub-problem, patients are assigned to a specific day based on a default list of exams, with the requirement that they be scheduled before their due date and that higher-priority patients receive scheduling preference. In the second sub-problem, the scheduler allocates a starting time for each exam required by the patients, considering operator availability and exam durations. Although our two-phase solution does not guarantee the best possible overall outcome, it was designed this way because: (i) it simplifies both the encoding and practical application, and (ii) it reflects how PAC schedules are often computed manually in hospitals.

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

[1] https://surgiq.com/.

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 Gebser et al. (2012) show that ASP is a suitable solving methodology also for the PAC scheduling and rescheduling problems.

The chapter is structured as follows. Sections 3.1 and 3.2 present an informal description of the problem and a precise, mathematical formulation, respectively. Section 3.3 shows our ASP encodings for both phases, whose experimental evaluation is presented in Section 3.4. Rescheduling ASP solutions are introduced and evaluated in Section 3.6 and final conclusion are presented in Section 3.7.

3.1 Problem Description

Pre-hospitalization is the phase in which the patient is admitted to the facility to undergo laboratory tests, radiological and cardiological examinations, visits for anesthesiological suitability, and any other assessments deemed necessary by medical specialists for defining the pre-operative diagnostic pathway (e.g., pulmonological or nephrological examinations). Managing the pre-operative pathway is crucial in healthcare organizations, as it requires the coordination of multiple facilities to ensure timely and accurate service delivery. However, the large number of services involved can create challenges in the patient's journey, particularly regarding service delivery time and the alignment between various providers (e.g., Cardiology, Pulmonology, Nephrology).

In many hospitals, including the one from which the specifications we use have been drawn, the date for pre-operative procedures is determined early, before the exact number of required exams and visits is known. Later, after an initial consultation with the patient, the specific number and type of exams and consultations are determined, allowing them to be scheduled more precisely, with each assigned a start time on the predetermined day. This process is often still handled manually, leading to challenges in appointment scheduling and provider coordination. The aim of automated scheduling is to complete the entire pathway in

one day, minimizing repeated patient visits and ensuring that all services are delivered in a timely manner.

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 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

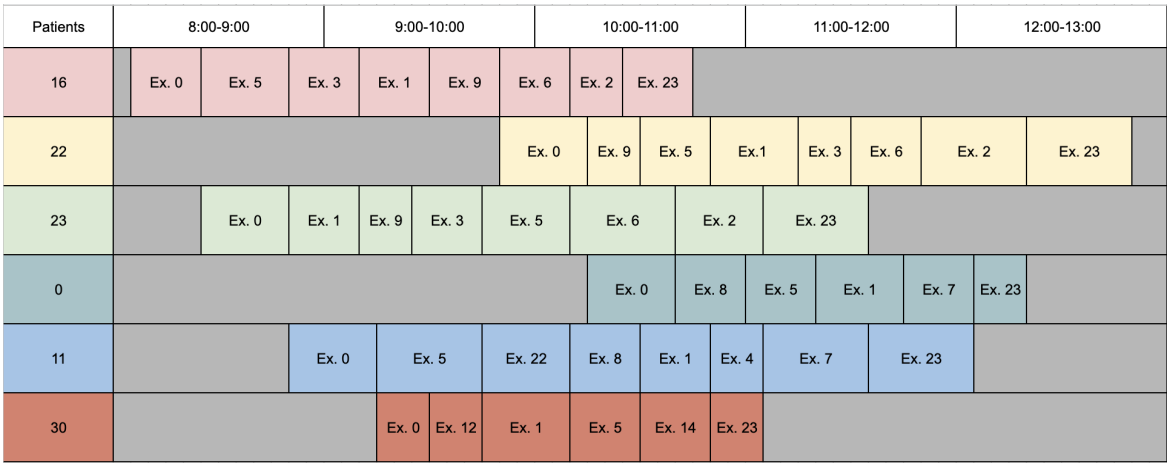

Figure 3.1 Schedule example: assignments of the starting time of the different exam locations of each patient in a single day.

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 3.1 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 3.1, 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.

3.2 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 (domination for the first sub-problem)**.** *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 (domination for the second sub-problem)**.** *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.*

3.3 ASP Encoding

In this section, starting from the specifications in Section 3.1 and the formalization of Section 3.2, we present the ASP encoding for the two sub-problems, based on the input language of CLINGO Gebser et al. (2016).

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 3.2 (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`.

```
1  {x(RID,PR,ST,ST+TOTDUR,DAY) : day(DAY),time(ST), DAY < DUEDATE, ST <=
       ts-TOTDUR} 1 :- reg(RID,PR,TARGET,TOTDUR,DUEDATE).
2  :- x(RID,_,_,_,DAY), exam(RID,AREAID,_), not examLoc(AREAID,_,_,DAY,_).
3  :- #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).
4  :- #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).
5  {operator(ID, AREAID, DAY) : operators(ID, AREAID, DAY)} == NOP :-
       examLoc(AREAID, NOP, _, DAY,_), x(RID, _, _, _,DAY), exam(RID, AREAID,
       _).
6  :- operator(ID,AREAID1,DAY), operator(ID,AREAID2,DAY), AREAID1 < AREAID2.
7  unassignedP1(N) :- M = #count {RID: x(RID,1,_,_,_)}, N = totRegsP1 - M.
8  unassignedP2(N) :- M = #count {RID: x(RID,2,_,_,_)}, N = totRegsP2 - M.
9  unassignedP3(N) :- M = #count {RID: x(RID,3,_,_,_)}, N = totRegsP3 - M.
10 unassignedP4(N) :- M = #count {RID: x(RID,4,_,_,_)}, N = totRegsP4 - M.
11 :~ unassignedP1(N). [N@8]
12 :~ unassignedP2(N). [N@7]
13 :~ unassignedP3(N). [N@6]
14 :~ unassignedP4(N). [N@5]
15 :~ x(RID,1,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@4,RID]
16 :~ x(RID,2,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@3,RID]
17 :~ x(RID,3,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@2,RID]
18 :~ x(RID,4,_,_,DAY), reg(RID,_,TARGET,_,_). [|DAY-TARGET|@1,RID]
```

Figure 3.2 ASP encoding of the first sub-problem

Encoding. The related encoding is shown in Figure 3.2, and is described in the following. To simplify the description, we denote as r_i the rule appearing at line i of Figure 3.2.

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 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 $unassignedP1$, $unassignedP2$, $unassignedP3$, and $unassignedP4$, 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

```
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]
```

Figure 3.3 ASP encoding of the second sub-problem

r_{15} and r_{18} minimize the difference between the assigned and target day of each registration, giving precedence to higher priorities.

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`.

Encoding. The encoding consists of the rules reported in Figure 3.3. 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. Rule 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).
```

Figure 3.4 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 3.3, 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. Similarly as it is done in CLP with the cumulative constraint when using the Edge Finder algorithm (Mercier and Van Hentenryck (2008); Nuijten (1994)), it is possible to reduce the number of possible assignments through the total time required by each exam. The difference between our approach and the Edge Finder one, is that we are pruning the search space thanks to an already known order, while the Edge Finder dynamically find a possible order due to the total time required by the resources. Thus, the guess rule can be improved by reducing the number of possible starting time slots of each exam with the following constraints:

- 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 3.3 with the rules reported in Figure 3.4, explained in the following. In rules r_6 and

```
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]
```

Figure 3.5 Optimized minimization rule

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 a higher (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 3.3, 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 3.3 with the rules shown in Figure 3.5. 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.

3.4 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, data have been randomly generated using parameters inspired by literature and real world data, then the results of the first sub-problem have been used as input for the second sub-problem. The last part 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 Gebser et al.

(2016) 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 .

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. Here, we present the results obtained by testing the first sub-problem. The primary optimization criterion in the PAC scheduling sub-problem is to assign as many patients as possible, starting with those of higher priority. Across all tested instances, our solution successfully assigns a day to 406 out of 437 patients with the highest priority. Furthermore, in 23 out of 30 instances, the solution assigns a day to all or all but one of the highest-priority patients.

Table 3.1 summarizes the results obtained for this first sub-problem, with a timeout set to 300 seconds per instance. A preliminary analysis on CLINGO was also conducted

Table 3.1 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%

using additional parameters, such as `-opt-strategy=usc`, which allows CLINGO to use a different optimization algorithm. Specifically, the table shows the average percentage of patients assigned from 10 instances with 40, 60, and 80 patients, prioritized according to their urgency. The test with 80 patients, in particular, illustrates how the solution performs under high demand and low resource availability, as the number of exam areas and operators is fixed. From the table, we can observe that the percentage of patients with priority 1 and 2 assigned is only slightly lower than in other benchmarks, while the percentage of lower-priority patients assigned decreases due to limited resources relative to demand.

Compared to the previous version of the encoding Caruso et al. (2021) used for the first sub-problem, this new version is able to assign more patients without resulting in unsatisfiable problems in the second sub-problem. The key change is that, while the old version overestimated the time required for each patient using a formula, the new version assigns a temporary starting time for the exams in the first sub-problem. This starting time is used to ensure that the first and last exams (which are mandatory for every patient) are not simultaneously required by more patients than allowed.

This modification in the encoding results in a higher number of patients being assigned in the first sub-problem. Specifically, while the previous version assigned 80% of the highest-priority patients, the new version assigns approximately 93%. Additionally, when testing the previous encoding with instances involving 80 patients, it was able to schedule only 45% of patients with priority 1 and 2, whereas the new version schedules 89% of these patients.

The second optimization criterion is to assign a day as close as possible to the patient's target day. While this criterion helps assign some patients near their target day, the quality of assignment decreases for others. This occurs for two reasons: first, a higher-priority optimization criterion exists, so the scheduler focuses on assigning as many patients as possible without considering their target days. Second, some patients have target days when their exam locations are unavailable, meaning that even in an optimal solution, their assigned day would not match their target day.

To corroborate the general explanation above, Figure 3.6 reports the result obtained by the scheduler with one of the instances with 60 patients as input. What can be seen from

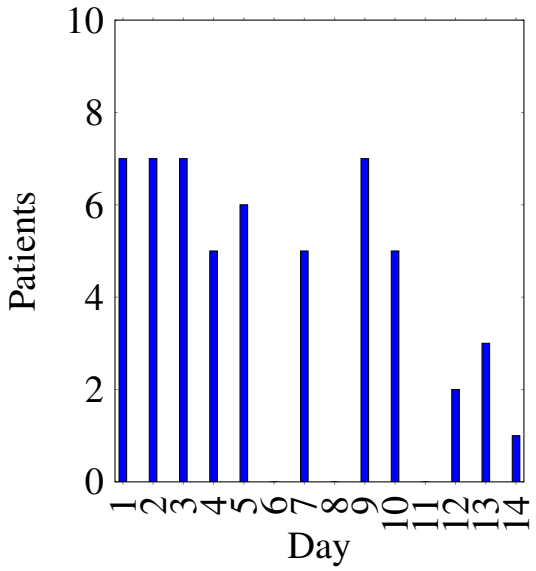

Figure 3.6 Number of patients assigned to each day by the scheduler with 60 patients as input.

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 does 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.

After having analyzed the results in the first sub-problem, we move on presenting the results in the second one, using the same experimental setup as the first. In the second sub-problem, the solution assigns the starting time for each patient's exams, using the input from the results of the first sub-problem. The goal is to minimize the time difference between the last exam's ending time and the first exam's starting time, thereby reducing the overall time patients spend in the hospital.

For this sub-problem, the scheduler consistently achieves an optimal solution across all tested instances. Additionally, in every instance, at most one patient experiences a one-time-slot wait between two exams, while all other patients have no waiting time between their exams. Specifically, in all but three instances, patients are assigned without any waiting time between exams.

The timings required by the scheduler in the second sub-problem are shown in Figure 3.7, 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.

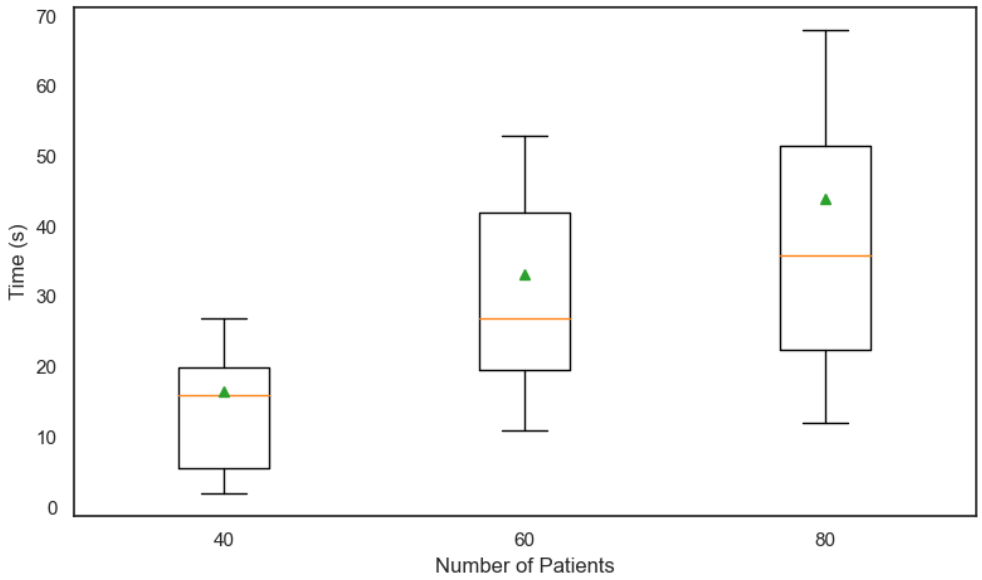

Figure 3.7 Results obtained by solving 10 instances generated from the results of the first sub-problem considering 40, 60, and 80 patients. The box starts from the first quartile and ends at the third quartile. The mean time is represented by the (green) triangle, while the (orange) line represents the median value.

Table 3.2 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%

To obtain these results we used the encoding defined in Section 3.3, included the domain specific optimizations. We are now interested in understanding the contribution that the two optimizations bring. Table 3.2 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 3.2, 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 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 3.8. 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.

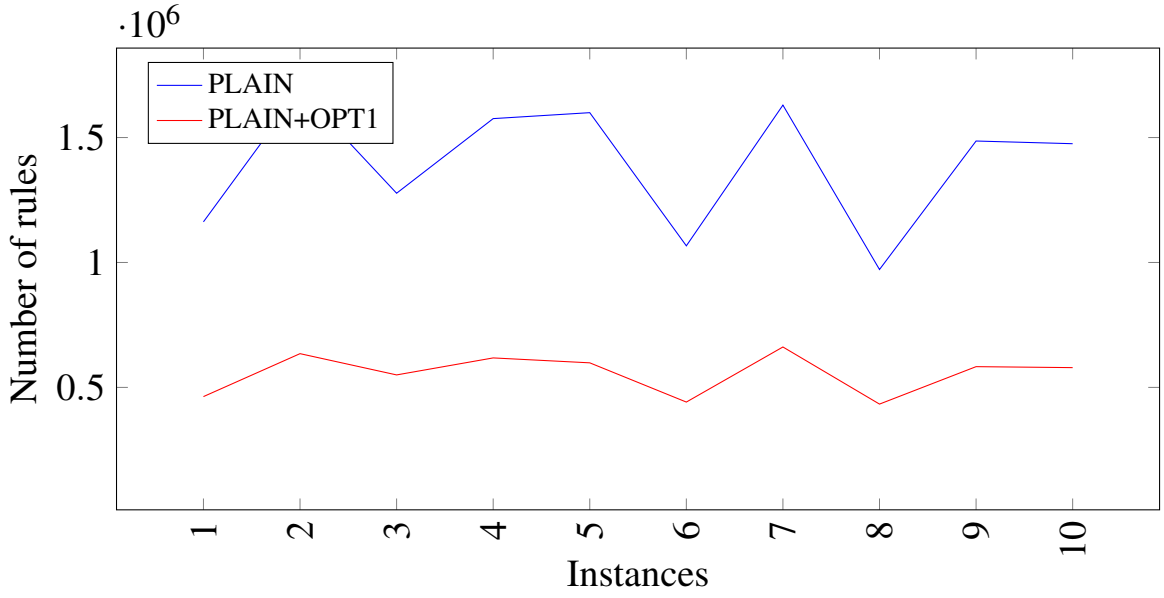

Figure 3.8 Comparison between the number of rules generated using the plain encoding and using the plain encoding with the OPT1 optimization.

Table 3.3 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

3.5 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 through automatic translations of ASP instances. Specifically, we used the ASP solver WASP Alviano et al. (2019a), with the option `-pre=wbo`, which converts ground ASP instances into pseudo-Boolean instances in the wbo format Olivier Roussel and Vasco Manquinho (2012). Then, we used the tool PYPBLIB Ansótegui et al. (2019) to encode wbo instances as MaxSAT instances. To ensure a fair comparison, since the other formalisms/solvers can not handle multi-levels optimizations, we also processed our ASP instances using WASP with the `-pre=lparse` option, which collapses all weak constraints levels into one single level using exponential weights. This allows for a comparison of the costs found by the different approaches. 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 Saikko et al. (2016), OPEN-WBO Martins et al. (2014), and RC2 Ignatiev et al. (2019), and the industrial tool for solving optimization problems GUROBI Gurobi Optimization, LLC (2021), 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 Morgado et al. (2014), 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 3.3, 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 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).

3.6 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; 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 to the original schedule. 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.

Given a scheduling and the information of why it is not possible to implement it, the PAC rescheduling problem consists of reassigning patients and operators to generate a new schedule that takes into account the new information and the constraints of the initial problem. To evaluate the rescheduling solution, we considered rescheduling after the second sub-problem: We did not consider the first sub-problem because, in case a need to change it emerged, since it is done well in advance it can be more appropriate to perform another schedule 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. 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.

To present the ASP encoding of the rescheduling, we start by presenting the Data model.

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

- 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 scheduling encoding, without the priority field.
- Instances of `operator(ID,AREAID,DAY)` represent the previous scheduling, i.e., the output of the scheduling encoding.
- 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 the scheduling problem, 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`.

Encoding. The ASP encoding of the PAC rescheduling problem we realized is made by the rules in Figure 3.9, where we denote with r_i rule appearing at the line i, plus rules r_5 and r_6 from Figure 3.2, where atom `operatorY` replaces atom `operator`, rules r_2-r_4 from Figure 3.3 (corresponding to the constraints of the second sub-problem), rules r_6 and r_7 from Figure 3.4 (for computing atoms `forbiddenAfter` and `forbiddenBefore` in the first

```
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]
```

Figure 3.9 ASP encoding of the rescheduling problem.

optimization), and rule r_9 in Figure 3.5 (corresponding to the second optimization), where atom y replaces atom x.

In Figure 3.9, 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} specify the optimization in a prioritized way. Weak constraints 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.

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 Gebser et al. (2016), 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 3.6. 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.

Table 3.4 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

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 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 3.1 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 3.10. The three patients were scheduled originally the day before Figure 3.1, 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.

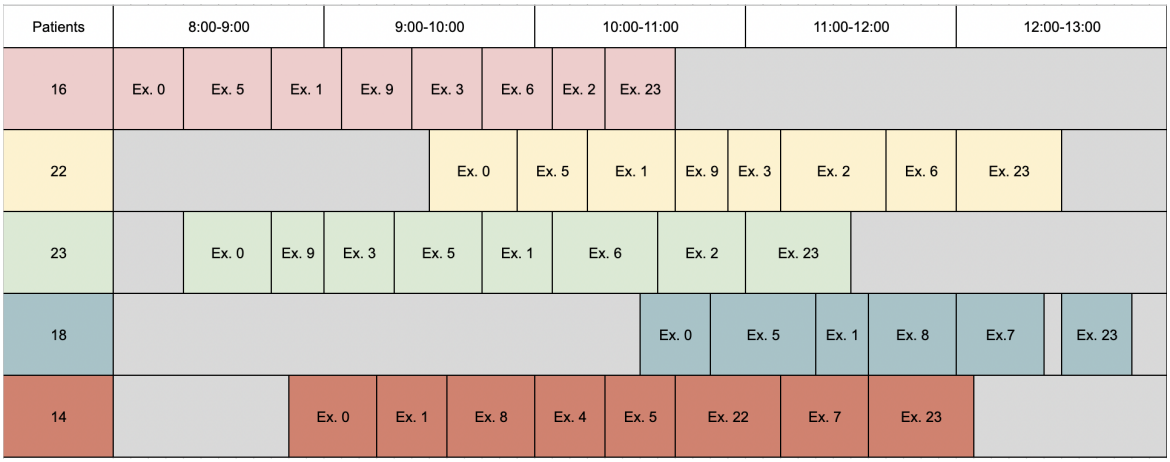

Figure 3.10 Rescheduling example related to registrations 16, 22, and 23 of Figure 3.1 rescheduled to a new day because of unavailability of exam location 3.

3.7 Conclusions

In this chapter, we have presented an analysis of a Digital Health scheduling problem, namely the PAC scheduling problem, modeled and solved with ASP. We started with a mathematical formulation of the problem, whose specifications come from a real scenario, and then presented our ASP solution. Results on synthetic data show that the solution can 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.

Chapter 4

Operating Room Scheduling Problem

Hospitals often face challenges like long waiting times, surgery cancellations, and resource overload. These issues can lower the level of patients' satisfaction and compromise the quality of care provided. In any modern hospital, Operating Rooms (ORs) are critical units. As indicated in Meskens et al. (2013), the OR management accounts for approximately 33% of the total hospital budget. This high cost is due to costs for staff (e.g., surgeons, anesthetists, nurses) and material costs. Nowadays, long surgical waiting lists are often present because of inefficient planning. Therefore, it is of paramount importance to improve the efficiency of OR management, in order to enhance the survival rate and satisfaction of patients, thereby improving the overall quality of the healthcare system.

To manage the ORs, a solution has to provide the date and the starting time of the surgeries required, considering the availability of ORs and beds, and of other resources requested. The Operating Room Scheduling (ORS) Abedini et al. (2016); Aringhieri et al. (2015); Hamid et al. (2019); Meskens et al. (2013) problem is the task of assigning patients to ORs by considering specialties, surgery durations, shift durations, and beds' availability, among others. Further, the solution should prioritize patients based on health urgency.

We start presenting a precise, mathematical formulation of the problem. Then, we introduce a new, ASP encoding for a basic version of the ORS problem previously employed that, e.g., does not take bed management into account. The improved encoding is more efficient not only for ASP, but also for pseudo-Boolean solvers run on benchmarks obtained by automatic translation from the ASP formulation. Further, we deal with the real case of ASL1 Liguria, an Italian health authority operating through three hospitals in the cities of Sanremo, Imperia, and Bordighera in the Liguria region, for computing weekly operating room surgery schedules. Starting from the improved encoding for the basic version of the problem, we present adaptations to deal with the data of the hospitals. We then define three

scenarios: in the first scenario, the goal is to test whether our solution is able to replicate the schedules that a hospital indeed followed, while the other two scenarios evaluate whether "better" schedules could have been determined. Further, we analyze the resulting encodings on the real data: results show that our solutions are able to both replicate and improve the original schedules, for weekly and monthly planning horizons, thus indicating that ASP produces satisfying results also when applied to such challenging, real data.

The chapter is structured as follows. Section 4.1 describes the ORS problem in an informal way, and its mathematical formulation is presented in Section 4.2. Section 4.3 compares two alternative encodings of a basic version of the ORS problem, while Section 4.5 presents the adaptation to real data. Then, the results of our experiments are presented on the aforementioned scenarios and include the impact of the optimized encoding on the performance of other logic-based solving approaches. Section 4.8 concludes the chapter.

4.1 Problem Description

This section outlines the ORS problem as described by ASL1 Liguria, Italy, which is a local health authority operating through three hospitals: Bordighera, Sanremo, and Imperia. The elements of a surgical waiting list are called *registrations*. Each registration links a particular surgical procedure, required by a patient, of a specific duration to a reservation number, a specialty, and a type of hospitalization.

The primary goal of the ORS problem is to assign as many registrations as possible from a waiting list to the appropriate ORs. It is possible that the surgery of some specialty could not be assigned to some ORs. In this way, we can ensure the most efficient use of OR time, which is a very valuable resource: OR costs are estimated in the range of tens of dollars per minute Smith et al. (2022), half of them being fixed costs due even when the OR is not treating patients Macario (2010). Given that patient overlap is not permitted in the same OR and to prevent overloading any OR, the first requirement is to ensure that the total duration of surgeries assigned to a particular OR does not exceed its available operational time. Considering the three hospitals of ASL1 Liguria, Bordighera has two ORs available from 07:30 to 13:30, while Imperia and Sanremo have five ORs available from 07:30 to 20:00.

Moreover, registrations can be linked to different types of hospitalizations. Specifically, patients may undergo *day surgery* or *ordinary surgery*, with the latter necessitating bed assignments for the patients before and/or after the surgery. Bed availability is a crucial resource for hospitals and often the actual bottleneck in surgical procedures scheduling.

Table 4.1 Beds available in Imperia.

OR	DAY 1	DAY 2	DAY 3	DAY 4	DAY 5
Gynecology	12	15	15	15	15
Cardiovascular	7	8	8	8	8
General Surgery	5	6	8	9	10
Urology	8	9	12	12	13

Table 4.2 Beds available in Sanremo.

OR	DAY 1	DAY 2	DAY 3	DAY 4	DAY 5
Gynecology	15	15	16	18	18
Orthopedics	7	9	8	12	13
ENT	5	5	5	5	5
General Surgery	6	7	9	10	11

As a result, an OR schedule must ensure that the number of patients requiring a bed for a particular specialty does not exceed the number of beds available each day. The allocation of beds for various specialties is determined by the Master Surgical Schedule (MSS), but the actual capacity may be reduced by beds that are already occupied by hospitalized patients or otherwise unavailable. The total number of beds available on each day of a week for different specialties in Imperia and Sanremo are presented in Tables 4.1 and 4.2, while Bordighera has no own beds available.

Other specific aspects of the problem involve patient priorities and ORs utilization. Not all registrations are created equal; they can be associated with various medical conditions and may have been on the waiting list for differing lengths of time. These two factors can be combined in a unique concept of *priority*. In our setting, we introduced four different priority categories, namely, p_1, p_2, p_3, and p_4. The first one gathers patients already planned by the hospital: it is required that each of these registrations is assigned to an OR. Then, the registrations of the other three categories are assigned depending on the ORs' capacities, prioritizing p_2 over p_3 and p_3 over p_4. Further, there are ORs that are used in a limited way, given they are reserved for emergencies and other purposes.

4.2 Mathematical Formulation

In this section, we present the mathematical formulation of the ORS problem according to the description presented in Section 4.1. We start by providing some preliminary concepts

essential for the definition of the ORS problem. Subsequently, we proceed by presenting the ORS problem in its basic context, along with the additional consideration of bed assignment.

Let N be the set of reservation numbers, OR represent the set of operating rooms, and D denote the set of days, where each day consists of two shifts spanning 5 hours, with the hours divided into time slots. Accordingly, let $SH = \{\text{shift}_1, \text{shift}_2\}$ be the set of shifts and $TS = \{t_1, \ldots, t_n\}$ represent the set of time slots. Finally, let SP denote the set of specialities, S represent the set of surgeries, and $P = \{p_1, p_2, p_3, p_4\}$ be the set collecting the four different levels of priorities. We point out that the unit of measurement in terms of time is represented by a time slot. Moreover, let:

- $open : OR \to \mathbb{N}^+$ be the function that associates to each operating room the number of time slots for which it is available;
- $end : SH \to TS$ be the function that associates a shift with its ending time;
- $\Delta : S \to \mathbb{N}^+$ be the function describing the *duration* of each surgery in terms of time slots and let $\Gamma = \{(s, \Delta(s)) \ : \ s \in S\}$ be the set that gathers all surgeries along with their corresponding durations;
- $c : S \to SP$ be the *clustering* function that assigns a surgery to a specific specialty;
- $\tau : SP \times OR \to \{0,1\}$ be the *room assignment* function such that $\tau(x,y) = 1$ if the specialty x is compatible with OR y, 0 otherwise.

Finally, let $COR = \tau^{-1}(1) = \{(x,y) \in SP \times OR \mid \tau(x,y) = 1\}$ be the set collecting all the specialties correctly associated to the ORs.

To link together a reservation number, a priority, and a surgery, we introduce the notion of *registration* via the following definition.

Definition 8 (Registration)**.** *A registration ρ is a function of the form*

$$\rho : N \times P \times S \to \{0,1\}$$

such that $\rho(n,p,s) = 1$ if there exists a reservation n with priority p for the surgery s, 0 otherwise.

The next step involves connecting a surgery with its corresponding specialty and identifying the ORs available for its execution. To this end, we define the notion of *assignment*.

Definition 9 (Assignment)**.** *An assignment α is a function of the form*

$$\alpha : \Gamma \times COR \to \{0,1\}$$

such that $\alpha((s,\Delta(s)),(x,y)) = 1$ if there exists a surgery s with duration $\Delta(s)$ such that s belongs to the specialty x, i.e. $c(s) = x$, and x is assigned to the OR y. Otherwise, $\alpha((s,\Delta(s)),(x,y)) = 0$.

In order to consider only suitable tuples, we consider the following sets of elements:

$$R := \rho^{-1}(1) = \{(n,p,s) \in N \times P \times S \mid \rho(n,p,s) = 1\}$$

and

$$A := \alpha^{-1}(1) = \{((s,\Delta(s)),(x,y)) \in \Gamma \times COR \mid \alpha((s,\Delta(s)),(x,y)) = 1\},$$

which collect all the suitable registrations and assignments, respectively. Finally, to join a registration with an assignment related to the surgery, we define the set

$$B := \{(n,p,s,y) \in N \times P \times S \times OR \mid (n,p,s) \in R, \exists x_1,x_2 \ : \ ((s,x_1),(x_2,y)) \in A\}.$$

Therefore, B is the set containing each registration n with priority p for the surgery s allocated to the OR y. Now we can define the notion of *scheduling*, which links together a reservation number, a priority, a surgery, an operating room, a time slot, a shift, and a day.

Definition 10 (Scheduling)**.** *A scheduling σ is a function of the form*

$$\sigma : B \times TS \times SH \times D \to \{0,1\}$$

such that $\sigma((n,p,s,y),u,w,d)) = 1$ if there exists a reservation n with priority p for the surgery s allocated to the OR y in the time slot u of shift w on day d.

The set $\Sigma = \sigma^{-1}(1) = \{\mathbf{t} = ((n,p,s,y),u,w,d) \in B \times TS \times SH \times D \mid \sigma(\mathbf{t}) = 1\}$ collects all the tuples eligible as scheduling. In the rest of the section, with a slight abuse of notation, we may write, for instance, $\sigma((\_,\_,\_,y),u,w,d) = 1$ to denote that there exists a tuple $(n,p,s) \in N \times P \times S$ such that σ assumes value 1.

ORS Problem In this section, we define the ORS problem within its basic framework, including day surgery and excluding the specific aspect of bed assignment.

Definition 11 (ORS). *The Operating Room Scheduling (ORS) problem is defined as the problem of finding a set ψ of tuples $\mathbf{t} = ((n,p,s,y),u,w,d) \in \Sigma$ that satisfies the following conditions:*

(c_1) $\forall (n,p,s) \in R$, *it holds that* $|\{(y,u,w,d) \mid ((n,p,s,y),u,w,d) \in \psi\}| \leq 1$;

(c_2) $\forall (n,p_1,s) \in R$, *it holds that* $|\{(y,u,w,d) \mid ((n,p_1,s,y),u,w,d) \in \psi\}| = 1$;

(c_3) $\forall \mathbf{t_1} = ((n,\_,\_,y),u,w,d), \mathbf{t_2} = ((n',\_,\_,y'),u,w,d) \in \psi$ *such that* $n \neq n'$, *it holds that* $y \neq y'$;

(c_4) $\forall y \in OR, \forall d \in D$, *it holds that* $\sum_{s:((\_,\_,s,y),\_,\_,d) \in \psi} \Delta(s) < open(y)$;

(c_5) $\forall \mathbf{t} = ((\_,\_,s,\_),u,w,\_) \in \psi$, *it holds that* $u + \Delta(s) < end(w)$;

(c_6) $\forall \mathbf{t_1} = ((\_,\_,s,y),u_1,w,d), \mathbf{t_2} = ((\_,\_,\_,y),u_2,w,d) \in \psi$ *such that* $u_1 < u_2$, *it holds that* $u_1 + \Delta(s) < u_2$.

The specified conditions are necessary to enforce the following constraints: (c_1) for each registration there exists at most one scheduling; (c_2) all registrations with priority p_1 must be scheduled; (c_3) double occupation of the same operating room is not allowed; (c_4) the total duration of surgeries assigned to a specific operating room must be within the operational time of the room; (c_5) the duration of surgeries cannot exceed the end of the assigned shift; (c_6) overlapping schedules for the same operating room are prohibited.

ORS Problem for Ordinary Surgery In the context of ordinary surgery, the ORS problem entails the assignment of a bed to the patient either before and/or after the surgery. Therefore, a solution to the problem must also consider the allocation of a bed for the patient and the corresponding availability of beds for each specialty and day.

To this aim, we need to define the following functions. Let

- $\beta : SP \times D \to \mathbb{N}_0$ be the function that returns the number of available beds for a specific specialty on a given day;
- $cd : R \to \mathbb{N}_0 \times \mathbb{N}_0$ be the required beds function, with $cd(n,p,s) = (\varepsilon_1, \varepsilon_2)$, where ε_1 (resp., ε_2) represents the number of days preceding (resp., following) the surgery for which the bed is required.

Definition 12 (ORS-OS)**.** *Let $rb : R \times D \rightarrow \{0,1\}$ be the function that maps a registration to a bed on a day such that, for each $((n,p,s,y),u,w,d) \in \psi$, it holds that $rb((n,p,s),d') = 1$, where $d' \in [d-\varepsilon_1, d+\varepsilon_2]$ and $cd((n,p,s)) = (\varepsilon_1,\varepsilon_2)$.*
The Operating Room Scheduling for Ordinary Surgery (ORS-OS) problem is defined as the problem of finding a set ψ of tuples $\mathbf{t} = ((n,p,s,y),u,w,d) \in \Sigma$ satisfying conditions (c_1), …, (c_6) such that:

(c_7) $\forall x \in SP, \forall d \in D$, it holds that $|RB| \leq \beta(x,d)$, where $RB = \{(n,p,s) \in R \mid c(s) = x, rb((n,p,s),d) = 1\}$.

Specifically, (c_7) ensures that every day the number of required beds for a given specialty must be less than or equal to the number of available beds for that specialty.

We define here the concept of: **Maximal Scheduling Solution** In Section 4.1, we have defined four different priority categories: p_1, p_2, p_3, and p_4. In accordance with constraint (c_2), all surgeries categorized under priority p_1 must be scheduled. Consequently, among the remaining priorities, we adopt a prioritization scheme favoring p_2 over p_3 and p_3 over p_4. To compare different solutions in terms of the number of schedulings with a specific priority, we provide the following definition.

Definition 13 (Dominating solution)**.** *Let $\Sigma_\psi^{p_i} := \{((n,p,s,y),u,w,d) \in \psi \mid p = p_i\}$ be the set collecting all the tuples corresponding to a specific priority p_i that constitute a solution for the problem under consideration. A solution ψ dominates a solution ψ' if $|\Sigma_{\psi'}^{p_2}| < |\Sigma_\psi^{p_2}|$, or if $|\Sigma_{\psi'}^{p_2}| = |\Sigma_\psi^{p_2}| \Rightarrow |\Sigma_{\psi'}^{p_3}| < |\Sigma_\psi^{p_3}|$, or if $|\Sigma_{\psi'}^{p_3}| = |\Sigma_\psi^{p_3}| \Rightarrow |\Sigma_{\psi'}^{p_4}| < |\Sigma_\psi^{p_4}|$.*

Consequently, we define the notion of a *maximal solution*.

Definition 14 (Maximal scheduling solution)**.** *A scheduling solution is maximal if it is not dominated by any other scheduling solution.*

The optimization variant of the ORS (resp., ORS-OS) problem, denoted as ORS-OPT (resp., ORS-OS-OPT), is to find a maximal scheduling solution.

4.3 ASP Encodings for the ORS-OPT problem

In this section, we present two encodings for the ORS-OPT problem, thus without bed management (with priority level limited to 3 as in previous formulations of the problem), one of which will be the base encoding to further employ in later sections. We start by presenting an encoding used for solving the ORS-OPT problem, and employed in the paper Scanu et al.

(2023), and then we introduce a new, improved encoding. The ASP encodings are based on the input language of CLINGO Gebser et al. (2016). Finally, through an analysis conducted on synthetic data, we show the benefits of the new version, which thus motivates the adoption of this encoding for the real-world ORS-OS-OPT problem at ASL1 Liguria, to be addressed in the next section.

In the following, we present the input and output data model as well as the first ASP encoding for the ORS-OPT problem.

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

- Constant `shift_duration` represents the duration of every shift in terms of time slots.
- Constants `totRegsP2` and `totRegsP3` represent the number of registrations of patients with priority p_2 and p_3, respectively.
- Instances of `registration(ID,P,SP,DUR)` represent the registration of the patient identified by an ID (`ID`) with priority level (`P`), the requested specialty (`SP`), and the duration of the surgery (`DUR`).
- Instances of `mss(OR,SP,SHIFT,DAY)` represent which specialty (`SP`) is assigned to an OR (`OR`) in a shift (`SHIFT`) on a day (`DAY`).
- Instances of `time(SHIFT,TS)` represent the available time slots (`TS`) in the shift (`SHIFT`).

The output is an assignment represented by atoms of the form

`x(ID,P,OR,DAY,SHIFT,TS)`,

where the intuitive meaning is that the patient identified by an ID (`ID`) having a priority (`P`) is assigned to the OR (`OR`) in the shift (`SHIFT`) at the time slot (`TS`) on the day (`DAY`).

Encoding. The related encoding is shown in Figure 4.1 and described next. To simplify the description, we denote the rule appearing at line i of Figure 4.1 by r_i.

Choice rule r_1 permits assigning an OR, a day, a shift, and a time slot to each registration. Rule r_2 ensures that each registration is assigned at most once. Rule r_3 ensures that, at every time slot, at most one registration is assigned to an OR on the same day and shift. Rule r_4 ensures that every registration with priority p_1 is assigned to some OR on a day, shift, and

```
1 {x(ID,P,OR,DAY,SHIFT,TS): TS+DUR <= shift_duration} :- registration(ID,P,SP,DUR),
      mss(OR,SP,SHIFT,DAY), time(SHIFT,TS).
2 :- #count{P,OR,DAY,SHIFT,TS: x(ID,P,OR,DAY,SHIFT,TS)} > 1, registration(ID,_,_,_).
3 :- #count{ID: x(ID,_,OR,DAY,SHIFT,TS), registration(ID,_,_,DUR), T >= TS, T < TS+DUR} > 1,
      mss(OR,_,SHIFT,DAY), time(SHIFT,T).
4 :- #count{ID: x(ID,1,_,_,_,_)} < totRegsP1.
5 :~ M = #count{ID: x(ID,2,_,_,_,_)}, N = totRegsP2 - M. [N@3]
6 :~ M = #count{ID: x(ID,3,_,_,_,_)}, N = totRegsP3 - M. [N@2]
```

Figure 4.1 ASP encoding for the ORS-OPT problem.

time slot. The weak constraints r_5 and r_6 minimize the number of unassigned registrations with priority p_2 or p_3, respectively.

After having presented an encoding for the ORS-OPT problem, we introduce here an optimized version of such problem. Unlike the previous encoding, this new approach doesn't attempt to assign a specific time slot along with the day, shift, and OR. Since the sequence of patients doesn't affect the utilization of other resources, such as beds considered in the next section, we can just assign an OR, a day, and a shift first. After that, we can determine the starting times for surgeries based on the order of the patients sharing the same OR on the same day and shift.

Data Model. The input data is the same as presented previously, without the predicate `time` and the constants `totRegsP2` and `totRegsP3`, while the output is an assignment represented by atoms of the form

`x(ID,P,DUR,OR,SHIFT,DAY)`,

where the intuitive meaning is that the patient identified by an ID (`ID`) having a priority (`P`) with a surgery duration (`DUR`) is assigned to the OR (`OR`) in the shift (`SHIFT`) on the day (`DAY`), together with atoms of the form

`end(ID,Q)`,

meaning that the greatest integer for `Q` is the ending time for the surgery of the patient.

Encoding. The related encoding is shown in Figure 4.2, where we again denote the rule appearing at line i of Figure 4.2 by r_i.

Rule r_1 defines an auxiliary predicate that represents the possible ORs, days, and shifts to which a registration can be assigned. Choice rule r_2 permits assigning at most one OR, a day, and a shift to each registration. Rule r_3 ensures that, for every OR, day, and shift, the sum of the durations of the assigned surgeries does not exceed the constant `shift_duration`. Rule

```
1  feasible(ID,P,DUR,OR,SHIFT,DAY) :- registration(ID,P,SP,DUR), mss(OR,SP,SHIFT,DAY),
       DUR <= shift_duration.
2  {x(ID,P,DUR,OR,SHIFT,DAY) : feasible(ID,P,DUR,OR,SHIFT,DAY)} 1 :- feasible(ID,_,_,_,_,_).
3  :- #sum{DUR,ID : x(ID,_,DUR,OR,SHIFT,DAY)} > shift_duration, mss(OR,SHIFT,_,DAY).
4  overlap(ID1,DUR1,ID2,DUR2) :- x(ID1,_,DUR1,OR,SHIFT,DAY), x(ID2,_,DUR2,OR,SHIFT,DAY),   ID1 < ID2.
5  {ordering(ID1,ID2,DUR2)} :- overlap(ID1,DUR1,ID2,DUR2).
6   ordering(ID2,ID1,DUR1)  :- overlap(ID1,DUR1,ID2,DUR2), not ordering(ID1,ID2,DUR2).
7  end(ID,0..DUR-1) :- feasible(ID,_,DUR,_,_,_).
8  end(ID,Q) :- ordering(ID1,ID,DUR), end(ID1,Q1), Q = Q1+DUR, Q <= shift_duration.
9  :- end(ID,shift_duration).
10 :- registration(ID,1,_,_), not x(ID,1,_,_,_,_).
11 :~ registration(ID,P,_,_), not x(ID,P,_,_,_,_), 1 < P. [1@-P,ID]
```

Figure 4.2 Improved ASP encoding for the ORS-OPT problem.

r_4 defines a predicate indicating registrations sharing the same OR, shift, and day. Rules r_5 and r_6 make use of this predicate to choose an ordering between registrations assigned to an OR on the same day and shift. Rules r_7 and r_8 propagate the ending times for surgeries along the chosen ordering, leading to an interval $[0, \mathtt{Q}]$ to represent the ending time `Q` for a surgery. Rule r_9 ensures that the ordering of operations is non-circular. Moreover, r_{10} ensures that every registration with priority p_1 is assigned, while the weak constraint r_{11} minimizes the number of unassigned registrations with priority p_2 or p_3, respectively.

4.4 Preliminary Comparative Analysis and Benchmarks

Here, we present a preliminary comparative analysis of the two encodings given in Figure 4.1 and Figure 4.2. In the first paragraph, we describe the benchmarks used for the comparison, while the second one discusses the results. The comparison has been carried out on an Apple M1 CPU @ 3.22 GHz machine with 8 GB of physical RAM and a time limit of 60 seconds per run. The ASP system used was CLINGO 5.6.2, configured with the parameters *--restart-on-model* and *--parallel-mode=6*: these parameters have been found to be effective in a preliminary analysis we performed with several options.

To compare the two encodings, we used the same synthetic data as previously taken to test the ORS-OPT problem with surgical teams Dodaro et al. (2020), where we disregard such teams here. The data emulate the operations of a typical medium-sized Italian hospital: the setting is composed of 5 different specialties, 10 ORs, and 70 registrations per day by patients with priority p_1, p_2, and p_3. We considered 4 distinct scenarios based on the scheduling duration: 1 day, 2 days, 3 days, and 5 days. Each day consists of two shifts, each spanning 5 hours, with the hours divided into time slots. We tested the encodings based on different values for the length of time slots: 10 minutes, 20 minutes, 30 minutes, and 60 minutes. For

Table 4.3 Comparison of the results obtained by the Base and the Optimized encoding for the ORS-OPT problem on the 5 days scenario. The cells provide the number of patients of priority p_2 (P2, top) and p_3 (P3, bottom) that could not be assigned.

Instance		10 Minutes		20 Minutes		30 Minutes		60 Minutes	
		Base	Optimized	Base	Optimized	Base	Optimized	Base	Optimized
1	P2	13	11	12	11	1	1	1	1
	P3	128	79	136	98	139	98	81	81
2	P2	13	6	5	5	7	6	1	1
	P3	134	114	94	83	85	90	74	77
3	P2	16	13	5	5	11	10	6	6
	P3	140	140	90	81	134	79	74	77
4	P2	1	0	5	5	6	5	9	9
	P3	136	85	142	95	92	84	84	75
5	P2	12	9	4	4	6	5	4	4
	P3	129	129	94	94	130	130	74	75
6	P2	11	9	9	7	5	5	9	9
	P3	102	85	98	81	148	91	135	86
7	P2	2	2	7	6	6	6	14	14
	P3	144	108	141	88	133	86	136	77
8	P2	7	5	3	2	10	10	10	10
	P3	130	130	104	91	139	79	140	72
9	P2	12	9	4	4	9	9	2	2
	P3	130	130	139	88	137	80	80	82
10	P2	13	10	11	10	7	6	6	6
	P3	134	86	126	91	142	142	80	82
Avg. Values	P2	10.0	7.4	6.5	5.9	6.8	6.2	6.2	6.2
	P3	130.7	108.6	116.4	89.0	127.9	95.9	95.8	78.4

every scenario, characterized by a particular number of days and length of the time slots, we randomly generated 10 instances.

Table 4.3 summarizes the results obtained by the two encodings considering 5 days with different lengths of time slots. Both encodings are unable to reach (proven) optimal solutions on the scenario with 5 days within the time limit. As the table shows, the optimized encoding leaves the same number or fewer patients with the higher priority p_2 unassigned on each tested instance. Moreover, the optimized encoding is able to assign not only more patients with priority p_2, thus providing a better solution, but it almost always assigns more patients with priority p_3 too. These results are also confirmed on the smaller scenarios covering 1, 2, or 3 days. Finally, we note that, while the base encoding does not reach any (proven) optimal solution for the instances, the optimized encoding provides optimal solutions on all instances of the 1 day scenario.

Thus, based on the presented analysis, we decided to use the optimized encoding in Figure 4.2 as the starting point for an extension to the ORS-OS-OPT problem, as dealt with in the real case of ASL1 Liguria.

4.5 ASP Encoding for the ORS-OS-OPT Problem

Starting from the improved ORS-OPT encoding in the previous section, here we present our compact and efficient ASP solutions for the ORS-OS-OPT problem, introduced in Section 4.2, which includes bed management (see Section 4.2) and, moreover, considers real hospital scheduling data. The section is divided into two parts for presenting solutions to the scenarios we deal with: replicate the hospital's schedule and improve such a schedule, respectively. However, the real data do not include information about surgeries' starting times and shifts, so we disregard time slots and shifts in the following. The replication encoding can be understood as a tool for checking whether the available capacities of ORs and beds are respected. This forms the base for two ORS-OS-OPT encodings, aiming to assign a maximal amount of new registrations in addition to patients already scheduled by the hospital. These two encodings vary in whether the original assignments can be moved to another OR and day, or not, and a constraint on the limited use of a specific OR, which has been identified in the original hospital schedule, is considered with one of the ORS-OS-OPT encodings.

Here, we present the input and output data model, and the ASP encoding replicating the original schedule of the hospital.

Data Model. The input data is specified by means of the predicates `registration` and `mss`, as presented before, slightly adjusted and presented here in the modified version, and the predicates `beds` and `givenSchedule` introduced here for the first time.

- Instances of `registration(ID,P,SP,DUR,D1,D2)` represent the registration of the patient identified by an ID (`ID`) with priority level (`P`), the requested specialty (`SP`), the duration of the surgery (`DUR`), and the number of days in which a bed is required before (`D1`) and after (`D2`) the surgery.
- Instances of `mss(OR,SP,DAY)` represent which specialty (`SP`) is assigned to an OR (`OR`) on a day (`DAY`).
- Instances of `beds(N,SP,DAY)` represent the number (`N`) of available beds for a specialty (`SP`) on the day (`DAY`).

```
4 stay(ID,SP,DD) :- x(ID,P,DUR,OR,DAY), registration(ID,P,SP,DUR,D1,D2), beds(N,SP,DD),
      D1 + D2 > 0, DD = D-D1..D+D2.
5 :- #count{ID: stay(ID,SP,D)} > N, beds(N,SP,D).
6 :- givenSchedule(ID,DAY,OR), not x(ID,_,_,OR,DAY).
```

Figure 4.3 ASP encoding for replicating the original hospital schedule.

- Instances of `givenSchedule(ID,DAY,OR)` represent the original schedule of the hospital, characterized by the patient identified by an ID (`ID`) scheduled on a day (`DAY`) in an OR (`OR`).

The output is an assignment represented by atoms of the form

$$\texttt{x(ID,P,DUR,OR,DAY)},$$

where the intuitive meaning is that the patient identified by an ID (`ID`) having a priority (`P`) with a surgery duration (`DUR`) is assigned to the OR (`OR`) on the day (`DAY`).

Encoding. The related encoding includes rules similar to r_1, r_2, and r_3 from Figure 4.2, with the predicates `registration`, `mss`, and `x` adjusted as described above. Regarding the additional rules shown in Figure 4.3, the predicate defined by r_4 indicates the days before and after a patient's surgery on which a bed is required for the respective specialty. Rule r_5 ensures that the number of patients requiring a bed for a particular specialty does not exceed the number of available beds per day. Moreover, r_6 ensures that the newly generated schedule coincides with the original hospital schedule.

Next, we explore ASP encodings designed to enhance the original hospital schedule. The goal is to manage both the patients already assigned by the hospital and new registrations. Our first encoding, called OPT1, focuses on assigning the original scheduled patients without needing to preserve the original OR and day. In contrast, the second encoding, known as OPT2, retains the original patients as scheduled by the hospital. Additionally, OPT2 includes a constraint based on real data to better reflect the actual circumstances.

Data Model. The input data for the encodings OPT1 and OPT2 is the same as for the replication encoding, yet the predicate `givenSchedule` is not used by OPT1.

Encodings. The ASP encoding OPT1 modifies the replication encoding in Figure 4.3 by dropping r_6, so that the original assignments of patients can be changed, while r_8 and r_9 in Figure 4.4 are added. Similar to the previous encoding in Figure 4.2, the new rules ensure

```
8 :- registration(ID,1,_,_,_,_), not x(ID,1,_,_,_).
9 :~ registration(ID,P,_,_,_,_), not x(ID,P,_,_,_), 1 < P. [1@-P,ID]
```

Figure 4.4 ASP rules for dealing with priorities.

```
10 :- #count{ID: x(ID,_,_,"OR A",_)} > 1.
```

Figure 4.5 ASP rule that encodes a constraint from the real data.

that patients with priority p_1 get assigned, while the number of assignments for patients with priority p_2, p_3, or p_4 is maximized in decreasing order of significance. The requirement that originally scheduled patients get assigned can thus be expressed by categorizing them as priority p_1.

The second encoding OPT2 keeps the original hospital schedule unchanged by including all rules from Figure 4.3 and Figure 4.4, i.e., r_6 is not dropped. Moreover, the rule r_{10} in Figure 4.5 is added to ensure that the specific OR "OR A" is assigned to at most one patient, as "OR A" was reserved for emergencies and used in this limited way in the original data.

4.6 Experimental Results

In this section, we report the results of an empirical analysis conducted using the defined ASP encodings, on the scenarios previously defined. For all the settings we used the original data.

The experimental setting is the same as Section 4.4, including the time limit set to 60 seconds. Encodings and benchmarks employed in this section can be found at: https://github.com/MarcoMochi/JLC2023ASL1.

We can now present the data we used to perform the analysis and the considered scenarios.

Data Description. To test our solution for scheduling surgeries, we utilized data from the hospitals of ASL1 in the Liguria region, Italy. The hospitals in ASL1 serve a population of around 213,000 people. For our analysis, we used data from a weekly schedule of surgeries across the three hospitals, as well as data from other weeks, including a list of available beds and ORs for all hospitals.

We collected and prepared the data for testing by working with five different xls files, each file represents a different type of data, in particular:

- The operating list of the considered week of surgeries, from 04/03/2019 to 10/03/2019, which provided information on the required surgery, the operating room, and the specialty originally scheduled.
- The historical list of surgeries scheduled in 2019, which includes information on the required surgery, the starting and ending time of the surgery, and the date of the surgery.
- The list of ORs in each hospital and their opening hours.
- The list of patients hospitalized the week before the considered week of the scheduling, along with their admission and discharge times.
- The list of beds in each specialty at each hospital.

Tested scenarios. Having presented the data, we can present the different scenarios we used to test the encodings. In the first scenario, the solution has to provide a schedule for the patients of the considered week and the number of available resources, beds and ORs, replicating the original schedule. This scenario was designed to confirm the consistency of the schedule produced by our encoding with the schedule of the patients as done by the hospital. For the two remaining scenarios, OPT1 and OPT2, we wanted to test our solution by scheduling the patients scheduled by the hospital plus other patients. This enabled us to assess how the ASP solutions could have improved patient's assignment and optimized resource allocation. In particular, in OPT1, we considered both the original patients assigned by the hospital and additional new patients, without imposing constraints requiring the ASP solution to replicate the hospital's schedule, while, in OPT2, the solution had to schedule the original patients in the same way done by the hospital. In the hospital of Bordighera, we had to make a slight change between OPT1 and OPT2. Indeed, the hospital scheduled just 1 patient in one OR, without using it for other patients. Thus, we decided to discard this OR in OPT1, while we maintained it just for that patient in OPT2 (this is linked to rule r_{10} in Figure 4.5). Both for OPT1 and OPT2, a selection of new patients was needed. To select these additional patients and distinguish them from the original ones, we made use of the concept of priority. In particular, originally scheduled patients were assigned priority 1 (we remind that patients with priority 1 are forced to be assigned, freely in OPT1 while following the original schedule in OPT2). Then, we randomly selected a number of patients assigned by the hospital in the following weeks, assigning priority 2 to patients assigned in the next week, priority 3 to patients assigned after 2 weeks, and priority 4 to patients assigned at least 3 weeks later. The number of additional patients the solution will try to schedule is linked to the

Table 4.4 Percentage usage of ORs in Bordighera. A "-" means that the OR is not available on that day.

OR	DAY 1	DAY 2	DAY 3	DAY 4	DAY 5	AVERAGE
OR A	-	-	-	8.2%	-	8.2
OR B	59.1%	69.7%	70.0%	74.2%	80.0%	70.6%

Table 4.5 Percentage usage of ORs in Imperia. A "-" means that the OR is not available on that day.

OR	DAY 1	DAY 2	DAY 3	DAY 4	DAY 5	AVERAGE
OR A	57.9 %	85.9 %	50.4 %	86.3 %	39.6 %	64.0 %
OR B	44.5 %	48.0 %	45.0 %	41.6 %	60.1 %	47.8 %
OR C	24.9 %	24.7 %	32.3 %	38.0 %	32.0 %	30.4 %
OR E	25.3 %	34.0 %	36.3 %	25.2 %	28.4 %	29.8 %
OR Ophthalmology	38.5 %	38.4 %	-	-	-	38.5 %

original number of scheduled patients. Each scenario was tested with 10 different instances composed of different samples of additional patients. In particular, for all the hospitals, the solution tried to schedule a number of patients equivalent to the 250% of the original one. We decided to increase the number of patients to 250% because, after a preliminary analysis, we found that even when this value was increased, the number of patients assigned did not increase. Conversely, using a lower value resulted in solutions where all the patients were assigned. OPT1 and OPT2 enabled us to assess the potential impact of our solution in terms of reducing patient waiting lists and optimizing resource allocation.

Results for Scenario 1 Scenario 1 consists of recreating the same schedule done by the hospital. The solution is obtained in less than 0.5 seconds for all the hospitals, indicating the correctness of our solution. In Tables 4.4, 4.5, and 4.6 it is possible to see the original percentage usage of the ORs obtained by the three hospitals of Bordighera, Imperia, and Sanremo, respectively. These results represent the benchmarks to compare to when evaluating OPT1 and OPT2.

Results for OPT1 Tables 4.7, 4.8, and 4.9 show the results obtained in this setting in terms of the number of patients assigned on all the ten instances in the hospitals of Bordighera, Imperia, and Sanremo, respectively. As can be seen from the tables, the solution is able to assign a consistent number of additional patients of priority levels P2, P3, and P4 for all the hospitals. Indeed, patients P1 are the patients originally scheduled, while all the other patients represent the additional ones. Moreover, even if not all the patients with priority 2

Table 4.6 Percentage usage of ORs in Sanremo. A "-" means that the OR is not available on that day.

OR	DAY 1	DAY 2	DAY 3	DAY 4	DAY 5	AVERAGE
OR 1	59.1 %	51.9 %	25.7 %	41.9 %	74.0 %	50.5 %
OR 2	-	21.6 %	63.2 %	-	-	42.4 %
OR 3	24.7 %	34.0 %	-	-	24.8 %	27.8 %
OR 4	-	35.3 %	-	86.3 %	-	60.8 %
OR C	12.9 %	-	-	-	14.4 %	13.7 %

Table 4.7 Number of assigned patients in Bordighera grouped by their priority level.

P1	P2	P3	P4
28/28	14/29	0/28	0/13
28/28	15/29	0/28	0/13
28/28	13/29	0/28	0/13
28/28	14/29	0/28	0/13
28/28	14/29	1/28	0/13
28/28	14/29	0/28	0/13
28/28	13/29	2/28	1/13
28/28	14/29	1/28	0/13
28/28	14/29	0/28	0/13
28/28	13/29	0/28	0/13

are assigned, some patients with lower priorities are. This is due to the fact that a bottleneck of the hospitals taken into account is beds availability. Thus, once all the beds are occupied, the solution is able to schedule some patients with lower priorities that do not require a bed before and/or after the surgery. To corroborate the explanation above, the percentages of beds usage in the different specialties, in Sanremo, are presented in Table 4.10. From the table it can be seen that the beds are used almost at full capacity throughout the week; thus, for the solution is not possible to assign additional patients requiring a bed. In the hospital of Imperia, beyond the beds, even the ORs are used almost at full capacity with the additional patients. In particular, in Figure 4.6, it can be seen a comparison between the obtained percentage usage of the ORs with the ASP solution in OPT1 and the usage obtained by the ASL1. Without following the previous assignments of ASL1, the ASP solution is able to schedule three times the number of patients originally scheduled.

Results for OPT2 After having tested, with the scenario OPT1, a solution assigning all the patients with no constraints, we now check the performance in the more constrained scenario OPT2, where the original schedule and a constraint entailed by real data are considered. The results obtained in this scenario are summarized for each hospital in Figure 4.7. From the

Table 4.8 Number of assigned patients in Imperia grouped by their priority level.

P1	P2	P3	P4
143/143	112/120	109/130	63/108
143/143	112/120	109/130	59/108
143/143	112/120	109/130	59/108
143/143	112/120	109/130	55/108
143/143	112/120	109/130	61/108
143/143	112/120	109/130	65/108
143/143	112/120	109/130	60/108
143/143	112/120	109/130	49/108
143/143	112/120	109/130	63/108
143/143	112/120	109/130	62/108

Table 4.9 Number of assigned patients in Sanremo grouped by their priority level.

P1	P2	P3	P4
43/43	12/28	7/26	5/54
43/43	12/28	7/26	6/54
43/43	12/28	7/26	3/54
43/43	12/28	7/26	4/54
43/43	12/28	7/26	5/54
43/43	12/28	7/26	5/54
43/43	12/28	7/26	9/54
43/43	12/28	7/26	7/54
43/43	12/28	7/26	5/54
43/43	12/28	7/26	1/54

figure, it can be noted that even in this more constrained scenario we are able to assign a consistent number of patients of priority levels P2, P3, and P4 for all the hospitals. Compared to OPT1, results are overall similar, and mixed if we consider individual hospitals. Thus, our solution is able to adapt to build an efficient schedule even starting from a sub-optimal one.

In particular, in Bordighera, the two scenarios assign the same number of patients. However, we remind that in this hospital one patient is assigned to an OR by the additional constraint for OPT2, which is discarded in OPT1. Therefore, even using fewer resources, the solution of OPT2 is still able to match the performance of OPT1.

In Imperia, the total number of patients assigned by OPT2 is actually higher than the one assigned in OPT1 but, as can be seen in Table 4.11, the schedule provided by OPT1 is of higher quality, since it is able to assign more patients with higher priority. Finally, in Sanremo, the schedule done by OPT1 outperforms that of OPT2 by assigning more patients while not decreasing the quality of the solution.

Thus, having analyzed the results in all three hospitals, we can conclude that starting from a sub-optimal solution has, as expected, an impact on the quality of the solution, however,

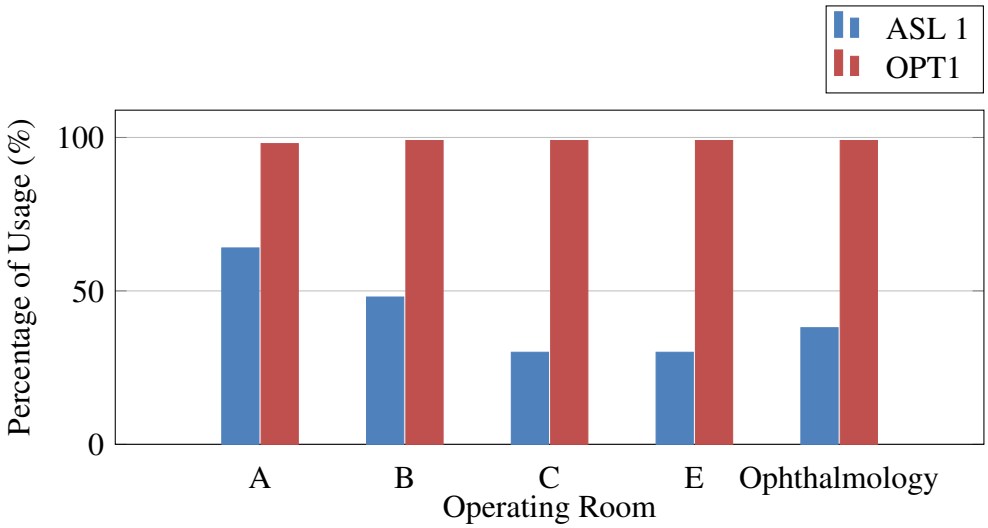

Figure 4.6 Comparison of the ORs usage in Imperia between the ASP solution of OPT1 and the ASL1 schedule.

Table 4.10 Percentage of usage of beds in Sanremo.

GYNECOLOGY	ORTHOPEDICS	ENT	GENERAL SURGERY
90%	100%	95%	100%
90%	100%	95%	99%
90%	100%	94%	97%
90%	100%	82%	99%
90%	100%	97%	99%
90%	100%	94%	97%
82%	100%	97%	97%
90%	100%	90%	97%
90%	100%	100%	97%
82%	100%	85%	100%

it is still to be noted that the solutions obtained in this scenario are, in terms of number of patients assigned, at least similar or equal to the solutions of scenario OPT1.

Results on Monthly Data After presenting the results from our one-week surgery data and evaluating the effectiveness of our solution, we decided to conduct further testing using a full month of data. This longer test was essential to determine whether our solution could outperform the hospital's schedule over an extended planning horizon. Our concern was that, given our weekly solution schedules a number of patients that lead to having the resources completely used, it could be the case that these results are obtained at the price of sacrificing the performance in the following weeks (but we will show that this is not the case). We decided to start the analysis from the same week used in Section 4.6, since it is the only

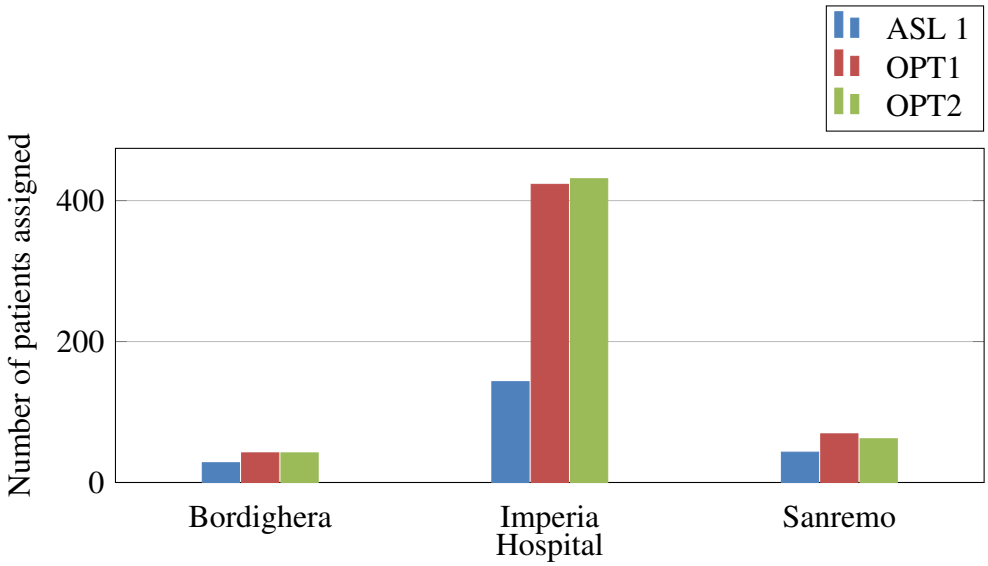

Figure 4.7 Comparison on the number of patients scheduled by ASL1, and by OPT1 and OPT2 solutions.

Table 4.11 Mean percentage of assigned patients with different priorities in Imperia for OPT1 and OPT2.

SETTING	P1	P2	P3	P4
OPT1	100%	93%	83%	41%
OPT2	100%	90%	83%	63%

week with the information regarding the beds' occupation. Moreover, we collected the new registrations from the following weeks as new input and derived the availability of the beds from the schedule of the week before the considered one. Following this schema, we started with 10 different instances having the original values for the usage of the beds. Subsequently, we used the result of each instance to derive the beds' availability, and the patients still not assigned in the subsequent week, and used them as new inputs for the instances of the next week. As a result, for every week, we have 10 different instances whose resource availability depends on the instances of the previous week.

Figure 4.8 shows the total cumulative number of patients assigned in each week by our solution in all the 10 instances, compared to the result of the hospital. From the results, we can state not only that the solution is able to schedule more patients in a week, compared to the hospital, but that our solution is able to assign more patients week after week. Moreover, by inspecting the single instances, in all the 10 instances (dealing with different registrations, thus different requirements in terms of surgery duration and beds requirement) the quality of

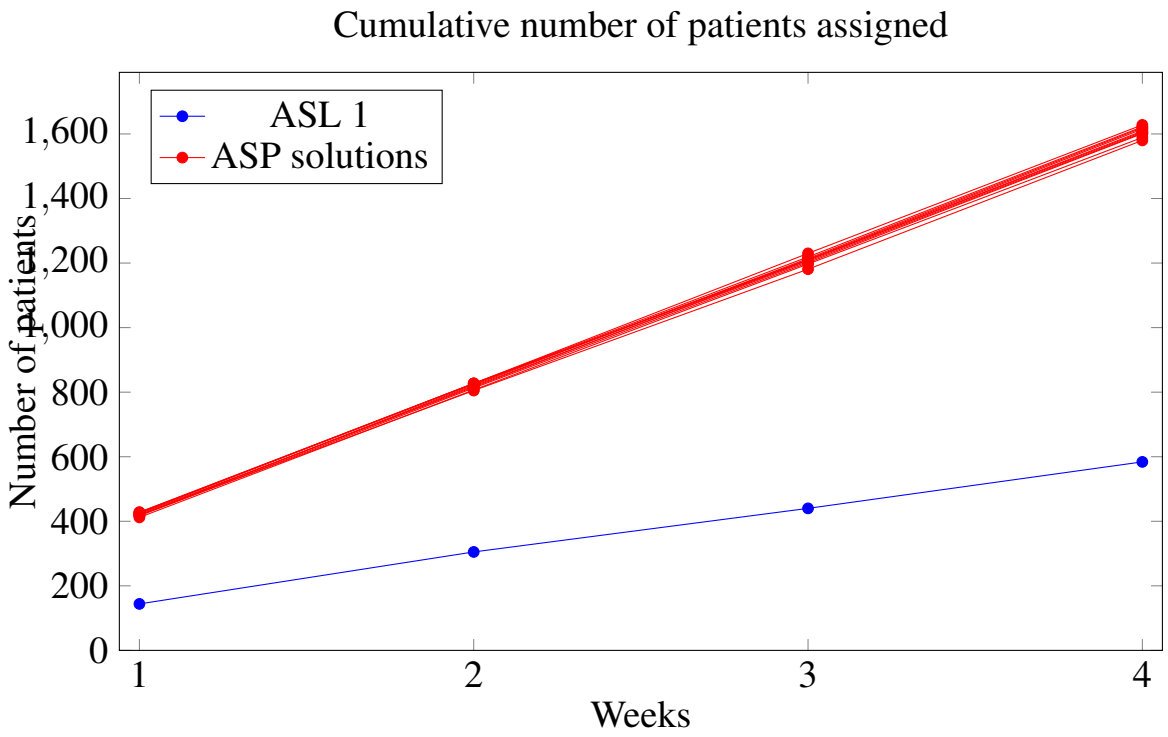

Figure 4.8 Cumulative number of patients assigned by the hospital (blue line) in 4 weeks compared to the number of patients assigned by the ASP-based solution (red line) in all the 10 instances of the Imperia hospital.

the solution did not decrease, meaning that our solution is not assigning too many patients in a week at the cost of the quality of the schedules in the subsequent weeks.

4.7 Comparison to Alternative Logic-based Formalisms

In the following, we present an empirical comparison of the original solution presented in Scanu et al. (2023) and our new optimized ASP-based solution to the scenario OPT1 presented in Section 4.6 on an alternative logic-based formalism, obtained by applying automatic translations of ASP instances. With this analysis, we want to ensure that the new solution leads to better results than the original solution even when using different logic-based formalisms. In more detail, we used the ASP solver WASP Alviano et al. (2019a), with the option `-pre=wbo`, which converts ground ASP instances into pseudo-Boolean instances in the wbo format Olivier Roussel and Vasco Manquinho (2012).

Table 4.12 Comparison of the original and the optimized ASP solution using CLINGO with the option `restart-on-model` (CLINGO-ROM in the table) and the alternative logic-based solution GUROBI on wbo instances. The value in each cell represents the time in seconds required to reach an optimal solution if the solver was able to find it in less than 60 seconds, or a percentage value representing the gap to the optimal solution.

Instance	Original Solution		Optimized Solution	
	CLINGO-ROM	GUROBI	CLINGO-ROM	GUROBI
1	0.01%	0.0001%	0.001%	2s
2	0.01%	0.0002%	0.001%	3s
3	0.01%	36s	0.001%	2s
4	0.01%	13s	0.001%	2s
5	0.01%	18s	0.001%	5s
6	0.01%	13s	0.001%	2s
7	0.01%	0.0001%	0.001%	2s
8	0.01%	0.0002%	0.001%	2s
9	0.01%	0.0003%	0.001%	3s
10	0.01%	0.0001%	0.001%	2s

Then, we considered CLINGO, with option `restart-on-model` (CLINGO-ROM), and the state-of-the-art industrial tool GUROBI Gurobi Optimization, LLC (2021), which are able to process instances in the wbo format.

The experiment was executed on the 10 instances of the Imperia hospital, which we remind is the biggest hospital of the ASL1 health authority, with a timeout of 60 seconds as in Section 4.6. Results are reported in Table 4.12, where for each solver and instance we report the required time, in seconds, to reach an optimal solution or, if an optimal solution is not found within the limit, the percentage gap between the sub-optimal solution found and the optimal one. The results obtained show that the new solution allows to reach an optimal solution with GUROBI. In particular, employing the new solution GUROBI is able to obtain an optimal solution in all the instances in a few seconds while, starting from the original encoding, the solver is able to reach the optimal solution in 4 instances. Concerning the performances of CLINGO-ROM, even if the solver is not able to reach optimal solutions, the obtained solutions improved and have a cost that is very near to the optimal one. Indeed, the average cost has a gap to the optimal cost that is smaller than 0.01%.

4.8 Conclusions

In this chapter, we have presented another Digital Health problem, the ORS problem. We have presented mathematical formulations for a basic and an extended version of the ORS problem. Moreover, we presented a new encoding for the basic version of the problem and

it turned out to be more efficient not only for ASP, but also for other logic-based solving approaches run on benchmarks obtained by automatic translation from the ASP formulation. This new version has been employed as a starting point for the adaptions tested on real data from the Italian health authority ASL1 Liguria. Results on some scenarios show that the ASP solutions produce satisfying schedules also when applied to such challenging, real data.

Chapter 5

Nuclear Medicine Scheduling Problem

Nuclear Medicine (Akhavizadegan et al. (2017); Pérez et al. (2011); Xiao et al. (2018)) is a medical specialty that uses radiopharmaceuticals, a particular kind of drug containing radioactive elements, to treat or diagnose diseases. According to data by the Italian Ministry of Health, almost 2 millions nuclear medicine exams have been carried on during 2022 in Italy[1]. The process of treating patients with this technique is complex since it involves multiple hospital resources and requires multiple steps at varying times. Moreover, often these drugs contain radioactive elements characterized by short half-lives, meaning that they decay rapidly after their preparation. Thus, the timing should be as precise as possible in order to obtain images of good quality. Addressing this problem effectively is crucial due to the nature of the diagnosed illnesses and treated through nuclear medicine, alongside the significant costs associated with this kind of technique. In fact, an efficient, possibly optimal, solution can reduce the waiting time of the patients and can thus increase the effective utilization of the resources, avoiding waste of time and resources. Nevertheless, reducing the unnecessary time spent by the patients in the hospital is vital for increasing the satisfaction of the patients.

The Nuclear Medicine Scheduling (NMS) problem consists of assigning patients to a day, in which the patient will undergo the medical check, the preparation, and the actual image detection process. The schedule of the patients consider the different requirements of the patients and the available resources, e.g., varying time required for different procedures and radiopharmaceuticals used, number of injection chairs and tomographs available. We followed the definition of the problem given by Medipass[2], leading provider of technological

[1] https://www.salute.gov.it/imgs/C_17_pubblicazioni_3425_allegato.pdf
[2] https://ergeagroup.com/it/

innovation across cancer care and diagnostic imaging in Italy, in collaboration with SurgiQ[3], an Italian company active in planning and scheduling solutions. The results of an experimental analysis conducted on real data demonstrate that our solution achieves satisfactory quality outcomes, even in time-constrained scenarios.

The chapter is structured as follows. Sections 5.1 and 5.2 present an informal description of the problem and a precise, mathematical formulation, respectively. Then, Section 5.3 presents the encoding, whose experimental evaluation is presented in Section 5.4. Finally, conclusions are presented.

5.1 Problem Description

The NMS problem consists of assigning patients to a day and to a tomograph and/or injection chair if required by the patient or the specific procedure. In our problem, for each day we consider a set of 120 time slots (TS), each representing 5 minutes. Each patient needs an exam and each exam is linked to a protocol defining the phases and the time required for each phase. We considered 11 different protocols. Each protocol can encompass up to four phases, i.e., (p_1) anamnesis, (p_2) medical check, (p_3) radiopharmaceuticals injection and bio-distribution time, and (p_4) image detection. Moreover, each phase can require a different amount of time depending on the exam. Table 5.1 shows the total time needed by each protocol and the partial time required by each phase, expressed in the number of time slots used, and if the protocol requires the infusion chair for phase (p_3).

Due to the high number of phases required by each patient and the variety of the considered protocols, in many clinics the schedule of the patients is sub-optimal. A sub-optimal schedule is problematic not only because of the high cost of the drugs and machines involved in the exams, but is particularly detrimental for the patients since the order and the time required by each phase, in particular the injection and the bio-distribution time, is fundamental for a proper image detection.

Different clinics have different resource availability and may have different requirements in defining a proper solution. Here we present the criteria followed in the clinic that provided us with the real data of the patients and that we use to define the problem. We considered a clinic with two rooms, each with one tomograph and three injection chairs. We started from a list of patients, each requiring a specific protocol, to be assigned in a day. A proper solution must satisfy the following conditions:

[3]https://surgiq.com/

Protocol Number	#TS for p_1	#TS for p_2	#TS for p_3	#TS for p_4	#TS total	Chair
813	3	2	0	8	13	NO
814	3	2	0	8	13	NO
815	2	2	4	6	14	YES
817	2	2	3	7	14	NO
819	2	2	5	7	16	YES
822	2	2	2	7	13	NO
823	2	2	10	7	21	YES
824	2	2	5	8	17	YES
827	2	2	2	7	13	NO
828	3	3	0	7	13	NO
888	2	2	2	9	15	YES

Table 5.1 Specifications for each protocol, including the number of time slots (TS) needed for each phase, the total time slots for the entire protocol, and if a chair is required (YES) or not (NO).

- a starting and an ending time should be assigned to every scheduled patient for each required phase;
- there must be at most two patients concurrently in the anamnesis phase;
- the injection phase must be done in an injection chair or on a tomograph according to the required protocol;
- the image detection phase must be done in a tomograph for all the considered protocols;
- each injection chair and tomograph can be used by just one patient at the same time;
- patients requiring an injection chair must be assigned to the tomograph of the same room;
- protocol identified by the id 815 cannot be assigned on the same day and tomograph for more than one patient.

The solution should also maximize the number of scheduled patients in the considered days and, to increase the satisfaction of the patients and the effectiveness of the exams, the solution should also try to minimize the unnecessary time spent in the clinic by the patients.

5.2 Formalization of the NMS problem

Let N be the set of reservation numbers, D be the set of days, and TS denote the set of time slots. Let R denote the set of rooms and $S = T \cup C \cup \{\varepsilon\}$ be the set representing the available resources, given by the union of the set T of tomographs, the set C of chairs and the element ε denoting that a resource is not required. Let PR be the set collecting the protocol numbers referred to exams and $\widetilde{PR}$ a subset of it containing only the protocols with a limit on the number of exams that can be executed on a tomograph for that protocol. Specifically, each protocol may comprise a maximum of four phases, represented by the set $P = \{p_1, p_2, p_3, p_4\}$.

Moreover, let:

- $\lambda : T \times \widetilde{PR} \to \mathbb{N}$ be the function that returns the maximum number of exams that can be executed on a tomograph of a specific protocol, for all the protocols that require a limitation;
- $\omega : P \times PR \to \mathbb{N}$ be the function that returns the number of time slots required for each phase of a specific protocol;
- $\beta : PR \to \{0,1\}$ be the function that assigns the value 1 if the protocol requires a chair for the injection phase; 0 if it requires a tomograph;
- $\alpha : S \times R \to \{0,1\}$ be the function that assigns the value 1 if there is an association between the resource and the room, 0 otherwise. Furthermore, let $\mathsf{A} = \alpha^{-1}(1)$ be the set collecting all the existing associations between resources and rooms.

To associate a reservation number, a day, and a protocol, we now introduce the notion of *registration*.

Definition 15. *A registration ρ is a function of the form $\rho : N \times D \times PR \to \{0,1\}$ such that $\rho(n,d,x) = 1$ if there exists a reservation n on a day d for the protocol x, 0 otherwise. Let $\mathsf{R} = \rho^{-1}(1) = \{(n,d,x) \in N \times D \times PR \mid \rho(n,d,x) = 1\}$ be the set collecting all the registrations.*

Consequently, we define the notion of *assignment* to link together a registration with a specific phase of the protocol under consideration and a time slot.

Definition 16. *An assignment τ is a function of the form $\tau : \mathsf{R} \times P \times TS \to \{0,1\}$ such that $\tau(n,d,x,p,y) = 1$ if a phase p and an initial time slot y are assigned to a registration. Let $\mathsf{T} = \tau^{-1}(1) = \{(n,d,x,p,y) \in R \times P \times TS \mid \tau((n,d,x),p,y) = 1\}$ be the set collecting all the assignments.*

Before presenting the main problem, we define the concept of *scheduling*, which connects an assignment with a resource and its allocation.

Definition 17. *A scheduling* σ *is a function of the form* $\sigma : \mathsf{T} \times \mathsf{A} \to \{0,1\}$ *such that* $\sigma(n,d,x,p,y,s,r) = 1$ *if there exists a reservation number* n *on day* d *for the protocol* x*, referred to the phase* p *on time slot* y*, using the resource* s *in the room* r*. The set* $\mathsf{S} = \sigma^{-1}(1) = \{\mathbf{t} = (n,d,x,p,y,s,r) \mid \sigma(\mathbf{t}) = 1\}$ *collects all the tuples eligible as scheduling.*

We can now define the *Nuclear Medicine Scheduling* (NMS) problem.

Definition 18 (NMS)**.** *The NMS problem is defined as the problem of finding a set* ψ *of tuples* $\mathbf{t} = (n,d,x,p,y,s,r) \in \mathsf{S}$ *that satisfies the following conditions:*

(c_1*)* $\forall d \in D,\ \forall y \in TS,\ \forall (s,r) \in \mathsf{A}\ |\{(n,d,x,p,y,s,r) \in \psi : p \neq p_1\}| \leq 1$*;*

(c_2*)* $\forall d \in D,\ \forall y \in TS,\ \forall (s,r) \in \mathsf{A}\ |\{(n,d,x,p,y,s,r) \in \psi : p = p_1\}| \leq 2$*;*

(c_3*)* $\forall x \in PR : \beta(x) = 1$*, it holds that* (n,d,x,p_3,y',c,r') *and* (n,d,x,p_4,y'',t,r'')*, with* $y' \neq y''$*, belong to* ψ *iff* $r' = r''$*;*

(c_4*)* $\forall d \in D,\ \forall x \in \widetilde{PR}, \forall (s,r) \in \mathsf{A}\ |\{(n,d,x,p,y,s,r) \in \psi : s = t\}| \leq \lambda(t,x)$*;*

(c_5*)* (n,d,x,p_i,y',s',r) *and* $(n,d,x,p_{i+1},y'',s'',r)$ *belong to* ψ *iff* $y'' \geq y' + \omega(p_i,x)$*;*

(c_6*)* $\forall (n,d,x,p,y,s,r) \in \psi$ *it holds that* $y + \omega(p,x) \in TS$*;*

(c_7*)* $\forall (n,d,x,p,y,s,r) \in \psi$

- *if* $p = p_1$ *it holds that* $s = \varepsilon$*;*
- *if* $p \in \{p_2,p_3\}$ *and* $\beta(x) = 1$ *it holds that* $s \in C$*;*
- *if* $p \in \{p_2,p_3\}$ *and* $\beta(x) = 0$ *it holds that* $s \in T$*;*
- *if* $p = p_4$ *it holds that* $s \in T$*.*

The specified conditions are necessary to enforce the following constraints: (c_1) each resource (chair or tomograph) can be used by at most one patient at a time; (c_2) at most two patients at a time slot are allowed during the anamnesis phase; (c_3) patients requiring an injection chair must be assigned to the tomograph of the same room; (c_4) the number of protocols executed on a single tomograph is limited; (c_5) given two consecutive phases p_i and p_{i+1} for a patient, the initial time slot of p_{i+1} must be consistent with respect to p_i, i.e. p_{i+1} must start after that p_i has terminated; (c_6) each schedule must not exceed the available

time slot; (c_7) the resources must be well distributed, i.e. the phase anamnesis does not include any resource, the phases 2 and 3 may require a chair or a tomograph depending on the protocol, and the last phase requires the usage of a tomograph.

As previously explained, an optimal solution aims to maximize the number of scheduled patients on the considered days while minimizing the idle time spent in the clinic by patients. In order to define the notion of the optimal solution, we want to evaluate the idle time spent by a patient. Accordingly, given (n,d,x,p_1,y',s',r) and $(n,d,x,p_4,y'',s'',r) \in \psi$ let $Htime(n) = (y'' + \omega(p_4,x)) - y')$ be the time spent by the patient in the hospital deriving from the scheduling and let $Rtime(n) = \sum_{p \in P} \omega(p,x)$ be the minimum time required to execute the protocol.

Definition 19 (Dominating Solution)**.** *Let $\delta_\psi = \sum_{n \in N} |Htime(n) - Rtime(n)|$ be the sum of the differences between the actual time required by a protocol and the time allocated by the solution for that protocol for each registration number n. Addiotionally, let $\Sigma_\psi = \{(n,d,x,p,y,s,r) \in \psi\}$ represent the set of all elements in a solution ψ. A solution ψ dominates a solution ψ' if*

- $|\Sigma_{\psi'}| < |\Sigma_\psi|$, *or if*
- $|\Sigma_{\psi'}| = |\Sigma_\psi| \Rightarrow \delta_\psi < \delta_{\psi'}$.

Finally, we define the notion of *maximal scheduling solution*.

Definition 20 (Maximal Scheduling Solution)**.** *A scheduling solution is maximal if any other scheduling solution does not dominate it.*

5.3 ASP Encoding

In this section, we present the ASP encoding for the problem presented in Section 5.1 and formalized in Section 5.2. The underlying encoding is based on the input language of CLINGO Gebser et al. (2016).

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

- instances of `reg(ID,D,PrID)` represent a registration with identification number `ID`, on day `D`, for a specific exam with protocol number `PrID`;
- instances of `avail(TS,D)` denote that the time slots `TS` is available on day `D`;

- instances of `exam(PrID,P,NumTS)` denote the features of an exam, where `PrID` denotes the exam protocol number, `P` indicates the phase, and `NumTS` specifies the time required for that phase in terms of the number of time slots;
- instances of `tomograph(T,R)` and `chair(C,R)` denote the allocation of the tomograph `T` and the chair `C` to the room `R`, respectively;
- instances of `required_chair(PrID)` denote the necessity of a chair for the phases 1 and 2 for the protocol `PrID`;

```
0 {x(ID, D, TS, PrID, 0) : avail(TS, D)} 1 :- reg(ID, D, PrID).
{x(ID, D, START, PrID, P+1) : avail(START,D), START >= TS+NumTS, START <
    TS+NumTS+6} = 1 :- x(ID, D, TS, PrID, P), exam(PrID, P, NumTS), P >= 0,
    P < 3.
:- x(ID, _, TS, PrID, 3), exam(PrID, 3, NumTS), TS + NumTS > 120.
timeAnamnesis(ID, TS..TS+NumTS-1) :- x(ID, D, TS, PrID, 0), exam(PrID, 0,
    NumTS).
:- #count{ID: timeAnamnesis(ID, TS)} > 2, avail(TS,D).
timeOccupation(ID, D, TS, END-1, PrID) :- x(ID, D, TS, PrID, 1), x(ID, D,
    END, PrID, 3).
res(ID, D, TS..END,0) :- timeOccupation(ID, D, TS, END, PrID),
    required_chair(PrID).
res(ID, D, TS..TS+NumTS-1,1) :- x(ID, D, TS, PrID, 3), exam(PrID, 3, NumTS),
    required_chair(PrID).
res(ID, D, TS..END+NumTS-1,1) :- timeOccupation(ID, D, TS, END, PrID),
    exam(PrID, 3, NumTS), not required_chair(PrID).
:- #count{ID: tomograph(T, ID, D), x(ID, D, _, PrID, _)} > N, limit(PrID,
    N), tomograph(T,_).
1 {chair(C, ID, D) : chair(C, _)} 1 :- x(ID, D, _, PrID, _),
    required_chair(PrID).
1 {tomograph(T, ID, D) : tomograph(T, _)} 1 :- x(ID, D, _, PrID, _).
:- chair(C, ID, D), tomograph(T, ID, D), chair(C, R1), tomograph(T, R2), R1
    != R2.
chair(C, ID, D, TS) :- chair(C, ID, D), res(ID, D, TS, 0).
tomograph(T, ID, D, TS) :- tomograph(T, ID, D), res(ID, D, TS, 1).
:- #count{ID: tomograph(T, ID, D, TS)} > 1, tomograph(T,_), avail(TS,D).
:- #count{ID : chair(C, ID, D, TS)} > 1, chair(C,_), avail(TS,D).
:~ not x(ID, D, _, _,0), reg(ID, D, _). [1@2, ID, D]
:~ x(ID, _, START, PrID, 0), x(ID, _, END, _, 3), cost(PrID, NumTS), END -
    START - NumTS >= 0. [END - START - NumTS@1, ID]
```

Figure 5.1 ASP encoding of the problem.

- instances of `cost(PrID, NumTS)` represent the total duration in terms of time slots, denoted as `NumTS`, of the phases within the protocol identified by `PrID`;
- instances of `limit(PrID,N)` denote the maximum number `N` of exams with protocol number `PrID` that can be executed on the same tomograph in a day.

The output consists of assignments represented by the atom `x(ID,D,TS,PrID,P)` where the intuitive meaning is that the registration with identification number `ID` for the exam with protocol number `PrID`, regarding the phase `P`, has been scheduled for the day `D` during the time slot `TS`. Additionally, it includes atoms `chair(C,ID,D)` and `tomograph(T,ID,D)`, denoting the resource (either the chair `C` or the tomograph `T`, respectively) allocated to the patient `ID` on the day `D`.

Encoding. The related encoding is shown in Figure 5.1 and is described next. To simplify the description, we denote as r_i the rule appearing at line i of Figure 5.1.
Rule r_1 may assign or not the registration with identifier `ID` to a specific time slot `TS` for the day `D` in phase 0, i.e. the phase of the anamnesis. Rule r_2 assigns an already scheduled session for a given phase `P` to the subsequent planned phases, under the condition that the start of the phase does not extend beyond the latest available time slot for a session on that day. Furthermore, it ensures that the subsequent phase starts at most 5 time slots after the ending of the previous phase. Rule r_3 ensures that the duration of the final phase is also consistent with the time slots, ensuring that all phases are completed within the specified limit. Rule r_4 keeps track of the time slots allocated to a patient during phase 0 via the auxiliary atom `timeAnamnesis(ID, TS)`. Rule r_5 restricts the number of patients during the anamnesis phase to a maximum of two. Rule r_6 produces the auxiliary atom `timeOccupation(ID,D,TS,END,PrID)`, representing the duration needed for each patient `ID` from the initial time slot `TS` of phase 1 to the final one `END` of phase 2, concerning the protocol `PrID` on the day `D`. Rule r_7 produces the auxiliary atom `res(ID, D, TS, 0)` for each time slot derived from the previous rule. Specifically, the constant `0` denotes that a chair is required for each of these time slots. Rules r_8 and r_9 produce the atom `res(ID,D,TS,1)`, which differs from the previous one for the constant `1`, indicating the use of a tomograph. From rule r_8, it is inferred that a tomograph is employed during phase 3, whereas rule r_9 indicates the tomograph's usage from phase 1 to 3, according to the atom `timeOccupation`. Rule r_{10} ensures that the limit of protocols that can be executed on a single tomograph is respected. Rule r_{11} produces the atom `chair(C,ID,D)` representing the assignment of a chair `C` on the day `D` to the patient `ID` when the protocol `PrID` requires a chair. Rule r_{12}

produces the atom `tomograph(T,ID,D)` representing the assignment of a tomograph `T` on the day `D` to the patient `ID`. Rule r_{13} prevents the patient who moves from the chair to the tomograph from changing room. Rule r_{14} and r_{15} generate the atoms `chair(C,ID,D,TS)` and `tomograph(T,ID,D,TS)` respectively, indicating the time slots `TS` during which the chair `C` and tomograph `T` are utilized by the patients `ID` on the day `D`. Rule r_{16} and r_{17} ensure that at most one patient is assigned to each tomograph and chair in every time slot, respectively. Finally, the optimal solution is achieved through the application of rule r_{18}, which minimizes (with the highest priority) the number of registrations not assigned to a schedule, and rule r_{19}, which minimizes the duration of patient appointments beyond the time necessary to perform the test.

5.4 Experimental Results

In this section, we report the results of an empirical analysis of the NMS problem via ASP. The data used for the empirical analysis are real data coming from a medium size hospital provided by Medipass. We performed experiments on an Apple M1 CPU @ 3.22 GHz machine with 8 GB of physical RAM. The ASP system used was CLINGO Gebser et al. (2016) 5.6.2, using parameters *--restart-on-model* for faster optimization and *--parallel-mode 2* for parallel execution: we conducted a preliminary analysis with various options and found these parameters to be the most effective. The encoding and the input presented in ths chapter can be found at https://github.com/MarcoMochi/JLC2024NSP/tree/main/Fair.

We tested instances of more than a year of daily exams. In particular, we tested 366 instances, each corresponding to a weekday excluding weekends, resulting in a total of 72 weeks. The solution schedules the patients in a day in a range of 10 hours, split into 120 time slots of 5 minutes. Each patient is linked to one of the possible exams. In particular, one exam protocol is required by more than 85% of the patients, thus, the majority of the patients need an exam protocol that requires 2 time slots for the anamnesis and other 2 time slots for medical preparation, 10 time slots for the drug injection and the bio-distribution time and, at last, the image detection requires 7 time slots. The other patients can be associated with one of the other 11 possible protocols. The schedule is calculated with two rooms as available resources for the procedure. In each room, there is a tomograph and 3 chairs. The number of patients requiring exam changes every day but, on average, there are 29 patients to be scheduled, with a maximum of 37 patients.

The results obtained by testing the encoding presented in Section 5.3 with the real data are reported in the following. We decided to schedule the patients with a time limit of 60

seconds. This allows the hospital to get a result in a very short time and to get an estimation of the usefulness of our solution in a time-constrained environment. Figure 5.2 presents the average, the maximum, and the minimum times required to obtain a solution according to the number of patients to be scheduled. In this way, it is possible to analyze the waiting time a hospital should expect to get a solution, depending on the number of patients to be assigned. From the graph it can be seen that up to 29 patients all the instances are optimally solved before the time limit. While on average the instances are solved optimally in less than 60 seconds in the instances with less than 36 patients. We were able to test the encoding in just a few days having 36 and 37 patients and, in that case, the encoding is not able to optimally schedule the patients. In general, we are able to find the optimal solution for more than 60% of the instances.

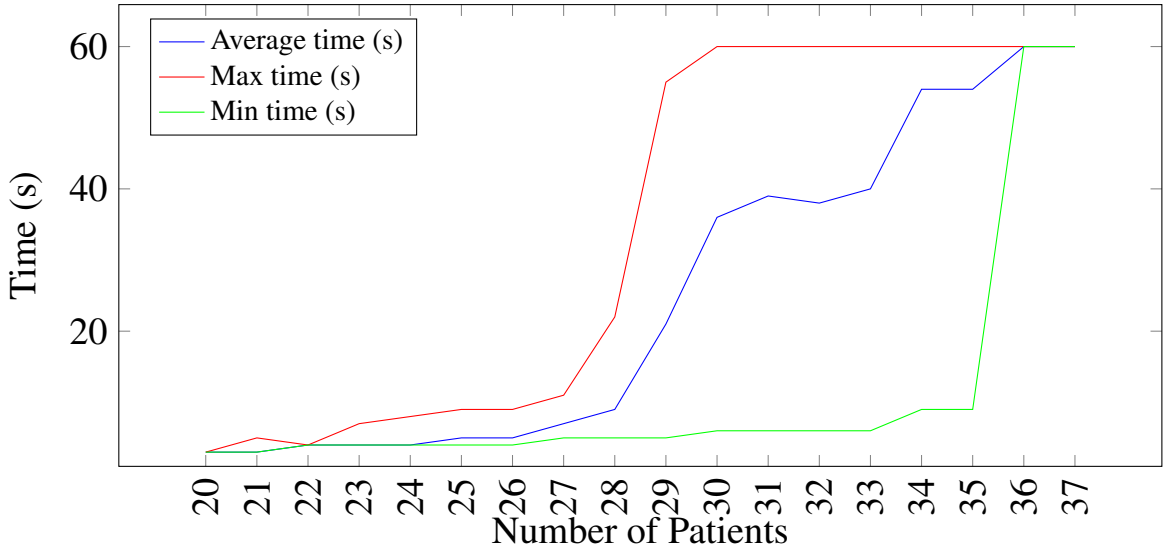

Figure 5.2 Average time (seconds) required to solve the instances with the different number of patients.

After analyzing the time required to generate schedules on different days, we proceed to evaluate the quality of the results obtained. Specifically, we first examine the number of registrations that the solution fails to assign during the day, as this was the primary optimization criterion. Subsequently, we assess the second optimization criterion.

To gain a better understanding of the schedule's quality, we present the results considering all schedules and we specifically focus on non-optimal solutions. Even when the solution is optimal, some days still have a high number of unassigned patients due to constraints imposed by the data.

When considering instances where the encoding failed to find an optimal solution, we observed that in the majority of cases (more than 80%), the solution assigns the exam to over 80% of the possible patients. In the remaining instances, the solution still manages to assign over 70% of the total patients in all cases. These findings indicate that even under strict time limits, the results remain of satisfactory quality, a crucial aspect for operational scenarios.

Next, we summarize the results obtained across all instances, whether optimally or non-optimally solved. Figure 5.3 illustrates the number of schedules with at least a certain percentage of assigned patients. Unlike the non-optimal solutions, optimally solved instances occasionally exhibit a percentage of assigned patients below 60%, attributed to resource limitations, which we further investigate. However, overall, the majority of solutions have a patient assignment rate exceeding 80%.

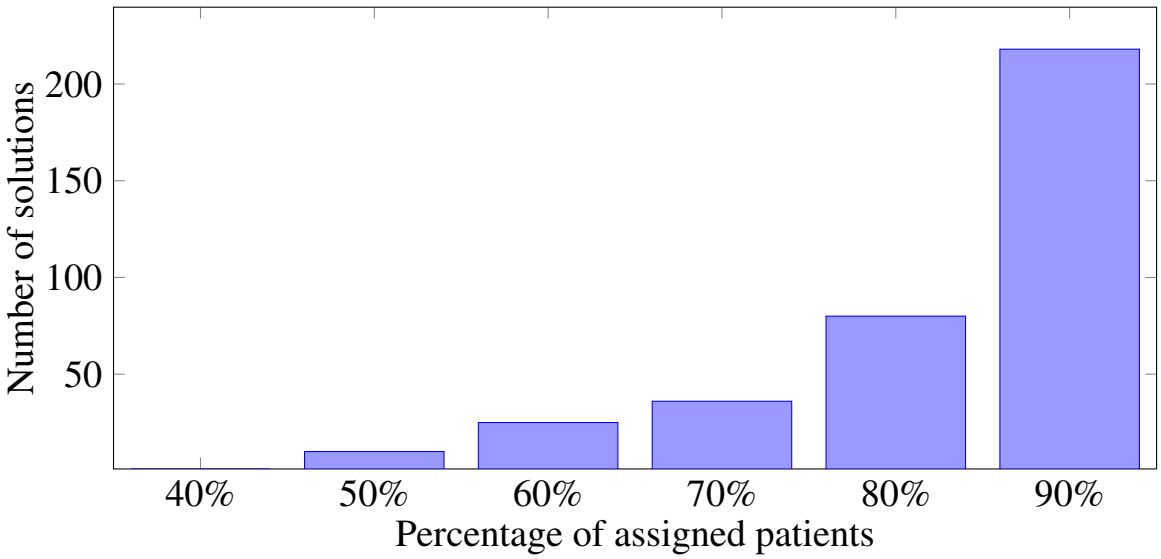

Figure 5.3 Number of solutions with a percentage of assigned patients at least equal to the corresponding x-axis value and smaller than the next value.

As previously discussed, some optimal solutions exhibit a low percentage of assigned patients. Here, we aim to analyze in more details this scenario to ensure that no better solution could have been identified. Specifically, we focus on the only instance where the schedule fails to assign an exam to more than 50% of the patients. In this case, the schedule assigns exams to 16 out of the total 33 patients. Upon closer examination, we find that among the 33 patients, 14 require protocol with id 823, while the remaining 19 require protocol with id 815. The solution successfully assigns exams to all 14 patients with protocol id 823. However, due to the nature of the protocol with id 815, the solution can only assign one patient with

this protocol to each tomograph. Consequently, it manages to assign exams to just 2 patients. This limitation explains the low percentage of assigned patients in this instance.

Finally, after examining the results related to the first optimization criterion, we proceed to analyze the second one. Specifically, we focus on the average waiting time for each patient, quantified as the unnecessary number of time slots spent in the hospital. In over 70% of instances, the encoding successfully assigns patients with zero waiting times. This indicates that in these solutions, patients can adhere to their protocols optimally. These results greatly improve those obtained by the hospital in terms of unnecessary time spent in the hospital by the patients. Indeed, in the original schedule of the hospital, just 19% of the considered days had zero waiting times for each patient. Moreover, while in the ASP-based solution we have 95% of days with less than 5 time slots of waiting time, in the original schedule there were just 75% of days with this average waiting time.

5.5 Conclusions

In this chapter, we have presented an analysis of the Nuclear Medicine Scheduling (NMS) 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 solutions. Specifically, we proposed an ASP encoding and tested it using real data. The results obtained from real data show satisfying outcomes in terms of quality.

Part II

Increase Explainability and Fairness

In this part of the thesis, we will present our contributions to one of the most important topics in AI, the Explainability of AI solutions (XAI) and their interpretations. Many works were devoted to the definition of XAI (Gunning et al., 2019) and its applications to different kinds of methods. In the following chapters of the thesis, we will present our approaches to this field, showing two different tools to increase the explainability and interpretability and one ASP encoding to solve the Fairness problem. We will start focusing on the XAI problem, where we will define the notions required to determine the reason for the existence or non-existence of a solution, allowing the usage of a richer ASP syntax, with respect to the State of the Art. Moreover, we will present the tool (E-ASP) developed to allow the usage of such a definition. Other than the problems linked to the interpretation of the solutions, another important topic, when a strict collaboration with non-experts is required, is the ability to express the models and the solutions in a high-level syntax. ASP has been described as a high-level declarative language, however, in certain scenarios, when the complexity of the problems increases, even a language such as ASP can become quite difficult to read for non-experts. These reasons led us to the definition of a Controlled Natural Language (CNL) that allows the translation from English sentences to ASP encodings. In the related chapter, we will present the definition of such CNL and the usage of the developed tool (CNL2ASP). Moreover, to comprehend the tool's usability, we will present three examples of real-world problems comparing the performances of the state-of-the-art encodings and the encoding obtained through the tool. Finally, to better understand the improvements in readability of such a tool, we will present a preliminary analysis conducted to assess the usability and readability of the proposed CNL.

Having presented works related to the explainability of the solution, it is crucial to address another problem that could arise by using AI solutions. It is the Fairness problem. Indeed, it is crucial to deliver a working solution and be mindful of the issues it might inadvertently create. For instance, in the context of Digital health, even an optimized AI solution can lead to financial waste if it results in some high-cost machines remaining underutilized, driving up expenses without corresponding improvements in patient care. This underutilization often arises when the system is designed to maximize the number of completed tasks by prioritizing shorter, less time-intensive protocols over longer ones. In this scenario, the solution may favor scheduling multiple short procedures rather than making full use of available resources. Even worse, beyond financial impacts, these biases can have serious ethical and legal implications, potentially compromising the quality of care and creating unequal treatment among patients. Fairness rules, while not strictly required to solve the technical problem, play a critical role in achieving a more balanced and inclusive outcome

by mitigating these unintended biases. For example, by incorporating fairness guidelines, AI systems can avoid systematically prioritizing certain types of patients or procedures, ensuring that resources are used equitably and that access to care is distributed fairly across all patients, not just those whose needs align with the model's optimization goals. In this direction, in a later chapter, we will present an encoding, related to the Nuclear Medicine Scheduling problem, developed to address the Fairness problem arose with a traditional ASP solution.

Chapter 6

Explainability

Recently, a demand has emerged for AI systems that not only provide answers but also explain their solutions. This transparency is necessary to build trust, as users want to know the reasons behind an AI-based decision. Moreover, in highly regulated sectors like finance and healthcare, offering explanations is essential for compliance with legal standards.

ASP (Baral, 2003; Brewka et al., 2011; Gelfond and Lifschitz, 1988; Janhunen and Niemelä, 2016) has emerged as a powerful declarative programming paradigm, offering a flexible framework for addressing complex combinatorial problems.

For these reasons, also in ASP, explanations are needed to understand the behavior and outcomes of ASP programs. We can identify two important types of explanations in ASP (Fandinno and Schulz, 2019). The first one includes explanations on the entire program, explaining why no valid solution has been found or why an answer set has been computed by showing a particular combination of rules that caused these solutions (Brain et al., 2007; Dodaro et al., 2019c; Gebser et al., 2008). The second type focuses on explaining the presence (resp. the absence) of an atom in an answer set giving insights into why a specific atom is (resp. is not) included in a given solution (Brain and De Vos, 2005).

In this chapter, we provide a practical contribution to the aforementioned context. Specifically, we present a tool, called E-ASP, that is designed to provide explanations of ASP programs. E-ASP extends the techniques implemented in DWASP (Dodaro et al., 2015a, 2019c) to support explanations for both the presence and absence of an atom in a given answer set. Moreover, E-ASP offers flexibility in the type of explanations provided, allowing users to choose between rule-based and/or literal-based explanations accommodating their specific needs and preferences. As an additional feature, E-ASP supports explanations over aggregates via a user-friendly stepping-through approach, where users can navigate through literals occurring in aggregate sets if needed for the explanations. Finally, as an

implementation contribution, the novel tool is based on the state-of-the-art system CLINGO (Gebser et al., 2016) and it does not require any modification of the internal workings of the solver, therefore ensuring total compliance with future versions of CLINGO. We will present an example of the usage of such a tool using as encoding a State of the Art encoding for a well-known problem, complementing our contributions to the Digital Health field, not only presenting solutions but also presenting solutions to increase their explainability.

6.1 Preliminaries

In this section, we recall key aspects of the debugging approach of DWASP proposed by Dodaro et al. (2019c).

Debugging Approach of DWASP We now recall some definitions and properties that are useful in the remainder of the chapter. For more details, we refer the reader to (Dodaro et al., 2019c).

We will define the *choice rule* of the form $\{p\} \leftarrow$, where p is an atom, as a syntactic shortcut for the rule $p \vee p' \leftarrow$, where p' is a fresh atom not appearing elsewhere in the program.

Whereas, given a list of weighted literals $L = (\langle w_1 : \ell_1 \rangle, ..., \langle w_n : \ell_n \rangle)$, we define $W(L) = \sum_{k=1}^{n} w_k$ and $pr(i,L)$ as the projection of the first s elements in L such that $s = min\{j \in [1..n] \mid \sum_{k=1}^{j} w_k \geq i\}$; note that, when $i \leq 0$, then $pr(i,L)$ is an empty list.

A *test case* T is a set of literals and Π_T denotes $\{\leftarrow \overline{\ell} \mid \ell \in T\}$. A test case T *fails* for a program Π if $\Pi \cup \Pi_T$ is incoherent.

For a program Π, $\mathscr{B} \subseteq \Pi$ is a set of rules called *background knowledge*. In the approach of DWASP such rules are considered to be correct. For a rule r, $\rho(r)$ denotes $H(r) \leftarrow B(r) \cup \{debug(r)\}$, and for an atom p, $\sigma(p)$ denotes $p \leftarrow not\ support(p)$.

The *debugging program* of Π, denoted as $\Delta_{(\Pi,\mathscr{B})}$, is

$$\bigcup_{r \in \Pi \setminus \mathscr{B}} \rho(r) \cup \bigcup_{p \in atoms(\Pi)} \sigma(p) \cup \mathscr{B}.$$

In the following, $\mathscr{D}$ and $\mathscr{S}$ denote $\{debug(r) \mid debug(r) \in atoms(\Delta_{(\Pi,\mathscr{B})})\}$ or $\{support(p) \mid support(p) \in atoms(\Delta_{(\Pi,\mathscr{B})})\}$, respectively. The *extended debugging program*, denoted as $\Delta^*_{(\Pi,\mathscr{B})}$, is defined as $\Delta_{(\Pi,\mathscr{B})} \cup \{\{p\} \leftarrow \mid p \in \mathscr{D} \cup \mathscr{S}\}$.

We recall that the extended debugging program preserves some properties of the original program.

Proposition 1 (see (Dodaro et al., 2019c))**.** *Let* Π *be a program,* $\mathscr{B}$ *a background knowledge, and* T *a test case. In addition, let* $\mathscr{P}$ *denote* $\{p \leftarrow \mid p \in \mathscr{D} \cup \mathscr{S}\}$*. Then, the program* $\Gamma_{(\Pi,\mathscr{B},T)} = \Delta^{*}_{(\Pi,\mathscr{B})} \cup \Pi_T \cup \mathscr{P}$ *is coherent iff* $\Pi \cup \Pi_T$ *is coherent.*

As a consequence, if $\Gamma_{(\Pi,\mathscr{B},T)}$ is incoherent, also $\Pi \cup \Pi_T$ is incoherent. In this case, $\Gamma_{(\Pi,\mathscr{B},T)}$ can be used to find the *reason of incoherence*, that is, a set $\mathscr{R} \subseteq \mathscr{P}$ such that $(\Gamma_{(\Pi,\mathscr{B},T)} \setminus \mathscr{P}) \cup \mathscr{R}$ is incoherent. Furthermore, $\mathscr{R}$ is *minimal* if there is no set of rules $\mathscr{R}' \subset \mathscr{R}$ such that $\mathscr{R}'$ is a reason of incoherence for $\Gamma_{(\Pi,\mathscr{B},T)}$.

It is important to observe that we defined a reason of incoherence as a subset of $\mathscr{P}$ for simplification purposes. However, $\mathscr{P}$ comprises a set of facts involving *debug* and *support* atoms. Therefore, when computing explanations, it is crucial to interpret the results according to the original rules and atoms. For instance, if $debug(r_1) \leftarrow$ and $support(p) \leftarrow$ are part of the reason for incoherence, those are interpreted by associating them with r_1 and p, respectively.

Example 1 (Running Example)**.** *Let* Π_{run} *be the following program:*

$$r_1: \ p_1 \vee p_2 \leftarrow \qquad r_2: \ p_3 \vee p_4 \leftarrow$$
$$r_3: \ \leftarrow \#count\{p_1,p_2,p_3,p_4\} \leq 1 \qquad r_4: \ \leftarrow \#count\{p_1,p_2,p_3,p_4\} \geq 4$$

Let $\mathscr{B}_{run} := \{r_4\}$ *be the background knowledge, and* $T_{run} := \{not\ p_1, not\ p_2, not\ p_3, p_4\}$ *be the test case. Then,* $\Gamma_{(\Pi_{run},\mathscr{B}_{run},T_{run})}$ *is the following:*

$$\begin{aligned}
\rho(r_1): &\ p_1 \vee p_2 \leftarrow debug(r_1) & \rho(r_2): &\ p_3 \vee p_4 \leftarrow debug(r_2)\\
\rho(r_3): &\ \leftarrow \#count\{p_1,p_2,p_3,p_4\} \leq 1, debug(r_3) & r_4: &\ \leftarrow \#count\{p_1,p_2,p_3,p_4\} \geq 4\\
r_5: &\ \leftarrow p_1 & r_6: &\ \leftarrow p_2\\
r_7: &\ \leftarrow p_3 & r_8: &\ \leftarrow not\ p_4\\
\sigma(p): &\ p \leftarrow not\ support(p) & &\qquad \forall p \in \{p_1,p_2,p_3,p_4\}\\
\mathscr{D}: &\ \{debug(r)\} & &\qquad \forall r \in \{r_1,r_2,r_3\}\\
\mathscr{S}: &\ \{support(p)\} & &\qquad \forall p \in \{p_1,p_2,p_3,p_4\}\\
\mathscr{P}_1: &\ debug(r) \leftarrow & &\qquad \forall r \in \{r_1,r_2,r_3\}\\
\mathscr{P}_2: &\ support(p) \leftarrow & &\qquad \forall p \in \{p_1,p_2,p_3,p_4\}
\end{aligned}$$

where $\Pi_{T_{run}} := \{r_5,r_6,r_7,r_8\}$*,* $\mathscr{P} := \mathscr{P}_1 \cup \mathscr{P}_2$*, and* $\{debug(r_1) \leftarrow\}$ *is a minimal reason of incoherence for* $\Gamma_{(\Pi_{run},\mathscr{B}_{run},T_{run})}$.

6.2 Explanation of ASP Programs

In this section, we present the approach implemented in E-ASP to explain why, given a program Π and an answer set A of Π, an atom α is included (resp. is not included) in A. Specifically, E-ASP takes advantage of the reason for incoherence proposed by Dodaro et al. (2019c) and is implemented in DWASP. To this end, E-ASP supports three different ways of computing explanations: *(i) rule-based*, i.e., the explanation is based only on the rules of the original program, *(ii) literal-based*, i.e., the explanation is based only on the answer set A, *(iii) mixed*, i.e., the explanation combines the approaches *(i)* and *(ii)*.

In all cases, the first step consists of providing a way to construct an incoherent program starting from a coherent one paired with a fixed answer set. Intuitively, the idea is to create a program where the answer set A remains unchanged, except for the atom to be explained, whose truth value is forced to be flipped from its original value in A. To this end, we provide the following proposition.

Proposition 2. *Let Π be a program, A an answer set of Π, and α an atom in $atoms(\Pi)$. Let ℓ_α denote α if $\alpha \in A$, otherwise ℓ_α denotes not α. Then, the program*

$$\widetilde{\Pi} := \Pi \cup \Pi_A \cup \{\leftarrow \ell_\alpha\} \tag{6.1}$$

with $\Pi_A := \{\leftarrow \overline{\ell} \mid \ell \in A \cup \overline{(atoms(\Pi) \setminus A)}, \ell \neq \ell_a\}$ is incoherent.

Proof. Let $A = \{p_1, \ldots, p_n\}$ and $atoms(\Pi) \setminus A = \{p_{n+1}, \ldots, p_m\}$. We show the claim for both scenarios: (i) when $\alpha \in A$, and (ii) when $\alpha \notin A$. By contradiction, let us assume that $\widetilde{\Pi}$ is coherent. For scenario (i), without loss of generality, we choose $\alpha = p_1$. Hence, the resulting program contains all rules of Π in addition to the following constraints:

$$\leftarrow p_1,\ \leftarrow not\ p_2,\ \cdots,\ \leftarrow not\ p_n,\ \leftarrow p_{n+1},\ \cdots,\ \leftarrow p_m.$$

In order to satisfy the above constraints, an answer set A' of $\widetilde{\Pi}$ is such that: (i) the atoms $p_{n+1}, \ldots, p_m$ that are considered false in A must still be false in A'; (ii) the atoms $p_2, \ldots, p_n$ already contained in A must also be included in A', except for p_1. Accordingly, if $A' = \{p_2, \ldots, p_n\}$ is an answer set of $\widetilde{\Pi}$, it is also an answer set of Π because $\Pi^{A'} = \widetilde{\Pi}^{A'}$. By the anti-chain property COSTANTINI and PROVETTI (2005), this contradicts the assumption that A is an answer set of Π, leading to the conclusion that $\widetilde{\Pi}$ must be incoherent.

For scenario (ii), we choose $\alpha = p_{n+1}$. The program $\widetilde{\Pi}$ contains all rules of Π in addition to the following constraints:

$$\leftarrow not\ p_{n+1},\ \leftarrow not\ p_1,\ \cdots,\ \leftarrow not\ p_n,\ \leftarrow p_{n+2},\ \cdots,\ \leftarrow p_m.$$

Hence, only $A' = \{p_1, \ldots, p_n, p_{n+1}\}$ can be an answer set of $\widetilde{\Pi}$, which is again contradicted by the anti-chain property because $\Pi^{A'} = \widetilde{\Pi}^{A'}$ yields A' as an answer set of Π. □

The second step consists of constructing the debugging programs for the different types of explanation. Specifically, the distinction among the three constructions lies in which rules are adorned with atoms of the form *debug*. To this end, we provide the following proposition.

Proposition 3. *Let Π be a program, A an answer set of Π, and α an atom in $atoms(\Pi)$. Let ℓ_α denote α if $\alpha \in A$, otherwise ℓ_α denotes not α. Consider $\widetilde{\Pi}$ as in (6.1) and the following debugging programs:*

(i) $\Gamma_R := \Gamma_{(\Pi \cup \{\leftarrow \ell_\alpha\}, \mathscr{B}, T)}$ *with* $\mathscr{B} := \{\leftarrow \ell_\alpha\}$ *and* $T := (A \cup \overline{(atoms(\Pi) \setminus A)}) \setminus \{\ell_\alpha\}$,

(ii) $\Gamma_L := \Gamma_{(\widetilde{\Pi}, \mathscr{B}, T)}$ *with* $\mathscr{B} := \Pi \cup \{\leftarrow \ell_\alpha\}$ *and* $T := \emptyset$, *and*

(iii) $\Gamma_M := \Gamma_{(\widetilde{\Pi}, \mathscr{B}, T)}$ *with* $\mathscr{B} := \{\leftarrow \ell_\alpha\}$ *and* $T := \emptyset$.

Then, Γ_R, Γ_L, and Γ_M are incoherent.

Proof. The programs Γ_R, Γ_L, and Γ_M coincide with $\widetilde{\Pi}$ in (6.1), and thus the claim follows from Proposition 2. □

Consequently, we can characterize the different types of explanation via the following definition.

Definition 21. *Let Π be a program, A an answer set of Π, and α an atom in $atoms(\Pi)$. Let ℓ_α denote α if $\alpha \in A$, otherwise ℓ_α denotes not α.*

1. *A rule-based explanation is a minimal reason of incoherence of the program Γ_R.*
2. *A literal-based explanation is a minimal reason of incoherence of the program Γ_L.*
3. *A mixed explanation is a minimal reason of incoherence of the program Γ_M.*

Example 2 (Continuing Example 1)**.** *Reconsider the program Π_{run}, let $A := \{p_1, p_4\}$ be an answer set, and $\alpha := p_1$. Then, Π_{run_A} consists of the following constraints:*

$$r_{p_2}:\ \leftarrow p_2 \quad r_{p_3}:\ \leftarrow p_3 \quad r_{p_4}:\ \leftarrow not\ p_4.$$

1. $\{debug(r_1) \leftarrow\}$ *is a rule-based explanation of* $\Gamma_{(\Pi_{run} \cup \{\leftarrow p_1\}, \{\leftarrow p_1\}, \{not\ p_2, not\ p_3, p_4\})}$, *meaning that* $r_1 \in \Pi_{run}$ *explains why* $p_1 \in A$. *Note that* $\{debug(r_3) \leftarrow\}$ *is also a rule-based explanation for similar reasons.*

2. $\{debug(r_{p_2}) \leftarrow\}$ *is a literal-based explanation of* $\Gamma_{(\Pi_{run} \cup \Pi_{run_A} \cup \{\leftarrow p_1\}, \Pi_{run} \cup \{\leftarrow p_1\}, \emptyset)}$, *meaning that* $p_1 \in A$ *because* $p_2 \notin A$.

3. $\{debug(r_{p_2}) \leftarrow, debug(r_1) \leftarrow\}$ *is a mixed explanation of* $\Gamma_{(\Pi_{run} \cup \Pi_{run_A} \cup \{\leftarrow p_1\}, \{\leftarrow p_1\}, \emptyset)}$, *meaning that* $p_1 \in A$ *because* $p_2 \notin A$ *and* $r_1 \in \Pi_{run}$. *Moreover, note that* $\{debug(r_{p_2}) \leftarrow , debug(r_{p_3}) \leftarrow , debug(r_3) \leftarrow\}$ *is also a mixed explanation of* $\Gamma_{(\Pi_{run} \cup \Pi_{run_A} \cup \{\leftarrow p_1\}, \{\leftarrow p_1\}, \emptyset)}$, *expressing that* $p_2, p_3 \notin A$ *and* $r_3 \in \Pi_{run}$ *explain why* $p_1 \in A$.

Aggregates are presented in a different way by the E-ASP tool. Here is given a detail presentation. When a rule containing an aggregate atom is in a reason of incoherence, E-ASP presents to the user an explanation with three different levels of detail. In this way, we avoid overwhelming the user with unnecessary information by initially abstracting the aggregate set. Then, if the user asks for more details, we provide an intermediate level of explanation before enumerating all the elements of the aggregate set.

Let Π be a program, A be an answer set of Π, and ℓ be the literal to explain. For an aggregate atom p of the form $f(S) \odot k$, we denote with $trues(S,A)$ and $falses(S,A)$ the lists of true and false literals in S, matching their truth value w.r.t. A. Our implementation takes advantage of the CLINGO API (see Section 6.3) to compute the two lists, sorting the literals according to the solver's derivation order. In the first stage of explaining p, E-ASP compactly represents its explanation, without expressing the elements contained in S, thus treating p as an atom. Subsequently, if the user decides to investigate the aggregate atom further, our system returns an explanation, $\mathscr{L} = \langle T, F \rangle$, where T and F are lists of true and false literals w.r.t S, respectively. The definition of $\mathscr{L}$ differs depending on the scenario listed below, where for the sake of simplicity, we will not consider the case when $\odot \in \{<, >\}$, as the expressions are equivalent to $\leq k-1$ and $\geq k+1$.

1. When $A \models p$ and $\ell \in S$:

   - if $f(S) \geq k$, then $\mathscr{L} := \langle [], falses(S,A) \rangle$;
   - if $f(S) \leq k$, then $\mathscr{L} := \langle trues(S,A), [] \rangle$;
   - if $f(S) = k$, then $\mathscr{L} := \langle trues(S,A), falses(S,A) \rangle$;
   - if $f(S) \neq k$, then $\mathscr{L} := \langle trues(S,A), falses(S,A) \rangle$.

2. When $A \not\models p$ and $\ell \in S$:

- if $f(S) \geq k$, then $\mathscr{L} := \langle trues(S,A), [] \rangle$;
- if $f(S) \leq k$, then $\mathscr{L} := \langle [], falses(S,A) \rangle$;
- if $f(S) = k$, then $\mathscr{L} := \langle trues(S,A), falses(S,A) \rangle$;
- if $f(S) \neq k$, then $\mathscr{L} := \langle trues(S,A), falses(S,A) \rangle$.

3. When $A \models p$ and $\ell \notin S$:

   - if $f(S) \geq k$, then $\mathscr{L} := \langle pr(k, trues(S,A)), [] \rangle$;
   - if $f(S) \leq k$, then $\mathscr{L} := \langle [], pr(W(S) - k, falses(S,A)) \rangle$;
   - if $f(S) = k$, then $\mathscr{L} := \langle trues(S,A), falses(S,A) \rangle$;
   - if $f(S) \neq k$, then $\mathscr{L} := \langle trues(S,A), falses(S,A) \rangle$.

   Notice that, when $k > W(S)$, then $f(S) \leq k$ is true for any interpretation; therefore, the computed $\mathscr{L}$ is a pair of empty lists since no elements in S is useful for the explanation.

4. When $A \not\models p$ and $\ell \notin S$, $\mathscr{L}$ is computed according to case 3 (i.e., when $A \models p$ and $\ell \notin S$), replacing p with its complementary aggregate computed as follows:

$$\overline{f(S) \odot k} = \begin{cases} f(S) \leq k-1 & \text{if } \odot \text{ is } \geq \\ f(S) \geq k+1 & \text{if } \odot \text{ is } \leq \\ f(S) \neq k & \text{if } \odot \text{ is } = \\ f(S) = k & \text{if } \odot \text{ is } \neq \end{cases}$$

If more information than filtered in $\mathscr{L}$ is requested by the user, E-ASP provides $\langle trues(S,A), falses(S,A) \rangle$ comprising the entire lists of true and false literals.

Example 3 (Continuing Example 2)**.** *Reconsider the program Π_{run}, let $A := \{p_1, p_4\}$ be an answer set, and $\alpha := p_1$. Assume that the user receives an explanation containing r_3, e.g. $\{debug(r_3) \leftarrow\}$ or $\{debug(r_{p_2}) \leftarrow, debug(r_{p_3}) \leftarrow, debug(r_3) \leftarrow\}$. At this point, (s)he can expand the aggregate $\#count\{p_1,p_2,p_3,p_4\} \leq 1$, which is false w.r.t. A, i.e., $A \not\models \#count\{p_1,p_2,p_3,p_4\} \leq 1$. Therefore, the user would receive $\mathscr{L} := \langle [], [p_2,p_3] \rangle$, meaning that $p_1 \in A$ because $\#count\{p_1,p_2,p_3,p_4\} \leq 1$ is false w.r.t. A and $p_2, p_3 \notin A$, therefore $p_1 \notin A$ would have made the aggregate true.*

Having presented the functionalities of the E-ASP approach, we now present how an explanation session would work. An explanation session begins with a user executing

a non-ground ASP program on the user interface of E-ASP. Upon processing the input program, E-ASP initially computes the corresponding ground ASP program, denoted as Π, by executing CLINGO (Gebser et al., 2016) with specific options, such as `-mode=gringo`, `-text`, and `-keep-facts`. The `-keep-facts` option is particularly useful for preventing certain simplifications made by CLINGO during grounding, while additional simplifications are restricted using an approach similar to the one described by Dodaro et al. (2019c). Subsequently, after obtaining the ground program, E-ASP invokes CLINGO to compute an answer set, which is then displayed to the user. This call to CLINGO uses its Python API along with the modules *Propagator* and *Observer*. These modules are needed for capturing the derivation order of the atoms in an answer set, used for methods like the stepping-through approach for aggregates (see Section 6.2). At this point, the user can select an atom (excluding input facts), denoted as α, and request an explanation regarding its inclusion or exclusion from the computed answer set. During this phase, the user can choose from the explanation methods outlined in Section 6.2, and a minimal reason of incoherence $\mathscr{R}$ is computed accordingly. Then, for each rule of the form $debug(r) \leftarrow$ in $\mathscr{R}$, E-ASP shows the non-ground version of r to the user. Moreover, for each aggregate atom occurring in the body of r, E-ASP offers the option to perform a stepping-through explanation. Finally, for each rule of the form $support(p) \leftarrow$ in $\mathscr{R}$, E-ASP provides a message indicating that p lacks support and displays the non-ground version of the rules in $heads(\Pi, p)$.

6.3 Implementation and Use Case

This section demonstrates the explanation approach of E-ASP and presents the developed tool through a real-world use case, focusing on a simplified version of the encoding for the Chemotherapy Treatment Scheduling (CTS) problem presented by Dodaro et al. (2021).

Implementation. We developed E-ASP using JAVA and the client application JAVAFX for the backend and the GUI, respectively. A compiled version of the tool and the source code can be found at https://github.com/MarcoMochi/E-ASP. Our system follows the same approach presented in Section 6.2 to compute explanations for the presence of the atoms in an answer set. Particularly, E-ASP relies on the Clingo API to retrieve the grounded atoms (Kaminski et al., 2023), and it automatically enriches the input ASP program Π with the debugging atoms, obtaining the extended debugging program, $\Delta^*_{(\Pi,\mathscr{B})}$.

Now, to better present the use cases and the E-ASP tool, we briefly present the CTS problem and its encoding. The CTS problem involves assigning the day and starting time

for patients's chemotherapy treatments. In the scenario, the problem's input consists of registrations, each including the preference between a chair or bed and the duration (in time slots) of four phases: hospital acceptance, doctor visit, blood collection, and treatments (with a duration of 0 if a phase is not required). A valid solution to the problem schedules registrations into time slots for a given day (representing the start of the fourth phase, i.e., treatment), while ensuring that the following conditions are met: (i) each patient must be allocated either a chair or a bed; (ii) each chair or bed can be used by only one patient per time slot; (iii) if the treatment spans multiple time slots, the patient must consistently use the same chair or bed.

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

- Instances of `reg(RID,S)` represent the registrations, characterized by a patient's id (`RID`) and the preference for a seat (`S`), that can be either `"bed"` or `"chair"`.
- Instances of `duration(RID,PH,D)` represent the duration (`D`) of each phase (`PH`) for a registration (`RID`), where each phase can be either `"ph1"`, `"ph2"`, `"ph3"`, or `"ph4"`.
- Instances of `mss(DAY,TS)` represent the available days (`DAY`) and time slots (`TS`).
- Instances of `seat(ID,T)` represent the available seat, with its identifier `ID` and its type `T` that can be either `"bed"` or `"chair"`.

The output is an assignment represented by atoms of the form `x(RID,DAY,TS)` and `patient_seat_ts(ID,T,RID,DAY,TS)`. The meaning of the former is that the treatment phase of a registration with id `RID` is assigned to the day `DAY` and time slot `TS`; while the latter means that the seat with `ID` and type `T` is used by the patient `RID`, during the day `DAY` and time slot `TS`.

Encoding. The encoding for the simplified CTS problem is shown in Figure 6.1, where each line i contains a rule referred to as r_i. We assume the reader to be familiar with the input syntax of CLINGO (Gebser et al., 2016) and with the ASP-Core 2 standard (Calimeri et al., 2020b). Rule r_1 allocates registrations to a specific day and time slot. Rule r_2 computes the preparation time for each registration as the sum of the durations of the three first phases. Following this, rule r_3 guarantees that hospital acceptance (the first phase) begins after the first time slot. Subsequently, rule r_4 associates each assignment with a chair or bed for a specific day. Rules r_5 and r_6 ensure that each chair/bed is allocated to at most one patient at each time slot.

```
1 x(RID,DAY,TS) : mss(DAY,TS) :- reg(RID,_).
2 preparation_time(RID,DUR) :- reg(RID,_), DUR = #sum{D,PH:
      duration(RID,PH,D), PH != "ph4"}.
3 :- x(RID,_,TS), reg(RID,_), preparation_time(RID,T), TS-T < 1.
4 patient_seat_day(ID,T,RID,DAY) : seat(ID,T) :- x(RID,DAY,_), reg(RID,T).
5 patient_seat_ts(ID,T,RID,DAY,TS..TS+DUR-1) :-
      patient_seat_day(ID,T,RID,DAY), x(RID,DAY,TS), duration(RID,"ph4",DUR),
      DUR > 0.
6 :- #count{RID: patient_seat_ts(ID,T,RID,DAY,TS)} > 1, seat(ID,T), mss(DAY,
      TS).
```

Figure 6.1 ASP encoding for the simplified CTS problem.

Explaining Session for the CTS Problem Having presented the CTS problem and the encoding to solve the problem, we describe a mixed explaining session by considering a small input instance of the CTS problem with three patients, one day with nine different time slots for the schedule, and the availability of one bed and two chairs, corresponding to the following set of input facts:

```
reg("pat1","bed"). reg("pat2","chair"). reg("pat3","chair"). mss(1,1..9).
duration("pat1","ph4",5). duration("pat1","ph1",2).
duration("pat2","ph4",3). duration("pat2","ph3",1).
duration("pat2","ph1",2). duration("pat3","ph4",4).
duration("pat3","ph3",2). duration("pat3","ph1",2).
seat("bed1","bed"). seat("chair1","chair"). seat("chair2","chair").
```

To use E-ASP for analyzing a solution of such instance with the encoding in Figure 6.1, the user should open the file (or write a new one) by clicking on the *File* menu. Figure 6.2 shows the first screen of the tool with the encoding of the simplified CTS problem. E-ASP allows the user to select the type of explanations (i.e., rule-based, literal-based, or mixed) by ticking the corresponding boxes; moreover, it can list more than one answer set so that the user can select which one to investigate. Let us assume that the user selects an answer set comprising the following "output" atoms:

```
x("pat1",1,9) x("pat2",1,9) x("pat3",1,6)
patient_seat_ts("bed1","pat1",1,9..13,"bed")
patient_seat_ts("chair2","pat2",1,9..11,"chair")
patient_seat_ts("chair1","pat3",1,6..9,"chair")
```

The obtained answer is then presented as a set of true or false literals that the user can decide to explain, as illustrated in Figure 6.3.

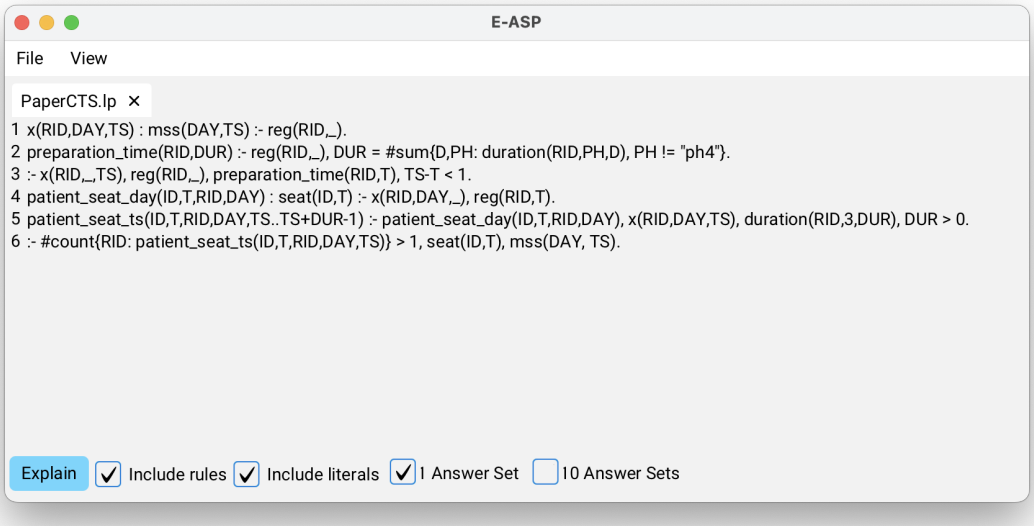

Figure 6.2 Main screen of E-ASP, where it is possible to open or write an encoding, select how many answer sets compute and the type of explanations.

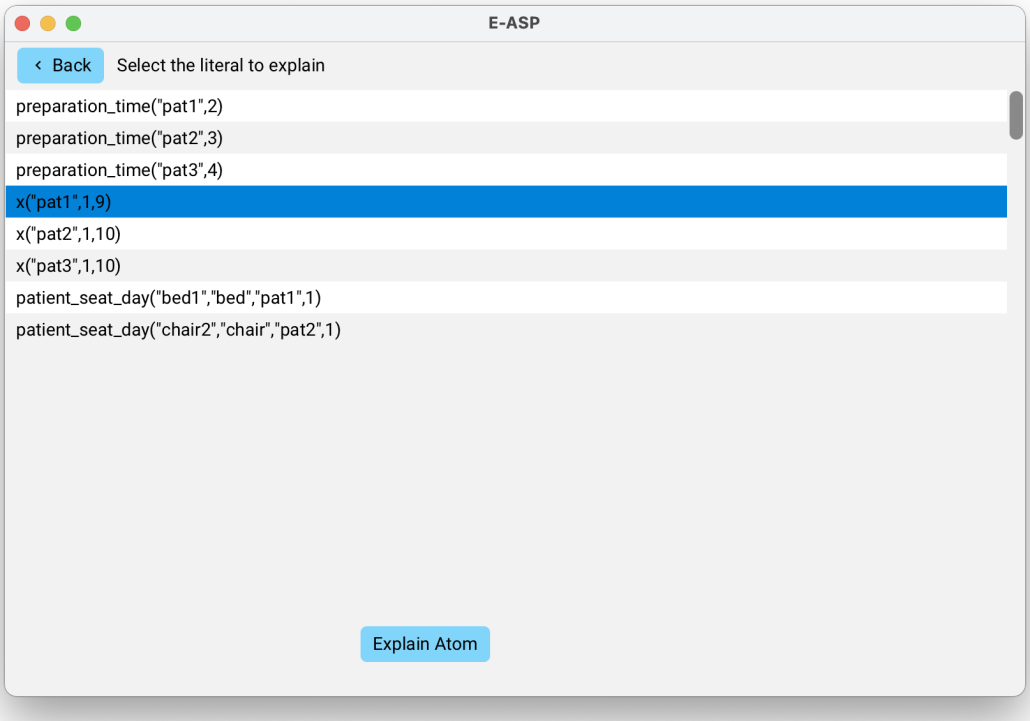

Figure 6.3 Selection screen. From this screen, the user can decide which atom to explain.

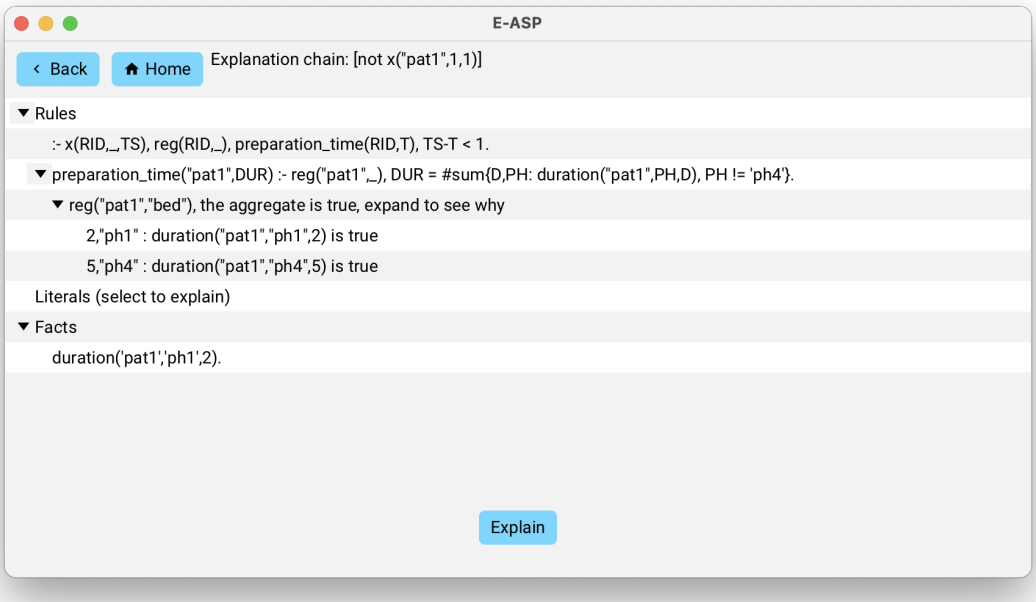

Figure 6.4 Explanation screen. From this screen, it is possible to see the set of rules, facts, and literals explaining an atom to be true/false. The aggregates can be expanded to show the set of atoms causing their truth value.

As an example, the user might ask why the atom `x("pat1",1,9)` is in the answer set, meaning why the patient `"pat1"` is assigned to the time slot `"9"`. E-ASP would return the rule r_1 together with the atom `reg("pat1","bed")` as an explanation, indicating that there is a registration for `"pat1"` and the rule r_1 requires the registration to be assigned to a time slot. Subsequently, the user might ask why the patient was assigned to a specific resource, e.g., a bed. In this case, s(he) would select `patient_seat_day("bed1","bed","pat1",1)` as the literal to explain and E-ASP would return the rules r_1 and r_4 along with the atom `x("pat1",bed)`. We can interpret this as: (i) there is a registration requiring a bed (atom `x("pat1",bed)`); (ii) every registration must be assigned to a time slot (rule r_1); (iii) every assignment must be assigned to a resource, matching the required one (rule r_4). Lastly, the user asks why the patient was not assigned to time slot 1, i.e., why `x("pat1",1,1)` is not in the answer set. In this case, E-ASP would return the rules r_2 and r_3 along with the atoms `reg("pat1",1)` and `duration("pat1","ph1",2)`. That is, the preparation time of the registration for the patient `"pat1"` requires two time slots (i.e., the duration of the first phase), computed by r_2 using `duration("pat1","ph1",2)`. Then, starting at the time slot 1 would violate the requirements expressed by the constraint r_3. Figure 6.4 shows how it is possible

to expand the aggregate in rule r_2, following the principle described in Section 6.2, to better understand how such value was obtained by the solver. Moreover, if one or more literals are in the list, the user can decide to further analyze the answer set and explain the literal, creating a chain of explanations. In this way, the previously explained literal is not considered in the new explanation.

6.4 Conclusions

This chapter presented a novel tool designed to provide explanations for ASP programs. Building upon techniques from DWASP, E-ASP offers explanations for both the presence and absence of atoms in answer sets, offering users insights into program properties and behaviors. The tool can be customized to user requirements since it supports both rule-based and literal-based explanations. Additionally, E-ASP is able to provide explanations over #*count* and #*sum* aggregates via a stepping-through approach, enhancing its utility in complex scenarios. Finally, we would like to point out that E-ASP is open-source and available at https://github.com/MarcoMochi/E-ASP.

Chapter 7

CNL2ASP: Controlled Natural Language to ASP

Despite the success of ASP, and in general of KR formalisms, it may be preferable for certain types of users to use a higher-level language that is closer to natural language for specifying ASP programs. For this reason, in the last decades, a number of attempts to convert English sentences expressed in a controlled natural language (CNL) into a KR formalism emerged Clark et al. (2005); Fuchs (2005). In the context of ASP, a CNL has been used for solving logic puzzles Baral and Dzifcak (2012), and for answering biomedical queries Erdem and Yeniterzi (2009).

Arguably, using a CNL may offer several practical advantages:

1. CNL specifications are usually more readable.
2. Writing CNL specifications is expected to be easier and faster than encoding knowledge in a formal KR language, like ASP. The generated ASP encodings can be used as a starting point for further optimization made by ASP experts.
3. CNL specifications tend to be more adaptable to changes compared to ASP encodings, e.g., adding a term in an ASP atom requires the substitution of all occurrences of the atoms, whereas in a CNL this should have almost no impact.
4. CNL specifications can be used as a basis for deploying richer language processing.

In this chapter, we present a tool called CNL2ASP that automatically translates sentences expressed in a CNL language into ASP rules. The CNL supported by CNL2ASP is inspired by the Semantics of Business Vocabulary and Business Rules (SBVR) Bajwa et al. (2011);

The Business Rules Group (2000), which is a standard proposed by the Object Management Group to formally describe complex entities, e.g., the ones related to a business, using natural language, and by PENG^{ASP}, a CNL defined by Schwitter (2018).

The tool was developed around four different types of use cases: *(i)* allowing users that have limited experience on ASP to specify ASP programs; *(ii)* enabling ASP experts to quickly prototype intuitive encodings, which can later be optimized; *(iii)* improving the readability of ASP programs since there is a one-to-one mapping between ASP rules and English specifications; and *(iv)* providing a modern tool that serves as a foundation for writing specifications in natural language. To demonstrate the capabilities of our CNL, we present several synthetic and real-world use cases, showing how the CNL can effectively solve complex combinatorial problems. Moreover, we performed an experimental analysis on the real-world use cases comparing the performance of the ASP encoding generated by CNL2ASP with the one created by ASP practitioners, showing that our tool can, in general, obtain good performance. We also carried out a preliminary analysis to evaluate the usability and readability of the proposed CNL. Finally, we mention that the implementation of the tool presented here is open source and publicly available at https://github.com/dodaro/cnl2asp.

7.1 CNL2ASP

This section deals with the specification of CNL language and with the implementation of the tool CNL2ASP, whose architecture is depicted in Figure 7.1. The tool takes as input a file containing a list of statements written in a CNL and produces as output a file containing a set of ASP rules. A specification written in this CNL is made of propositions, the structure of which is defined by clauses, linked by connectives, that are used to express concepts, to query them for information or to express conditions on them. Concepts in a proposition define the application domain, i.e., they describe entities that are used as subjects of other propositions. The combination of clauses that produces a proposition defines its type, that is used to understand what the proposition is supposed to mean and how that meaning can be translated into ASP rules and facts.

CNL2ASP is made of three main components, namely the *Parser*, the *Concepts Data Structures*, and the *ASP Rewriter*. In particular, each CNL proposition in the input file is processed by the Parser, whose role is *(i)* to create appropriate data structures for concepts to be stored in the Concept Data Structures, and *(ii)* to tokenize the CNL statements and send the result to the ASP Rewriter component. In more details, the Parser interprets three subtypes of CNL propositions, namely explicit definition propositions, implicit definition

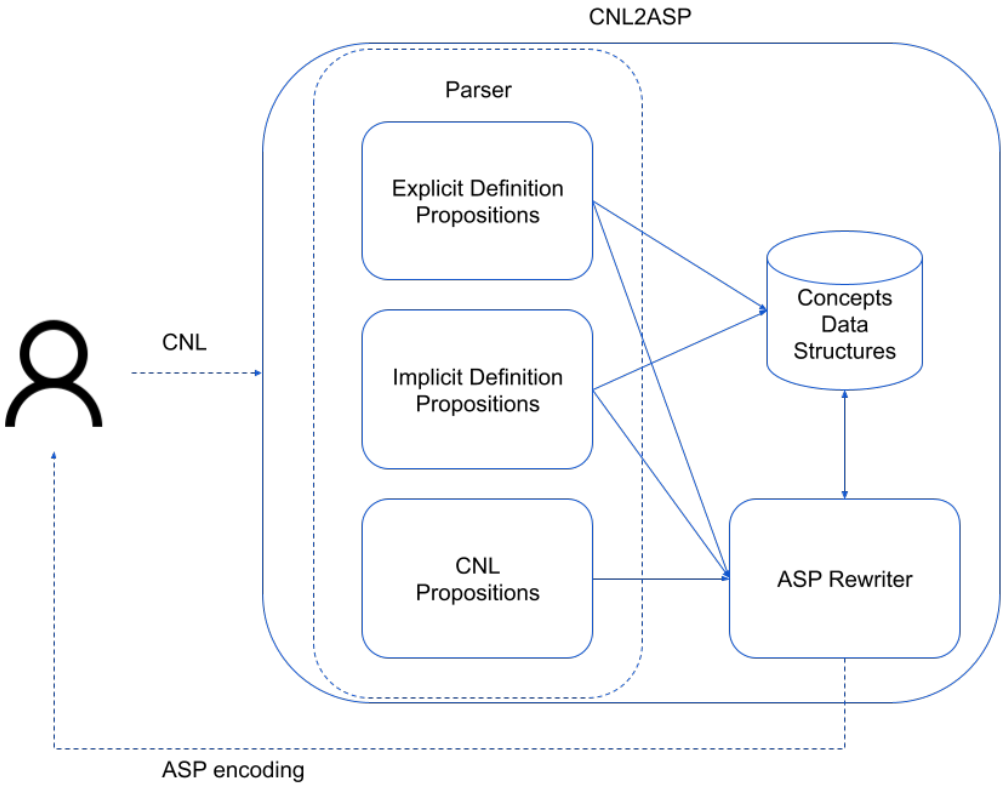

Figure 7.1 Architecture of the tool CNL2ASP.

propositions, and (standard) CNL propositions. In particular, the starting production rule is the following:

```
start ⟶ (explicit_definition_proposition | implicit_definition_proposition
    | standard_proposition)+
```

The first two types of propositions are used to define the concepts, where in our context a concept is a thing, a place, a person or an object that is used to model entities of the application domain of the CNL. Standard CNL propositions are sentences describing the rules of the application domain. As an example, consider the application domain of describing the rights and the obligations of a customer of an online store. In this context, concepts can be the customer, the company, the product, and so on, whereas CNL propositions are sentences stating what actions customers and companies can/cannot do. It is important to highlight that, in our tool, concepts are exclusively defined by their names. Consequently, taking the earlier example into account, there exists only a single concept for customer, company, product, and so forth. After all the sentences have been processed by the Parser, they are sorted as

follows: (explicit and implicit) definition propositions are processed before (standard) CNL propositions. Among the CNL propositions, the ASP Rewriter first processes the ones that are related to choice and disjunctive rules since they can also define new concepts in the data structures, and then processes strong and weak constraints. For each proposition, the ASP Rewriter first initializes the ASP atoms, then creates aggregates, arithmetic operations and comparisons, and further it merges all of them to create the head and the body of the ASP rules. Finally, after all ASP rules are created, they are stored in a output file that is returned to the user. In the following sections we first describe the different propositions accepted by the Parser along with their grammar and their translation as ASP rules (Sections 7.1, 7.1, and 7.2), and then we report a brief description of the usage of the tool.

Explicit propositions Explicit definition propositions are used to define the concepts occurring in the domain application, and they are used to create data structures which are later on used by the ASP Rewriter to produce ASP rules. In more details, the production rule of explicit definition propositions is the following:

```
explicit_definition_proposition ⟶ (domain_definition |
    temporal_concept_definition)CNL_END_OF_LINE
```

where each proposition is terminated by `CNL_END_OF_LINE` (in our case, a dot). In particular, an explicit definition proposition can be either a `domain_definition`, used to define all the entities of the problem and their structure; or a `temporal_concept_definition`, used to define only temporal elements, as days or timeslots.

Domain definition Domain definitions start with a subject optionally followed by the sentence `"is identified by"` and the definition of the keys, i.e., the parameters that uniquely represent the entity and then, also optionally, a sentence used to express the other parameters. The production rule is the following:

```
domain_definition: ("A " | "An ")? subject_name ("is identified by"
    atom_key)? ", and"? ("has" parameter ((","|", and") parameter)*)?
```

Domain definitions are not directly translated as ASP rules, instead they are used to add elements in the data structures. All properties can be later on used in propositions to refer specific properties of a concept. By default, if no property is referred to by a sentence, then the identifier is used.

The following sentences are examples of domain definitions:

```
A movie is identified by an id, and has a title, a director, and a year.
```

```
A director is identified by a name.
A topMovie is identified by an id.
A scoreAssignment is identified by a movie, and by a value.
```

Note that `scoreAssignment` is identified by a movie, which is a concept that is created by the user. This has an impact on its translation into ASP, as shown in Section 7.2.

Temporal concept Temporal concept definitions start with a subject followed by the sentence `"is a temporal concept expressed in"`, then by the temporal type that can be minutes, days or steps. The preposition continues with a sentence used to express the temporal range and, finally, it can be closed with a sentence used to specify the length of each temporal step. The production rule is the following:

```
temporal_concept_definition: ("A " | "An ") subject_name "is a temporal
    concept expressed in" CNL_TEMPORAL_TYPE "ranging from"
    temporal_range_start "to" temporal_range_end ("with a length of"
    CNL_NUMBER ("minutes" | "days"))?
```

Temporal concepts enable the possibility to refer to them using special words like after, before, and so on.

An example of a temporal definition is the following sentence:

```
A timeslot is a temporal concept expressed in minutes ranging from 07:00 AM
    to 09:00 AM with a length of 30 minutes.
```

Such concepts are conveniently translated as ASP facts by the ASP Rewriter as follows:

```
timeslot(1,"07:00").
timeslot(2,"07:30").
timeslot(3,"08:00").
timeslot(4,"08:30").
```

and the association between the used number and the corresponding time slot is stored into a dedicated data structure, so that when a user refers to a particular time slot (e.g., 07:30 AM), it is automatically encoded as the corresponding ASP atom (e.g., `timeslot(2, "07:30")`). The second term is a string representing the time slot, which is never used in the generated ASP encoding, but that can be useful when provided as output to the user.

Implicit propositions Implicit definition propositions are used to define concepts that can, then, be used by other propositions. These definitions express the *signature* of the concept indicated by the subject of the proposition, carrying information regarding the concept in

the definition that our tool can use later on in the specification whenever the same concept is used. Differently from explicit definition propositions, users do not have to specify the properties of the concepts, because they are inferred from the sentence. In more details, the production rule of implicit definition propositions is the following:

```
1 implicit_definition_proposition ⟶ (constant_definition_clause |
    compounded_clause | enumerative_definition_clause)CNL_END_OF_LINE
```

In particular, an implicit definition proposition can be a `constant_definition_clause`, used to specify constants; or a `compounded_clause`, used to define elements using lists and ranges; or a `enumerative_definition_clause`, used to define elements one at a time, optionally closing the proposition with a `when` clause, defining a condition in which the element is defined (e.g., `X is true when Y is true`), and with a `where` clause.

Constant Constant definitions are used to introduce constants to be used later on in the specification.

The following sentences are examples of constant definitions:

```
1 minKelvinTemperature is a constant equal to 0.
2 acceptableTemperature is a constant.
```

As we can see from the proposition at line 1, the constant 0 is introduced with a `equal to` clause, and it is bound to the subject of the proposition. Instead, in the proposition at line 2, we are defining a constant without assigning it a value, which can be later on assigned by the user (e.g., the ASP system CLINGO Gebser et al. (2016) supports the option `--const` to specify constants). In the case of constant definitions, there are no translations to ASP available, because they are instead stored in the data structures and substituted in the resulting program when the subject of the definition is used.

Compound Compound definitions are used to introduce a set of related concepts all at once, by making use of either ranges of numbers or lists. The following sentences are examples of compound definitions:

```
1 A ColdTemperature goes from minKelvinTemperature to acceptableTemperature.
2 A day goes from 1 to 365.
3 A drink is one of alcoholic, nonalcoholic and has color that is equal to
    respectively blue, yellow.
```

Propositions at lines 1 and 2 are examples of definitions using a range, identified by the construct `goes from/to`. In particular, proposition at line 1 uses the constants defined in Section 7.1.

Proposition at line 3 is an example of a definition of a drink using lists with a `one of` clause, where one can also specify additional attributes for each element of the list in a positional way using a `respectively` clause, and a list with the same number of elements of the list enumerating all the possible values that the subject of the proposition can have.

The corresponding ASP code, in this case, is quite straightforward and is depicted below:

```
1 coldtemperature(0..acceptableTemperature).
2 day(1..365).
3 drink(1, "alcoholic", "blue").
4 drink(2, "nonalcoholic", "yellow").
```

First of all, note that constant `minKelvinTemperature` is directly replaced by its value (i.e., 0), whereas constant `acceptableTemperature` is left as is. List elements defined in proposition at line 3 carry on their position number with them, which turns out to be handy as a basic way to encode precedence relationships when the subject is not a number. Note that, `day(1..365).` denotes the set of facts `day(1).` ... `day(365).`

Enumerative Enumerative definitions are used to introduce a property for a single concept or a relationship among a set of concepts. The peculiarity of this kind of propositions lies in the different translations into ASP as the clauses used within them change.

The following sentences are examples of enumerative definitions:

```
1 John is a waiter.
2 1 is a pub.
3 Alice is a patron.
4 Waiter John works in pub 1.
5 Waiter John serves a drink alcoholic.
6 Pub 1 is close to pub 2 and pub X, where X is one of 3,4.
7 Waiter W is working when waiter W serves a drink.
```

Such propositions show the construction to define relationships or properties related to a particular subject. In particular, propositions from line 1 to line 3 are used to define the concepts of waiter, pub, and patron, respectively, whereas propositions at lines 4 and 5 define concepts related to work in and to serve, respectively. Proposition at line 6 illustrates another feature of our CNL, i.e., `where` clauses, that are used in the example to define the values that

the variable `X` can take. Proposition at line 7 is a *conditional* definition, identified by a `when` clause.

The translations in ASP of these examples are the following:

```
1 waiter("john").
2 pub(1).
3 patron("alice").
4 work_in("john", 1).
5 serve("john", "alcoholic").
6 close_to(1, 2, 3). close_to(1, 2, 4).
7 working(W) :- serve(W,_).
```

Propositions from line 1 to line 6 always hold true, therefore they are used by the ASP Rewriter to produce the corresponding ASP facts. In particular, proposition at line 6 is translated in a similar manner to compound definitions with lists. Instead, proposition at line 7 holds true only if the statement introduced by the `when` clause is true, hence it is translated into an ASP rule, where the body of the rule is the element inside the `when` clause.

Note that, in these examples, `W` is considered as a variable, whereas `John` and `Alice` are treated as ASP strings. This is because CNL2ASP assumes that every object starting with an upper case letter and containing only upper case letters, numbers or symbols is considered as a variable, while other objects are strings or numbers (e.g., `MY_VARIABLE` is considered as a variable, whereas `My_String` is considered as a string).

7.2 CNL propositions

Explicit and implicit definition propositions are used to define the concepts of the domain application, whose specifications are instead described by (standard) CNL propositions. The production rule of standard CNL propositions is the following:

```
1 standard_proposition ⟶ (
2         whenever_then_clause_proposition |
3         fact_proposition |
4         quantified_choice_proposition |
5         negative_strong_constraint_proposition |
6         positive_strong_constraint_proposition |
7         weak_constraint_proposition |
8         )CNL_END_OF_LINE
```

Therefore, the CNL considers several types of propositions, which are described in the following.

Whenever/then clauses `Whenever/then` clauses are used to describe actions occurring when a condition is fulfilled. In more details, the production rule is the following:

```
whenever_then_clause ⟶ (whenever_clause ","?)+ then_clause
```

They start with `whenever` clauses, i.e., sentences specifying conditions, followed by a `then` clause, that is a sentence used to express the actions that must or can hold when the `whenever` clauses are fulfilled.

The following sentences are examples of `whenever/then` clauses:

```
Whenever there is a movie with director equal to spielberg, with id X then
    we must have a topmovie with id X.
Whenever there is a director with name X different from spielberg then we
    can have at most 1 topmovie with id I such that there is a movie with
    director X, and with id I.
Whenever there is a movie with id I, with director equal to nolan then we
    can have a scoreAssignment with movie I, and with value equal to 3 or a
    scoreAssignment with movie I, and with value equal to 2.
```

Such propositions are encoded in ASP as follows:

```
topmovie(X) :- movie(X,_,"spielberg",_).
0 <= {topmovie(I):movie(I,_,X,_)} <= 1 :- director(X), X != "spielberg".
scoreassignment(movie(I),3) | scoreassignment(movie(I),2) :-
    movie(I,_,"nolan",_).
```

In particular, the form `whenever/then` followed by the word `must` is translated as a normal rule by the ASP Rewriter, whereas if it is followed by the word `can` then it can be translated as a choice rule or as a disjunctive rule depending on whether the CNL sentence contains the keyword `or`. Here, we want also to emphasize the fact that the first term of `scoreAssignment` is of the form `movie(I)` since it is defined to be of the type movie.

Fact proposition Fact propositions are used to define the facts of the problem. Differently from implicit definition propositions, here no new concepts are introduced, meaning that all concepts used here must be explicitly defined. An example of a fact proposition is the following sentence:

```
There is a movie with id equal to 1, with director equal to spielberg, with
    title equal to jurassicPark, with year equal to 1993.
```

This sentence is translated as:

```
movie(1,"jurassicPark","spielberg",1993).
```

It is worth mentioning that the order of the elements listed in the sentence has no impact on its translation, since the properties of the concepts are explicitly defined. Therefore, the specifications listed below all produce the same ASP output.

```
There is a movie with id equal to 1, with director equal to spielberg, with
    year equal to 1993, with title equal to jurassicPark.
There is a movie with director equal to spielberg, with year equal to 1993,
    with id equal to 1, with title equal to jurassicPark.
```

Quantified choice propositions Quantified choice propositions are used to define relationships or properties that *can* be true for a given set of selected concepts following a choice. Also these propositions define a *signature* for the concept upon which the choice has to be made. Quantified propositions are always introduced by the `every` quantifier and, since they express possibilities, always contain a `can` clause. In more details, the production rule is the following:

```
quantified_choice_proposition ⟶ quantifier subject_clause "can"
    CNL_COPULA? (verb_name | verb_name_with_preposition)
    (quantified_exact_quantity_clause | quantified_range_clause)?
    (quantified_object_clause | disjunctive_clause)? foreach_clause?
```

Thus, they start with a quantifier, and are always followed by a subject and a verb, optionally connected by a `CNL_COPULA` (e.g., `is, is a, is an, ...`) and then, also optionally, either by a sentence of type `quantified_exact_quantity_clause`, used to express the quantity in exact terms (e.g., `exactly 1`); or by a sentence of type `quantified_range_clause`, used to express it using a range (e.g., `between 1 and 2`). The proposition can be closed either with an object clause, i.e., a sentence used to express an object for the proposition, in a `subject verb object` fashion, or with a disjunctive clause; and, finally, a `foreach` clause, i.e., a sentence used to express additional objects for which any possible value can be tried.

The following sentences are examples of quantified choice propositions:

```
Every patron can drink in exactly 1 pub for each day.
Every waiter can serve a drink.
```

```
3 Every movie with id I can have a scoreAssignment with movie I, and with
      value equal to 1 or a scoreAssignment with movie I, and with value equal
      to 2, or a scoreAssignment with movie I, and with value equal to 3.
```

Proposition at line 1 shows how one can express an exact number of choices that can be made for the concept expressed by the subject, and also how other concepts can be used in tandem with the subject to create a sort of cartesian product of choices, using a `for each` clause. These last constructions are optional, as shown in proposition at line 2. Proposition at line 3, instead, shows an example of a disjunctive clause. Their full translations into ASP is shown below:

```
1 <= {drink_in(_X1,_X2,_X3):pub(_X3)} <= 1 :- patron(_X1), day(_X2).
{serve(_X1,_X2):drink(_,_X2,_)} :- waiter(_X1).
scoreassignment(movie(I),1) | scoreassignment(movie(I),2) |
    scoreassignment(movie(I),3) :- movie(I,_,_,_).
```

The first two translations use choice rules (possibly with bounds), that are the ASP constructs that make it possible to represent propositions of this type, whereas the third one uses a disjunctive rule. Note that the first two rules also employ generated variables (starting with symbol _) that are used wherever two atoms have to be bound and no variable to use has been found in the specification given in input. This feature enables the specification writer to avoid cluttering the document with unnecessary variables, as can be seen throughout the propositions, with the only limitation that anaphoras have to be expressed explicitly by providing the correct variable.

Negative and positive strong constraints Negative and positive strong constraint propositions are used to define assertions that *must* be true for a given set of selected concepts. This kind of propositions does not introduce new *signatures* but, on the contrary, they consume other signatures that were previously defined, meaning that the concepts used inside such constraints have to be defined before they are used. A strong constraint can represent either a prohibition (sentences starting with `It is prohibited`) or a requirement (sentences starting with `It is required`). After specifying if the strong constraint is a prohibition or a requirement, then a user can add *simple* clauses, that are made of a subject, a verb, and related object clauses; *aggregate* clauses, either in active or passive form, that define an aggregation of the set of concepts that satisfy the statement in their body with the operator that was specified (`number`, `total`, `lowest`, `highest`); or other complex clauses as shown below.

In more details, the production rules of strong constraints are the following:

```
1 negative_strong_constraint_clause ⟶ "it is prohibited that" (simple_clause
    ("and also" simple_clause)* (where_clause)? ("," (whenever_clause
    ","?)+)? | aggregate_clause comparison_clause (where_clause)? ("," 
    (whenever_clause ","?)+)? | when_then_clause (where_clause)? |
    quantified_constraint (where_clause)? | condition_clause ","
    (whenever_clause ","?)+ | temporal_constraint "," (whenever_clause
    ","?)+)
2 positive_strong_constraint_proposition ⟶ "it is required that"
    (simple_clause "," (whenever_clause ","?)+ | aggregate_clause
    comparison_clause (where_clause)? ("," (whenever_clause ","?)+)? |
    when_then_clause (where_clause)? | quantified_constraint (where_clause)?
    | condition_clause "," (whenever_clause ","?)+ | temporal_constraint ","
    (whenever_clause ","?)+)
```

It is possible to observe that they start with the sentence `it is prohibited that` or with the sentence `it is required that` and are always followed by a simple clause, i.e., a sentence of the form `subject verb object`; by an aggregate clause, a sentence expressing a form of aggregations (e.g., `the number of`); by a whenever clause, described in Section 7.2; by a quantified constraint, used to specify clauses with quantifiers as `every` or `any`; or by a temporal constraint, used to specify constraints on temporal concepts as `after 11:00 AM` or `before 11:00 AM`. After simple clauses, aggregate clauses, and quantified constraints, additional sentences can be added, which can be of the type `where_clause`, used to specify conditions; or of the type `comparison_clause`, used to specify comparison between elements (e.g., `X is equal to 1`).

The following sentences are examples of negative and positive strong constraints:

```
1 It is prohibited that waiter W1 work in pub P1 and also waiter W2 work in
    pub P1, where W1 is different from W2.
2 It is prohibited that X is equal to Y, whenever there is a movie with id X,
    and with year equal to 1964, whenever there is a topMovie with id Y.
3 It is prohibited that the lowest value of a scoreAssignment with movie X is
    equal to 1, whenever there is a topMovie with id X.
4 It is required that the total value of a scoreAssignment with movie X is
    equal to 10, such that there is a topMovie with id X.
5 It is required that the number of pub where a waiter work in is less than 2.
6 It is required that when waiter X works in pub P1 then waiter X does not
    work in pub P2, where P1 is different from P2.
```

```
7 It is required that V is equal to 3, whenever there is a movie with id I,
    and with director equal to spielberg, whenever there is a
    scoreAssignment with movie I, and with value V.
8 It is required that every waiter is payed.
```

Proposition at line 1 shows a practical example of the combination of several simple clauses, and the feature enabled by `where` clauses, that makes it possible to express comparisons between variables. Proposition at line 2 shows an example of `whenever` clause. Propositions at line 3, at line 4, and at line 5 show examples of aggregation expressing conditions on the minimum value, on the sum of values, and on the number of occurrences, respectively. Proposition at line 6 shows a `when/then` clause. Proposition at line 7 shows an example of `whenever` clause in the context of positive strong constraints. Lastly, proposition at line 8 is an example of how to specify a requirement that must hold for all the elements present in a particular set of concepts. Such propositions are encoded as ASP rules as follows:

```
1 :- work_in(W1,P1), work_in(W2,P1), W1 != W2.
2 :- movie(X,_,_,1964), topmovie(Y), X = Y.
3 :- topmovie(X), #min{_X1: scoreassignment(movie(X),_X1)} = 1.
4 :- #sum{_X1: scoreassignment(movie(X),_X1), topmovie(X)} != 10.
5 :- waiter(_X1), #count{_X2: work_in(_X1,_X2)} >= 3.
6 :- work_in(X,P1), work_in(X,P2), P1 != P2.
7 :- movie(I,_,"spielberg",_), scoreassignment(movie(I),V), V != 3.
8 :- not payed(_X1), waiter(_X1).
```

Note that their translation is quite intuitive, and positive strong constraints are translated as ASP constraints by negating the condition expressed by the sentence.

Weak constraint propositions Weak constraint propositions are used to define assertions that are *preferably* true for a given set of selected concepts. Also this type of propositions consumes *signatures* from previously defined concepts. They are always introduced by `it is preferred` and need the specification of the optimization objective (either minimization or maximization), and the level of priority of the optimization (low, medium or high). The production rule is the following:

```
1 weak_constraint_proposition ⟶ "it is preferred that"
    CNL_WEAK_OPTIMIZATION_OPERATOR? ","? weak_priority_operator ","? "that"
    (condition_operation | aggregate_clause | subject_clause CNL_COPULA
    (verb_name | verb_name_with_preposition) object_clause ","
    whenever_clause) weak_optimization_operator? (where_clause)?
```

In particular, they start with the sentence `it is preferred ... that`, and can be followed by a sentence expressing the nature of the optimization (i.e., `as much as possible` or `as little as possible`), and are always followed by a priority operator, i.e., a sentence expressing the level of relevance of the constraint with respect to other weak constraints (e.g., `"with low priority"`) and either a clause followed by a whenever clause, an aggregate clause or a condition operation, i.e., a sentence expressing operations between variables in the proposition (e.g., `the sum of X and Y`). The proposition is closed with an optimization operator, i.e., a sentence expressing the nature of the optimization (i.e., `"is minimized"` or `"is maximized"`) and an optional where clause. Note that here we have two ways for expressing the object, either in the form of as much (little) as possible at the beginning of the sentence or using is maximized (minimized) at the end of the sentence. The two ways are equivalent, but we support both of them to make sentences more natural. Moreover, sentences containing both kind of specifications are well-formed, thus they are correctly parsed even if they are in contrast (e.g., a user can specify `as much as possible` and `"is minimized"` in the same sentence). However, CNL2ASP subsequently checks if this happens and, in case, it triggers an error so that only one of the form is used.

The following sentences are examples of weak constraint propositions:

```
1 It is preferred with low priority that the number of drinks that are serve
    is maximized.
2 It is preferred as little as possible, with high priority, that V is equal
    to 1, whenever there is a scoreAssignment with movie I, and with value
    V, whenever there is a topMovie with id I.
3 It is preferred, with medium priority, that whenever there is a topMovie
    with id I, whenever there is a scoreAssignment with movie I, and with
    value V, V is maximized.
4 It is preferred, with medium priority, that the total value of a
    scoreAssignment is maximized.
```

The sentence at line 1 shows an example of a maximization over the result of a `#count` aggregate. The sentence at line 2 instead is an example of minimization using the form `as little as possible`. Then, the sentence at line 3 shows a sentence where the subject of maximization is a variable defined in `scoreAssignment`. Finally, the sentence at line 4 is an example of a `#sum` aggregate, where the result of the aggregation is subject to maximization. Translation of the propositions above are shown below:

```
1 :~ #count{_X1: serve(_,_X1)} = _X2. [-_X2@1]
2 :~ scoreassignment(movie(I),V), topmovie(I), V = 1. [1@3, I,V]
```

```
3 :~ topmovie(I), scoreassignment(movie(I),V). [-V@2, I]
4 :~ #sum{_X1: scoreassignment(movie(_),_X1)} = _X2. [-_X2@2]
```

Also in this case the translation is quite intuitive, however one should note that maximization constructs are translated using negative weights.

Having presented all the definitions, we now report the usage of the tool and its implementation. CNL2ASP has been implemented using the programming language Python, and the open-source library lark (https://github.com/lark-parser/lark) for creating the Parser, which is the only required dependence to run it. Moreover, the tool requires to use the version 3.10 (or higher) of Python. Concerning the distribution license, CNL2ASP is released under the Apache 2.0 license, which allows the user to use it also in industrial contexts. Its usage is quite intuitive since it can be used as a standalone tool by issuing the command

```
1    python3 src/main.py input_file [output_file]
```

or, as an alternative, it can be used as a library in other Python projects by simply importing it.

7.3 Synthetic use cases

In this section, we present some examples to demonstrate how the language can be used to define well-known combinatorial problems in a natural and easily understandable way. The corresponding translations into ASP are also provided. In particular, we present the use cases of the Graph coloring, Hamiltonian path, and Maximal clique problem.

We begin by presenting an encoding of the graph coloring problem using our CNL. We recall that the graph coloring problem is the problem of finding an assignment of colors to nodes in a graph such that two adjacent nodes do not share the same color.

```
1 A node goes from 1 to 3.
2 A color is one of red, green, blue.
3 Node 1 is connected to node X, where X is one of 2, 3.
4 Node 2 is connected to node X, where X is one of 1, 3.
5 Node 3 is connected to node X, where X is one of 1, 2.
6 Every node can be assigned to exactly 1 color.
7 It is required that when node X is connected to node Y then node X is not
    assigned to color C and also node Y is not assigned to color C.
```

One can notice the presence of explicit definition propositions (lines 1–5), with a ranged definition proposition (line 1) and a list definition proposition (line 2), enumerative definition

propositions with where clauses (lines 3–5), a quantified clause (line 6) and, lastly, a positive strong constraint (line 7).

The resulting ASP encoding is the following:

```
1 node(1..3).
2 color(1,"red"). color(2,"green"). color(3,"blue").
3 connected_to(1,2). connected_to(1,3).
4 connected_to(2,1). connected_to(2,3).
5 connected_to(3,1). connected_to(3,2).
6 {assigned_to(_X1,_X2): color(_,_X2)} = 1 :- node(_X1).
7 :- connected_to(X,Y), assigned_to(X,C), assigned_to(Y,C).
```

where each proposition at line i is translated as the rule(s) reported at line i (with $i = 1..7$).

The second problem we consider here is the well-known Hamiltonian path problem, which is the problem of finding a path in a graph that visits each node exactly once, starting from a given node.

```
1 A node goes from 1 to 5.
2 Node 1 is connected to node X, where X is one of 2, 3.
3 Node 2 is connected to node X, where X is one of 1, 4.
4 Node 3 is connected to node X, where X is one of 1, 4.
5 Node 4 is connected to node X, where X is one of 3, 5.
6 Node 5 is connected to node X, where X is one of 3, 4.
7 Every node X can have a path to a node connected to node X.
8 It is required that the number of nodes where node X has a path to is equal
    to 1.
9 It is required that the number of nodes that have a path to node X is equal
    to 1.
10 Node Y is reachable when node X is reachable and also node X has a path to
    node Y.
11 It is required that every node is reachable.
12 start is a constant equal to 1.
13 Node start is reachable.
```

Line 1 defines the nodes and lines from 2–6 define the connections between nodes. Then, line 7 reports a quantified proposition with an object accompanied by a verb clause, lines 8 and 9 report strong constraint propositions with aggregates, line 10 reports a conditional definition clause, line 11 reports a constraint clause with the presence of a quantifier, and

line 12 defines the constant `start`, which is subsequently used in line 13. The ASP encoding corresponding to the CNL statements is the following:

```
1  node(1..5).
2  connected_to(1,2). connected_to(1,3).
3  connected_to(2,1). connected_to(2,4).
4  connected_to(3,1). connected_to(3,4).
5  connected_to(4,3). connected_to(4,5).
6  connected_to(5,3). connected_to(5,4).
7  {path_to(X,_X1): connected_to(X,_X1)} :- node(X).
8  :- node(X), #count{_X2: path_to(X,_X2)} != 1.
9  :- node(X), #count{_X3: path_to(_X3,X)} != 1.
10 reachable(Y) :- reachable(X), path_to(X,Y).
11 :- not reachable(_X4), node(_X4).
12 reachable(1).
```

where a CNL statement at line i is represented by the rule(s) at line i with ($i = 1..11$), whereas CNL statements reported in lines 12 and 13 are encoded by the rule at line 12. As an alternative, one could also use the sentence `start is a constant`, and then use the solver options to change the starting node.

The third problem is the maximal clique problem, which is the problem of finding a clique, i.e., a subset of the nodes of a given graph where all nodes in the clique are adjacent to each other, and the cardinality of the clique is maximal.

```
1 A node goes from 1 to 5.
2 Node 1 is connected to node X, where X is one of 2, 3.
3 Node 2 is connected to node X, where X is one of 1, 3, 4, 5.
4 Node 3 is connected to node X, where X is one of 1, 2, 4, 5.
5 Node 4 is connected to node X, where X is one of 2, 3, 5.
6 Node 5 is connected to node X, where X is one of 2, 3, 4.
7 Every node can be chosen.
8 It is required that when node X is not connected to node Y then node X is
    not chosen and also node Y is not chosen, where X is different from Y.
9 It is preferred with high priority that the number of nodes that are chosen
    is maximized.
```

where statements from line 1 to line 6 define the input graph. Then, line 7 reports a quantified proposition with no object, line 8 contains a strong constraint proposition with a comparison on the variables used inside it, and line 9 reports a weak constraint expressing a maximization

preference on the highest priority level. The resulting ASP encoding is reported in the following:

```
node(1..5).
connected_to(1,2). connected_to(1,3).
connected_to(2,1). connected_to(2,3). connected_to(2,4). connected_to(2,5).
connected_to(3,1). connected_to(3,2). connected_to(3,4). connected_to(3,5).
connected_to(4,2). connected_to(4,3). connected_to(4,5).
connected_to(5,2). connected_to(5,3). connected_to(5,4).
{chosen(_X1)} :- node(_X1).
:- not connected_to(X,Y), chosen(X), chosen(Y), X != Y.
:~ #count{_X1: chosen(_X1)} = _X2. [-_X2@1]
```

where each CNL proposition at line i is translated as the rule(s) reported at line i (with $i = 1..9$).

7.4 Real-world use cases

In this section, we show the usage of the CNL specifications to encode three real-world problems which we previously addressed using plain ASP encodings, namely the Nurse Scheduling Problem (NSP) (Section 7.4; Dodaro and Maratea (2017)) and the Chemotherapy Treatment Scheduling (CTS) Problem (Section 7.4; Dodaro et al. (2021)). Moreover, for each of the reported problem, we also show an empirical analysis comparing the performance of the encoding generated in an automatic way by CNL2ASP and the encoding written by human experts. The encodings were compared using the same solver, i.e., CLINGO version 5.6.1 configured with the same options used in the original papers where the problems were presented. The experiments were executed on a AMD Ryzen 5 2600 with 3.4 GHz, with time and memory limits set to 1200 seconds and 8 GB, respectively. All the encodings and code used for the translations can be found at https://github.com/dodaro/cnl2asp/tree/main/src/tests.

Nurse scheduling problem (NSP) The NSP is the problem of computing an assignment of nurses to shifts (morning, afternoon, night, or rest) in a given period of time such that the assignment satisfies a set of requirements. In particular, the NSP described in this section was originally defined by Dodaro and Maratea (2017), where authors classified the requirements as follows: *(i)* Hospital requirements, which impose the length of the shifts and that each shift is associated with a minimum and a maximum number of nurses that must be present in

the hospital; *(ii)* Nurses requirements, which impose that nurses have a limit on the minimum and maximum number of working hours during the considered period of time, and that each nurse has an adequate rest period; *(iii)* Balance requirements, which impose that the number of times a nurse can be assigned to morning, afternoon and night shifts is fixed.

The first part of our CNL specifications concerns the definition of the domain and of the input facts of the NSP, and it is reported in the following:

```
numberOfNurses is a constant.
A nurse goes from 1 to numberOfNurses.
A day goes from 1 to 365 and is made of shifts that are made of hours.
A shift is one of morning, afternoon, night, specrest, rest, vacation and
    has hours that are equal to respectively 7, 7, 10, 0, 0, 0.
maxNurseMorning is a constant.
maxNurseAfternoon is a constant.
maxNurseNight is a constant.
minNurseMorning is a constant.
minNurseAfternoon is a constant.
minNurseNight is a constant.
maxHours is a constant equal to 1692.
minHours is a constant equal to 1687.
maxDay is a constant equal to 82.
maxNight is a constant equal to 61.
minDay is a constant equal to 74.
minNight is a constant equal to 58.
balanceNurseDay is a constant equal to 78.
balanceNurseAfternoon is a constant equal to 78.
balanceNurseNight is a constant equal to 60.
```

In the definition above, we used implicit definition propositions that therefore also create the input facts of the problem. Note that the number of nurses is a constant that is specified by the user, some constants like `maxNurseMorning`, `maxNurseAfternoon`, etc. depend on the number of nurses, therefore they are also left to the user, whereas all other constants are specific to the NSP considered, therefore they are stated.

Then, the second part of our CNL specifications are used for solving the problem:

```
Every nurse can work in exactly 1 shift for each day.
It is required that the number of nurses that work in shift S for each day
    is at most M, where S is one of morning, afternoon, night and M is one
    of respectively maxNurseMorning, maxNurseAfternoon, maxNurseNight.
```

```
3 It is prohibited that the number of nurses that work in shift S for each day
    is less than M, where S is one of morning, afternoon, night and M is one
    of respectively minNurseMorning, minNurseAfternoon, minNurseNight.
4 It is prohibited that the total of hours in a day where a nurse works in is
    more than maxHours.
5 It is prohibited that the total of hours in a day where a nurse works in is
    less than minHours.
6 It is prohibited that the number of days with shift vacation where a nurse
    works in is different from 30.
7 It is prohibited that a nurse works in shift S in a day and also the next
    day works in a shift before S, where S is between morning and night.
8 It is required that the number of occurrences between each 14 days with
    shift rest where a nurse works in is at least 2.
9 It is required that when a nurse works in shift night for 2 consecutive days
    then the next day works in shift specrest.
10 It is prohibited that a nurse works in a day in shift specrest and also the
    previous 2 consecutive days does not work in shift night.
11 It is prohibited that the number of days with shift S where a nurse works in
    is more than M, where S is one of morning, afternoon, night and M is one
    of respectively maxDay, maxDay, maxNight.
12 It is prohibited that the number of days with shift S where a nurse works in
    is less than M, where S is one of morning, afternoon, night and M is one
    of respectively minDay, minDay, minNight.
13 It is preferred, with high priority, that the difference in absolute value
    between B, and the number of days with shift S where a nurse works in
    ranging between minDay and maxDay is minimized, where B is one of
    balanceNurseDay, balanceNurseAfternoon and S is one of morning,
    afternoon.
14 It is preferred, with high priority, that the difference in absolute value
    between balanceNurseNight, and the number of days with shift night where
    a nurse works in ranging between minNight and maxNight is minimized.
```

Here, it is interesting to observe that the specifications first define that a nurse can work in exactly one shift for each day leaving a free choice about the shift to assign to each nurse, and then they impose some requirements on the assigned shift. Moreover, note that in general we used negative constraints (i.e., sentences starting with `It is prohibited`), with the exception of the ones at lines 8 and 9 since we found that this formulation is more natural.

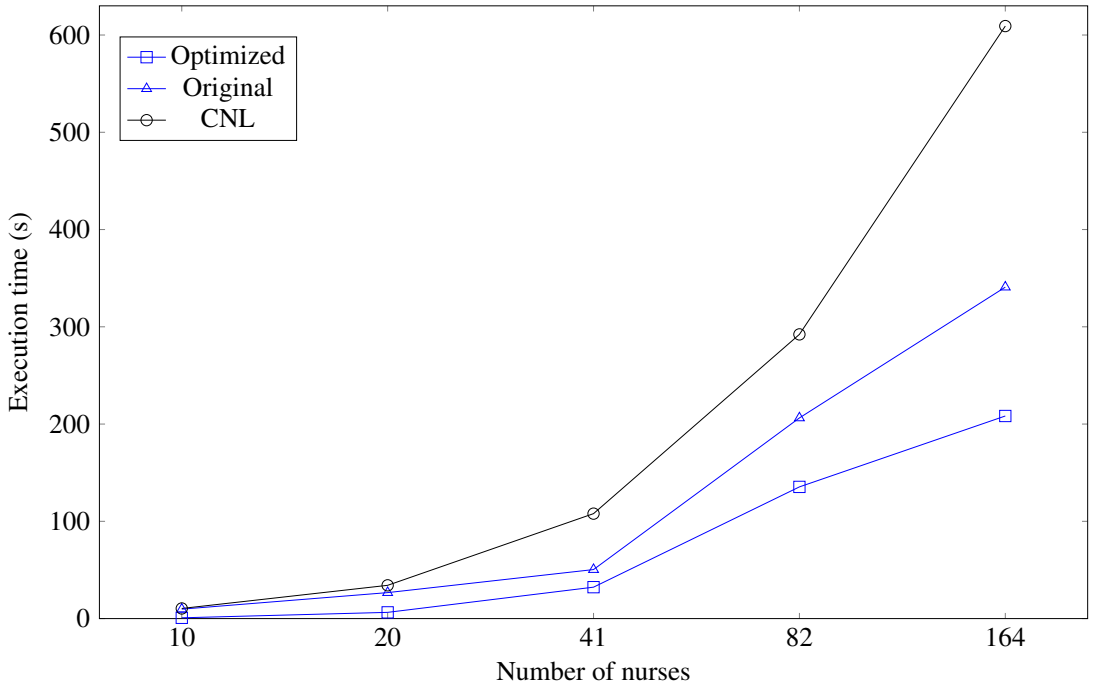

Figure 7.2 Time comparison of the performance of the original, the optimized and the CNL encodings to solve instances of the NSP.

Comparison of the performances. The encoding generated by the CNL specifications described before has been compared to the original one proposed by Dodaro and Maratea (2017) (referred to as *Original*) and with an optimized version proposed by Alviano et al. (2018) (referred to as *Optimized*). The experiment consists of five instances of the NSP with increasing number of nurses. Results are shown in Figure 7.2, where it is possible to observe that the Optimized encoding outperforms both the original and the CNL one. This result is not surprising since the Optimized encoding takes advantage of specific properties of the NSP to reduce the search space for the solver. Concerning the performance of the CNL encoding, it is possible to observe that it is approximately between 1.5 to 2 times slower than the original one. The main difference in terms of performance is due to the fact that CNL encoding generates aggregates for constraints at lines 9 and 10, which are less efficient in this context than the normal rules used in the original encoding. In this respect, tools for the automatic rewriting of aggregates, such as the one proposed by Dingess and Truszczynski (2020), can be helpful also in our context. However, it is worth mentioning that, even on the hardest instance, the generated encoding is able to terminate in approximately ten minutes.

Chemotherapy treatment scheduling (CTS) problem The CTS problem is a complex problem taking into account different constraints and resources. In this section, we consider

a simplified version of the problem described by Dodaro et al. (2021) that presented a case study based on the requirements of an Italian hospital. The idea here is to focus on the main constraints and optimization statements that are useful to show the capabilities of our tool, without considering all the variants described by Dodaro et al. (2021). In particular, the CTS problem consists of assigning a starting hour to the treatment of all the patients, and to the phases before the treatment, where the phases are *(i)* the admission to the hospital, *(ii)* the blood collection, and *(iii)* the medical check. Moreover, during the treatment, each patient must be assigned either to a bed or a chair. A proper solution to the CTS problem requires the satisfaction of a number of constraints, e.g., the starting time of the admission to the hospital must be after the opening time of the hospital, patients with long therapy must be assigned after 11:20 AM, and each bed or chair must be assigned to just one patient at a time. Finally, every patient has a preference between chairs and beds and the solution should try to maximize the number of patients assigned to the preferred resource.

The first part of our CNL specifications concerns the definition of the domain of the problem, and it is reported in the following:

```
1 A timeslot is a temporal concept expressed in minutes ranging from 07:30 AM
    to 01:30 PM with a length of 10 minutes.
2 A day is a temporal concept expressed in days ranging from 01/01/2022 to
    07/01/2022.
3 A patient is identified by an id, and has a preference.
4 A registration is identified by a patient, and by an order, and has a number
    of waiting days, a duration of the first phase, a duration of the second
    phase, a duration of the third phase, and a duration of the fourth phase.
5 A seat is identified by an id, and has a type.
6 An assignment is identified by a registration, by a day, and by a timeslot.
7 A position in is identified by a patient, by an id, by a timeslot, and by a
    day.
```

The second part of the CNL defines the CTS problem, and it is reported in the following:

```
1 Whenever there is a registration R with an order equal to 0, then R can have
    an assignment to exactly 1 day, and timeslot.
2 Whenever there is a registration R with patient P, with order OR, and with a
    number of waiting days W, whenever there is an assignment with
    registration patient P, with registration order OR-1, and with day D,
    whenever there is a day with day D+W, then we can have an assignment
    with registration R, and with day D+W to exactly 1 timeslot.
```

```
3 It is required that the sum between the duration of the first phase of the
    registration R, the duration of the second phase of the registration R,
    and the duration of the third phase of the registration R is greater
    than the timeslot of the assignment A, whenever there is a registration
    R, whenever there is an assignment A with registration R, with timeslot
    T.
4 Whenever there is a patient P, whenever there is an assignment with
    registration patient P, with timeslot T, and with day D, whenever there
    is a registration R with patient P, and with a duration of the fourth
    phase PH4 greater than 0, then P can have a position with id S, with
    timeslot T, with day D in exactly 1 seat S for PH4 timeslots.
5 It is required that the number of patient that have position in id S, day D,
    timeslot TS is less than 2, whenever there is a day D, whenever there is
    a timeslot TS, whenever there is a seat with id S.
6 It is required that the assignment A is after 11:20 AM, whenever there is a
    registration R with a duration of the fourth phase greater than 50
    timeslots, whenever there is an assignment A with registration R.
7 It is preferred as much as possible, with high priority, that a patient P
    with preference T has a position in a seat S, whenever there is a seat S
    with type T.
```

Here, we want to emphasize the simplicity of using specific constructs for temporal concepts like the time slot, as done in the sentence at line 6, where we state that an assignment is after 11:20 AM.

Comparison of the performances. The encoding generated by the CNL specifications described before has been compared to the original one proposed by Dodaro et al. (2021), referred to as *Original*. The results are presented in Figure 7.3. As expected, the original encoding is in general faster than the generated encoding. Nevertheless, the performance of generated encoding is still satisfactory, since on average it requires 32 seconds to compute a solution, with a peak of 2 minutes on the hardest instance.

7.5 Preliminary User Validation

In this section, we present an analysis conducted to assess the usability and readability of the proposed CNL. The test was conducted on August 1st, 2023, and involved 10 individuals among doctoral students and researchers from the Department of Mathematics and Computer

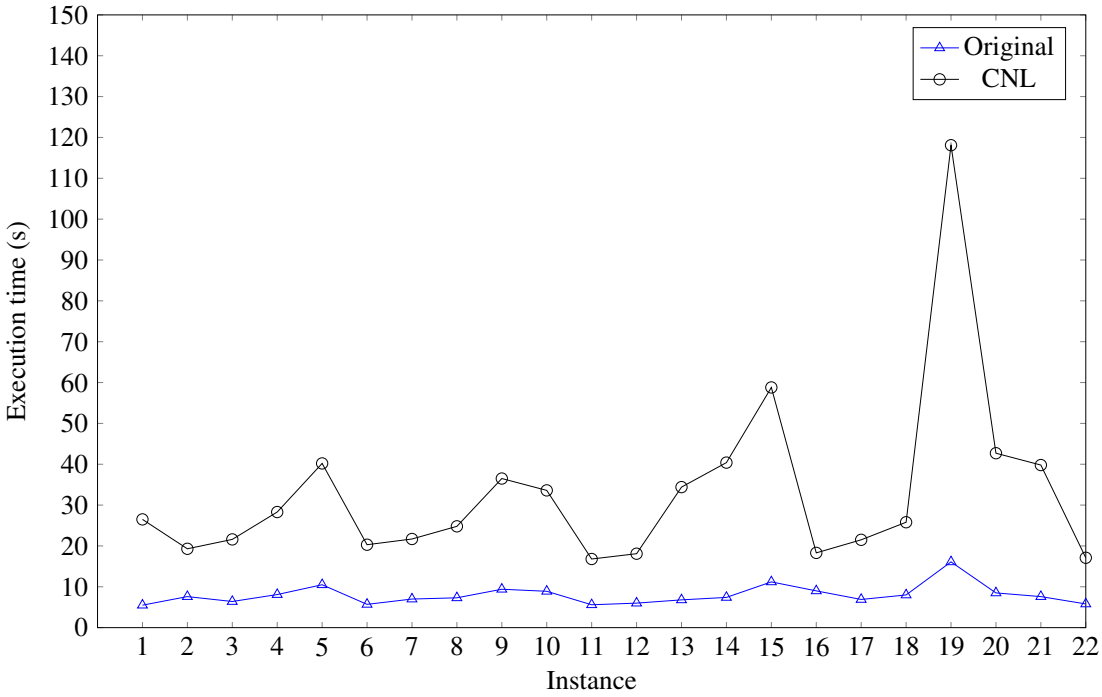

Figure 7.3 Time comparison of the performance of the original and CNL encodings to solve instances of the CTS problem.

Science at the University of Calabria. It is worth noting that 5 participants work with ASP daily and can be considered experts, while the other 5 work on different research topics. Additionally, 7 participants had attended at least one course on ASP during their studies, whereas the others attended only short seminars about ASP. The tool was not introduced beforehand, and the content of the experiment was not announced in advance. Moreover, we ensured that: *(i)* participants had no prior experience with CNL2ASP; *(ii)* the set of participants did not exclusively consist of individuals interested in tools or those with specific biases toward using programming environments; *(iii)* the set of participants included a mix of both proficient and less proficient ASP programmers, which is the expected target of users. Indeed, we believe that a limited experience on ASP or at least on declarative languages for solving combinatorial problems might be needed to proficiently use the tool.

Finally, we mention that this analysis should be considered preliminary due to the limited number of participants, and none of them had received prior training on CNL2ASP.

Concerning the usability of the tool, we designed a test in which participants were asked to solve the following problem: Given a set of n persons and m teams (assuming $n > m$), the goal is to assign persons to teams while satisfying the following conditions:

- Each person must be assigned to exactly one team (c_1).

Language	Main research area	Attended ASP course?	c_1	c_2	c_3	c_4
ASP	ASP Systems and Tools	Y	1	1	1	1
ASP	ASP Systems and Tools	Y	1	1	1	1
ASP	ASP Semantics and Theory	Y	1	1	1	0
ASP	Deep Learning	Y	0	0	0	0
ASP	Deep Learning	Y	0	0	0	0
CNL	ASP Systems and Tools	Y	1	0	1	0
CNL	ASP Systems and Tools	Y	0	0	0	0
CNL	Theoretical Computer Science	N	0	1	0	0
CNL	Deep Learning	N	0	0	0	0
CNL	Deep Learning	N	0	0	0	0

Table 7.1 Results on the usability test. Each row represents the results of an individual participant. A value of '1' indicates that the provided ASP rule/CNL specification was correct, while '0' indicates that it was incorrect.

- Each team can have a maximum of 4 persons (c_2).
- Two persons who are incompatible cannot be on the same team (c_3).
- If possible, two friends should be placed in the same team (c_4).

We divided the participants into two groups. The first group was expected to use ASP to solve the problem, while the second group was instructed to use our CNL. The test began with a brief description of the task and some basic instructions on the CNL syntax for the second group. To ensure a fair comparison, individuals who had never attended an ASP course were included in the second group.

The results are presented in Table 7.1. As expected, participants familiar with ASP were able to create an ASP program that successfully addressed the given problem. In contrast, individuals who had taken an ASP course during their studies but were not actively using ASP were unable to solve the problem.

Regarding the second group, the best performance came from a researcher who also had experience with ASP, achieving partial success in solving the problem. Interestingly, one of the researchers who did not work with ASP managed to correctly specify condition c_2. Consequently, even without prior training, 2 out of 5 participants in this group were able to specify some of the problem's conditions accurately.

Finally, to asses the readability of the tool, we designed a test in which participants were required to determine the truth or falsity of the following statements:

ASP	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	Number of correct answers
ASP	0	1	1	0	1	0	0	0	1	0	4
ASP	0	1	1	0	1	0	0	1	1	0	5
ASP	0	1	1	1	0	0	0	0	0	0	3
ASP	1	1	1	1	1	1	0	1	1	0	8
ASP	1	1	1	1	1	1	0	1	1	0	8
CNL	1	1	1	0	1	0	0	1	1	0	6
CNL	1	1	1	1	1	1	1	1	1	1	10
CNL	1	0	1	1	1	1	1	1	0	1	8
CNL	1	1	1	1	1	0	0	1	1	0	7
CNL	0	0	0	1	1	1	0	0	0	0	3

Table 7.2 Results on the readability test. Each row represents the results of an individual participant. A '1' indicates that the participant correctly identified the truth or falsity of the corresponding statement, while '0' denotes an incorrect or an empty response.

1. After two consecutive nights there is a special rest day.
2. Each nurse has at least 2 rest days every two weeks.
3. Each nurse has exactly 30 days of holidays.
4. A nurse can work at most two consecutive nights.
5. Each nurse has at most 30 days of holidays.
6. A nurse can work at most three consecutive nights.
7. A special rest day must be provided when a nurse is in vacation.
8. Each nurse can be assigned to at most “maxNight" nights shift during the whole year.
9. Each nurse can be assigned to at least “minNight" nights shift during the whole year.
10. Each nurse should be assigned to exactly “balanceNurseNight" nights shift during the whole year.

Subsequently, we grouped the participants in the same manner as in the usability experiment. The first group was provided with the ASP encoding for the Nurse Scheduling Problem (NSP) as described by Dodaro and Maratea (2017). The second group received the CNL specifications described in Section 7.4. To alleviate social pressure, we requested that

participants remain anonymous during this test. The fifth statement (s_5) was contested as ambiguous, as it can be interpreted as both true and false. Therefore, we assigned a score of 1 for both true and false responses and 0 if the answer was left blank. Results are reported in Table 7.2. On average, participants in the CNL group obtained a score of 6.8 with a peak of 10, whereas participants in ASP group obtained a score of 5.6 with a peak of 8.

7.6 Conclusions

To conclude, in this chapter, we presented a tool to automatically translate English sentences expressed in a specific Controlled Natural Language (CNL) to ASP. We defined and implemented a CNL designed for solving complex combinatorial problems, this tool supports the main features of the ASP language, including disjunctive and choice rules, aggregates, and weak constraints (Section 7.1). We then presented several use cases on well-known, synthetic domains (Section 7.3), as well as on real-world problems described in the literature (Section 7.4). Concerning the latter, we also provided the results of an experimental analysis comparing the performance of the generated encodings with the ones written by human experts. Finally, as requested by many works in the literature, to evaluate the usability of our tool, we performed a preliminary user validation to evaluate the usability and the readability of the CNL (Section 7.5).

Chapter 8

Handling Fairness in Nuclear Medicine Scheduling Problem

In this chapter, we state why fairness rules are important, present the encoding we implemented to solve the Fairness problem related to the solution to the Nuclear Medicine Scheduling (NMS) problem, presented in previous Chapter 5, and analyze the obtained results.

8.1 Motivation and Fairness Problems in the NMS

In the context of AI and in particular in Digital Health it is important to not only present a working solution but also be aware of the possible problems that a solution could create. As an example, even an optimal solution could lead to financial waste due to, e.g., some machines remaining underutilized, leading to unnecessary costs. Further, this can have serious ethical and legal implications, can compromise the quality of care and could result in unequal treatment of patients. Fairness rules, not required to solve a problem, help to eliminate unwanted biases towards patients and diseases.

In the NMS, fairness issues may correspond to the underutilization of the tomographs, which are the machines that have a high cost and should be used as much as possible during the day, and the tendency of the solutions to schedule patients with protocols requiring low times. Indeed, being the minimization of unassigned patients the weak constraints with the highest priority, the solution assigns as many patients as possible, i.e. it prefers to assign two patients requiring low time in the tomograph than a single patient that requires long time in the tomograph.

```
1 to_check(ID1, PrID1, ID2, PrID2) :- not x(ID1, DAY, _, PrID1,0), reg(ID1,
    DAY, PrID1), x(ID2, DAY, _, PrID2,0), PrID1 != PrID2.
2 :~ to_check(ID1, PrID1, ID2, PrID2), cost(PrID1, N), exam(PrID1,3,M),
    exam(PrID2,3,M1), cost(PrID2, N1), (N+M) > (N1+M1). [1@1, ID1]
```

Figure 8.1 ASP rules for mitigating fairness problems.

Before presenting the rules to handle these problems, it is important to analyze the results we have obtained in order to decide if tailored rules are required and to determine if they work. By analyzing the results, it is possible to see that in more than 60% of the tested instances the tomographs are used more than 80% of the time. Further, considering only instances that are not optimally solved, the tomographs are used on average 89% of the time. Considering that all the protocols require some time before passing through the tomographs and then some unused time is unavoidable, we can state that in our solution the machines are already deeply utilized as a consequence of the minimization of the unassigned patients.

Concerning the biases towards registrations with protocols requiring a low time in the tomograph and on the phases before it, we again analyzed the results obtained by the direct encoding. In particular, we focused on two protocols that could get penalized from the schedule: protocol number 888, being the protocol requiring the longest time on the tomograph, and protocol number 819, being among the protocols requiring the largest time between all the phases. We excluded from this analysis protocol number 823 even if it is the protocol requiring the largest time in total because it is by far the most required protocol and the most assigned, occurring on some days as the only protocol requested. In the analysis, we found that, in general, 85% of the patients with any protocol but the 888 or the 819 are assigned. However, just 76% and 71% of the patients with protocols 888 and 819, respectively, are assigned. This highlights a bias towards the protocols that require less time, thus, in the following, exploiting the modularity of ASP, we will present a set of rules that can be added to the original encoding to solve this issue.

8.2 Rules for Handling Fairness and Results

The rules added to improve the fairness of the solutions are reported in Figure 8.1. In particular, they try to force the schedule to assign the treatment to as many registrations with long therapy as possible, without reducing the number of patients assigned by the schedule, being the added weak constraints with lowest priority. Indeed, since we want to ensure that registrations requiring protocols having longer treatments are assigned as the others, the atom

`to_check` is used to derive all the registrations that are not assigned and have a protocol that is different from the protocols required by the assigned registrations. Finally, the weak constraint minimizes the number of registrations that were not assigned and required a longer time in the hospital than the assigned ones.

We used the rules in Figure 8.1 in combination with the encoding presented in previous Chapter 5, here included again for clarity in Figure 8.2, in all the instances to test the effectiveness of the solution to overcome the bias towards some registrations (the encoding employed in this section can be found at: https://github.com/MarcoMochi/JLC2024NSP/tree/main/Fair). To determine if the new rules had a positive impact on the biases, we evaluated

```
0 {x(ID, D, TS, PrID, 0) : avail(TS, D)} 1 :- reg(ID, D, PrID).
{x(ID, D, START, PrID, P+1) : avail(START,D), START >= TS+NumTS, START <
    TS+NumTS+6} = 1 :- x(ID, D, TS, PrID, P), exam(PrID, P, NumTS), P >= 0,
    P < 3.
:- x(ID, _, TS, PrID, 3), exam(PrID, 3, NumTS), TS + NumTS > 120.
timeAnamnesis(ID, TS..TS+NumTS-1) :- x(ID, D, TS, PrID, 0), exam(PrID, 0,
    NumTS).
:- #count{ID: timeAnamnesis(ID, TS)} > 2, avail(TS,D).
timeOccupation(ID, D, TS, END-1, PrID) :- x(ID, D, TS, PrID, 1), x(ID, D,
    END, PrID, 3).
res(ID, D, TS..END,0) :- timeOccupation(ID, D, TS, END, PrID),
    required_chair(PrID).
res(ID, D, TS..TS+NumTS-1,1) :- x(ID, D, TS, PrID, 3), exam(PrID, 3, NumTS),
    required_chair(PrID).
res(ID, D, TS..END+NumTS-1,1) :- timeOccupation(ID, D, TS, END, PrID),
    exam(PrID, 3, NumTS), not required_chair(PrID).
:- #count{ID: tomograph(T, ID, D), x(ID, D, _, PrID, _)} > N, limit(PrID,
    N), tomograph(T,_).
1 {chair(C, ID, D) : chair(C, _)} 1 :- x(ID, D, _, PrID, _),
    required_chair(PrID).
1 {tomograph(T, ID, D) : tomograph(T, _)} 1 :- x(ID, D, _, PrID, _).
:- chair(C, ID, D), tomograph(T, ID, D), chair(C, R1), tomograph(T, R2), R1
    != R2.
chair(C, ID, D, TS) :- chair(C, ID, D), res(ID, D, TS, 0).
tomograph(T, ID, D, TS) :- tomograph(T, ID, D), res(ID, D, TS, 1).
:- #count{ID: tomograph(T, ID, D, TS)} > 1, tomograph(T,_), avail(TS,D).
:- #count{ID : chair(C, ID, D, TS)} > 1, chair(C,_), avail(TS,D).
:~ not x(ID, D, _, _,0), reg(ID, D, _). [1@2, ID, D]
:~ x(ID, _, START, PrID, 0), x(ID, _, END, _, 3), cost(PrID, NumTS), END -
    START - NumTS >= 0. [END - START - NumTS@1, ID]
```

Figure 8.2 ASP encoding of the problem.

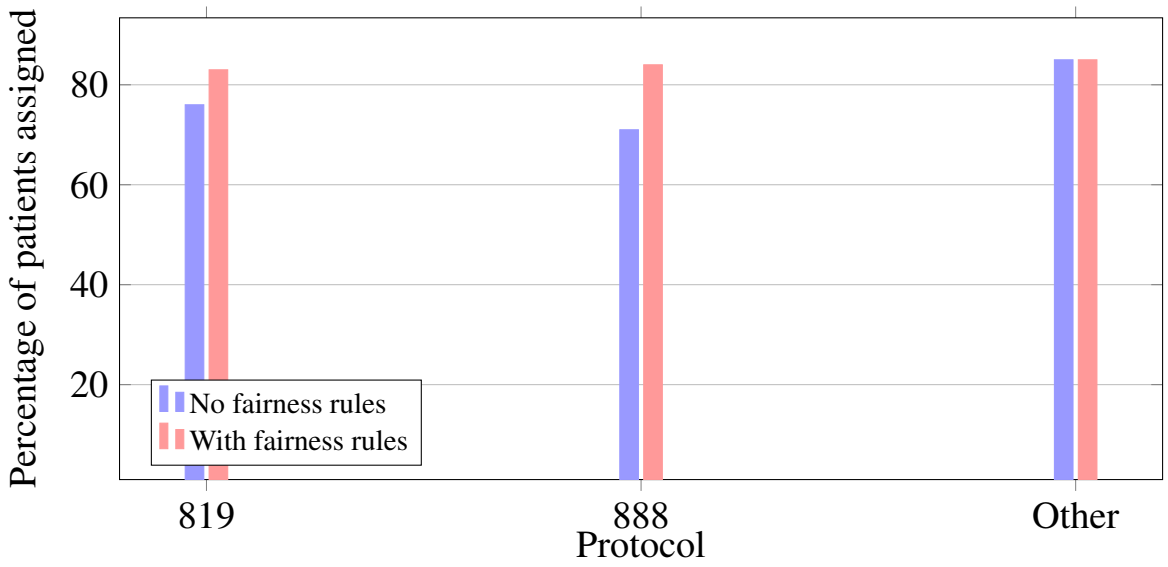

Figure 8.3 Percentage of patients having protocol 819 and 888 assigned when using the direct encoding without and with the fairness rules.

again the percentage of patients assigned according to their protocols and then we determined the cost of these new rules, in terms of the quality of the solutions. As can be seen from Figure 8.3, we obtained that patients with protocol 888 went from being assigned 76% of the time to 83%, while patients with protocol 819, in the new solution are assigned 84% of time, while it was 71% previously. These values are very close to the percentage of assigned patients having other protocols, that is 85%. This analysis confirms that the added rules are effective in reducing the bias towards the patients under certain protocols. However, it is important to understand what is the cost of these added rules. To analyze the impact of the added rules we consider as a benchmark the results obtained by the direct encoding. First of all, we found that 95% of the results obtained with the added rules have the same or fewer not assigned patients, which means that adding the rules does not decrease the quality of the solutions. Thus, just 5% of the tested instances obtained a worse solution with the added fairness rules. However, in all of these worsened solutions, the number of assigned patients decreased by just one patient, meaning that the added rules have a weak impact on decreasing the quality of the encoding. It is important to note that these results are obtained by maintaining the same timeout and that all the previously optimally solved solutions are still solved optimally introducing the fairness rules. Moreover, when using the encoding with the fairness rules, the solver requires at most 5 seconds more to find an optimal solution.

8.3 Conclusions

In conclusion, in this chapter we have highlited the importance of handling fairness problems possibly introduced by AI systems. In particular in the context of Digital Health. Having presented the motivation, we have proposed a solution to the problem in a previously presented ASP model and analyzed the results obtained through its implementation. The analysis of the results in done taking into account the impact of the added rules in the quality of the solution and in their ability to reduce the fairness problems. The analysis allows us to state that the proposed solution is a viable one and similar approaches could be implemented in other scheduling problems.

Part III

Conclusions

Chapter 9

Further Research

In this thesis, we presented the works we have conducted directly linked to scheduling Digital Health problems and increasing the explainability of ASP solutions. However, during the PhD, we have focused on other areas of Knowledge Representation and Reasoning too. Even if these studies were related to ASP but not centered around Digital Health problems, we still found that they are of interest due to their more theoretical approaches, to increase the performances and the usage of ASP in general.

The first work we want to present is "Comparing Planning Domain Models Using Answer Set Programming" (Chrpa et al., 2023), which was presented at the 18th European Conference on Logics in Artificial Intelligence. The work tackles a critical problem regarding automated planning and domain-independent planning. One of the primary aspects of domain-independent planning is the domain knowledge that must be fed into a planning engine that comes under the form of a domain model, a symbolic representation of the environment and actions, that has to be engineered prior its use (McCluskey and Porteous, 1997). The importance of good quality domain models in planning, and of the corresponding knowledge engineering process, has been well-argued (McCluskey et al., 2017; Vallati and Chrpa, 2019). However, there is a lack of approaches to support the knowledge engineering process, and with the widespread use of LLM-based approaches there is the concrete risk of generating a large number of low-quality models (Oswald et al., 2024). Beside providing the first concrete approach to assess if two domain models are strongly equivalent, the proposed notion of similarity, and the ASP-based approach to measure it, have several practical implications. In particular, there is no “diff” tool that compares different versions of a domain model and highlights differences among them helping knowledge engineers in understanding the changes. Indeed, often, models acquired by automated tools might have different ordering of elements or different names of predicates. Such a tool could help identify the equivalence of

two models. Moreover, it could be exploited as a plagiarism checker, where it can provide a “similarity” score to flag potential cases of plagiarism. The main contributions of the work are a formal definition of similarity between models and strong equivalence, a formal definition of submodels similarity / equivalence, and an ASP approach to compare lifted domain models with no assumptions on their characteristics or ability to solve problems from the same class. The proposed tool can deal with both classical STRIPS planning, and negative preconditions and durative actions as introduced in PDDL2.1 (Fox and Long, 2003). We proposed a directed graph representation of lifted domain models and we showed that domain models are structurally equivalent if and only if the graphs representing them are isomorphic. For strong equivalence, on top of structural equivalence, the numeric attributes such as action durations or action costs must be equal. Then, we define *distance* between domain models as the minimum number of modifications that have to be made to both models to make them strongly equivalent. It corresponds to a variant of the well-known notion of *edit distance* between two graphs (representing the domain models). Concerning the ASP-based solution, it is capable of providing optimally minimal sets of changes to transform one model into the other. In the work, we evaluated the approach on well-known benchmark domains from international competitions involving classical planning domains and PDDL2.1 domains containing durative actions, of different sizes with regards to the number of models’ predicates and operators, on ICKEPS domains and on the problem of computing "common cores" of planning domains.

The other work we want to present is "A Simple Proof-theoretic Characterization of Stable Models: Reduction to Difference Logic and Experiments", which has been recently accepted with minor revisions to the Artificial Intelligence Journal and a preliminary version was already published in Giunchiglia et al. (2023). We think that this work, which tackles both theoretical and practical aspects of ASP, is important for two reasons. Firstly, many scheduling problems, even in the Healthcare domain, have to deal with constraints that could be represented in Difference Logic. Moreover, the contribution about the characterization of answer sets, together with the definition of new reductions in Difference Logic, is an alternative way to increase the performances of the available tools. Indeed, in the paper, we present that in some of the tested benchmarks, our reduction can solve problems faster than state-of-the-art ASP solvers. Differently from other works, our proposed reduction to difference logic is not explicitly based on a formulation of Clark’s completion. Thus, it does not require Boolean variables. Other than providing a new theoretical characterization and a reduction, we have implemented it straightforwardly and optimized through the usage of the concept of Strongly Connected Components, which allows to reduce the number of

Difference logic constraints in the reduction. The test we have performed with our novel translation, by means of an SMT formula on well-known ASP benchmarks, shows that our approach is competitive to and often better than other translations and that it can also be faster than native ASP-solver in some domains.

Chapter 10

Related Work

In this chapter, we present detailed related work for the contributions presented in the thesis. Differently from the Section in Chapter 1, after a more general presentation of the scheduling problems solved through ASP, we focus on each contribution and present the related state of the art.

10.1 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., Alviano et al. (2020) for a recent survey), the first solved problem was the *Nurse Scheduling Problem* Alviano et al. (2017, 2018); Dodaro and Maratea (2017), where the goal is to create a scheduling for nurses working in hospital units. Then, the problem of assigning operating rooms to patients, in variations different from the one presented in Chapter 4, denoted as *Operating Room Scheduling* Dodaro et al. (2018, 2019b); Galatà et al. (2021), has been treated, and further extended to include bed management Dodaro et al. (2019a). More recent problems include the *Chemotherapy Treatment Scheduling* problem Dodaro et al. (2021), in which patients are assigned a chair or a bed for their treatments, the *Rehabilitation Scheduling Problem* Cardellini et al. (2021), which assigns patients to operators in rehabilitation sessions, the *Master Surgical Scheduling* problem Galatà et al. (2024); Mochi et al. (2023), which assigns the specialties to the available and compatible operating rooms to meet the hospital target of assignments. In Cappanera et al. (2022), it is proposed a solution using ASP to the problem of assigning a date to visit or therapy for multiple recurrent exams to chronic patients. The problem is split into two sub-problems.

Concerning scheduling problems beyond the Healthcare domain, ASP encoding were proposed for the following problems: *Incremental Scheduling Problem* Balduccini (2011), where the goal is to assign jobs to devices such that their executions do not overlap one another; *Team Building Problem* Ricca et al. (2012), where the goal is to allocate the available personnel of a seaport for serving the incoming ships; and the *Conference Paper Assignment Problem* Amendola et al. (2016), which deals with the problem of assigning reviewers in the Program Committee to submitted conference papers. Other relevant papers are Gebser et. al Gebser et al. (2018b), 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. Falkner et al. (2018), where industrial applications dealt with ASP are presented, including those involving scheduling problems. Other problems related to the industrial context involve the works on Semiconductor Manifacturing and Job-Shop scheduling El-Kholany et al. (2023, 2022).

10.2 Solving the Pre-Operative Assessment Clinic problem

Edward at al. Edward et al. (2008) 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. Stark et al. (2015) 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 Correll et al. (2006); Ferschl et al. (2005); Harnett et al. (2010); Tariq et al. (2016); Woodrum et al. (2017) 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 an important tool to avoid the cancellation of surgeries and to significantly reduce the risk associated with the surgery.

10.3 Solving the Operating Room Scheduling problem

Şeyda and Tamer Şeyda Gür and Eren (2018) presented a comprehensive overview of different approaches to the ORS problem. Two works proposed solutions to the ORS problem and testing them with real data are the ones by Aringhieri et al. Aringhieri et al. (2015) and Landa et al. Landa et al. (2016). In the former, the problem of scheduling surgical interventions over a one-week planning horizon is analyzed, considering several departments that share a fixed number of ORs and post-operative beds. The proposed two-phase method aims at minimizing patient waiting times and maximizing hospital resource utilization. The second work deals with two interconnected sub-problems: first patients are assigned to specific dates in a given planning horizon, while the second sub-problem concerns the assignment to ORs and the sequencing of operations in each OR. To solve the overall problem, a hybrid two-phase optimization algorithm exploits neighborhood search techniques combined with Monte Carlo simulation. Moreover, Hamid et al. Hamid et al. (2019) incorporated the Decision-Making Styles (DMS) of surgical teams to deal with constraints such as the disposition of material and resources, patient priorities and availability, as well as skills and competencies of the surgical team. Two metaheuristics to find Pareto-optimal solutions, namely, a non-dominated sorting genetic algorithm and multi-objective particle swarm optimization, are developed and evaluated on the data from a hospital in Iran. Zhang, Dridi, and El Moudni Zhang et al. (2017) addressed the problem of scheduling ORs with different needs for both elective and non-elective patients. A time-dependent policy is applied to determine patient priorities based on urgency levels and waiting times. This problem is formulated as a stochastic shortest-path Markov Decision Process (MDP) with blind alleys and solved using the asynchronous value iteration method. Results of a numerical experiment on synthetic data show that, compared to the regular MDP model, the proposed time-dependent policy is more efficient in reducing patient waiting times without leading to an excessive increase of the ORs usage.

10.4 Solving the NMS problem

The work in Pérez et al. (2011) proposed four different scheduling solutions to the NMS, each giving priority to a different aspect. Specifically, the first solution focuses on the preferences of the patients. The second one assigns the patients as soon as possible. The third one is a combination of the first two, while the fourth one fixes the technologists to the machines and assigns the patients based on the machine availability. The proposed solutions were tested on the data of one of the biggest hospitals in Texas in a simulated environment. Another work

using real data to validate the solution is Xiao et al. (2018). The authors got the data from the West China Hospital and proposed a two-step solution. The first step is the development of a nonlinear integer programming model considering the settings of the hospital and the drugs. After obtaining the solution of the first step, they developed a stochastic online algorithm following different scheduling strategies to adapt the solution to the real requests of patients. The authors of Akhavizadegan et al. (2017) addressed the NMS problem using a Markov decision process (MDP) to decide how many patients to schedule in a day from a tactical perspective and which patient can move on to the next phase from an operational perspective. The MDP is then solved using two heuristic algorithms and a mathematical programming model. This system is evaluated against historical data of patients scheduled following a First-Come-First-Served strategy. The historical data comes from the public Hospital in Tehran-Iran.

10.5 Solving XAI problems in ASP

In the context of ASP, XAI translates to understanding why an atom is or is not included in an answer set, or why an answer set was or was not computed. There are multiple approaches to XAI in the context of ASP, including algorithmic (Perri et al., 2007), stepping-through (Oetsch et al., 2011), meta-programming (Syrjänen, 2006), graph-based, argumentation-based (Schulz and Toni, 2016), semantic-based (Schulz et al., 2015), and unsat-based methods. For a complete survey, we refer the reader to (Fandinno and Schulz, 2019).

Algorithmic approaches. An algorithmic approach is represented by IDEAS (Brain and De Vos, 2005) that explains both why a set of atoms S is in an answer set M, and why S is not in any answer set. Both IDEAS algorithms are similar to the ones implemented in ASP solvers and try to decide which rules are responsible for the derivation or non-derivation of atoms in S. Moreover, IDEAS allows a programmer to query for an explanation of an observed fault, to analyze the obtained results and then reformulate the query to make it more precise. Arias et al. (2020), in S(CASP), provide partial stable models representing the relevant part of the stable model related to an initial query. This is done in the context of ASP with constraints (CASP) and the justifications are obtained through a top-down evaluation tree. This top-down approach does not require a grounded ASP program.

Stepping-through approaches. Oetsch et al. (2011) presented an interactive debugging technique inspired by imperative language tools. Since ASP is a pure declarative language

lacking an explicit control flow the authors introduced the concept of active rules and states. Through these states, the user can decide which rules to activate and then analyze the induced states.

Meta-programming approaches. Meta-programming approaches employ a program written in a meta-language, which can be seen as a simulation of an ASP solver, to manipulate a program written in an object language (the faulty program). Each answer set of a meta-program contains a diagnosis, which is a set of meta-atoms describing the reasons why a particular interpretation of the faulty program is not an answer set. SPOCK (Gebser et al., 2008) and OUROBOROS (Oetsch et al., 2010; Polleres et al., 2013) enable the identification of faults connected with over-constrained problems and unfounded sets. Both approaches represent the input program in a reified form, allowing the application of a debugging meta-program. The main difference between the two lies in the fact that SPOCK can debug only ground programs, whereas OUROBOROS can handle both ground and non-ground programs.

Graph-based approaches. In the context of graph-based approaches, XASP2 (Alviano et al., 2024) focuses on explaining why an atom is present or absent in an answer set, starting from a few initial atoms. It represents explanations in the form of directed acyclic graphs, with meta-programming techniques employed for obtaining these explanations. DISCASP (Li et al., 2021) introduces a graph-based algorithm capable of determining, from a given atom, the set of atoms in the answer set that are relevant to it. Notably, this set of relevant atoms can be computed incrementally, with the program defining the relationship between the original atom and these relevant atoms. Another graph-based approach was presented by Cabalar et al. (2014), where a causal justification graph is constructed. Each atom is associated with a set of justifications for its derivation, and the edges between atoms are enriched through various operations, explaining the interactions between different justifications and the atom. The system XCLINGO (Cabalar et al., 2020) is based on this concept and uses annotations to generate derivation trees to construct a causal chain of derivation. Its extension, XCLINGO2 (Cabalar and Muñiz, 2023), relies on a different notion of an explanation graph, and introduces filtering operations. Lastly, an XAI approach using a tree-based system was described by Marynissen et al. (2022), facilitating the creation of justifications for literals. While this framework supports constraints and aggregates, the technique presented in the paper has yet to be implemented in a tool, as far as we know.

Table 10.1 Summary of the supported features of different XAI approaches.

	Interactive Expl.	Expl. of False Atoms	Support Constraints	Support Aggregates	System Avai.
XCLINGO (Cabalar et al., 2020)	No	No	No	No	Yes
XCLINGO2 (Cabalar and Muñiz, 2023)	No	No	No	No	Yes
S(CASP) (Arias et al., 2020)	No	Yes	Yes	No	Yes
DISCASP (Li et al., 2021)	Yes	No	Yes	No	Yes
VISUAL-DLV (Perri et al., 2007)	Yes	No	Yes	No	Yes
OUROBOROS (Oetsch et al., 2010)	No	No	Yes	No	Yes
SPOCK (Gebser et al., 2008)	No	No	Yes	No	Yes
DWASP (Dodaro et al., 2019c)	Yes	No	Yes	No	Yes
ASPERIX (Béatrix et al., 2016)	No	Yes	Yes	No	Yes
LABAS (Schulz and Toni, 2016)	No	Yes	No	No	Yes
(Eiter and Geibinger, 2023)	No	Yes	Yes	Yes	No
(Marynissen et al., 2022)	No	Yes	Yes	Yes	No
XASP2 (Alviano et al., 2024)	Yes	Yes	Yes	Yes	Yes
E-ASP	Yes	Yes	Yes	Yes	Yes

Argumentation-based approaches. Schulz and Toni (2016) introduced two justification methods based on argumentation theory and the relationship between a logic program and the corresponding ABA (Argumentation-Based Approaches) framework. These methods aim to explain why an atom is or is not included in an answer set and rely on Attack Trees, from which information regarding supporting and attacking literals can be derived.

Semantic-based approaches. Eiter and Geibinger (2023) proposed representing the justification of the presence or absence of an atom in an answer set using *abstract constraint* atoms (referred to as *c-atoms*). However, as far as we know, no system employing these techniques has been developed. Another semantic-based approach is presented by Cabalar and Fandinno (2016), where the authors extend a logic programming semantics based on causal justifications to handle disjunctive programs. In this framework, each positive atom is associated with a set of justifications, obtained using rule labels and three algebraic operations: addition, product, and application.

Unsat-based approaches. Among unsat-based approaches, DWASP (Dodaro et al., 2015a, 2019c) identifies the faulty rules through the usage of supporting atoms and minimal unsatisfiable sets (essentially reasons of incoherence correspond to minimal unsatisfiable sets, see (Alviano et al., 2023a)). Additionally, DWASP supports a query-based interface (Shchekotykhin, 2015) for reducing the size of unsatisfiable sets. E-ASP also employs minimal unsatisfiable sets and extends the capabilities of DWASP by providing explanations for the presence or absence of atoms in a given answer set. Additionally, E-ASP supports a stepping-through approach for aggregates, a feature not available in DWASP. From an implementation

perspective, E-ASP is built upon the state-of-the-art system CLINGO (Gebser et al., 2016), while DWASP is based on WASP (Alviano et al., 2015). Importantly, E-ASP is implemented as an external tool and does not require modifications of the internal workings of CLINGO. This design choice ensures that updating the underlying solver should be relatively straightforward as long as its external interface (input and output format) remains unchanged.

Summary. Table 10.1 gives a summary of some of the features of some XAI tools presented in recent years in the context of ASP. The table expands the one presented by Alviano et al. (2023b). In particular, the first column reports whether a system allows for interacting with the found explanations, since interactivity is one of the key aspects identified by Miller (2017) to obtain a good explanation.

10.6 Translation from Controlled Natural Language to ASP

In this section, we overview the main CNLs proposed in the area of logic programming; for a complete review of CNL, we refer the reader to the interesting survey by Kuhn (2014).

One of the first attempts of designing encoding expressed in a CNL as logic programs was presented by Fuchs and Schwitter (1995) and by Schwitter et al. (1995), where Attempto CNL (Fuchs, 2005) was proposed, whose idea was to convert sentences expressed in a CNL as Prolog clauses. Clark et al. (2005) presented a Computer-Processable Language (CPL), whose key principle was to be easier for computers rather than a language easier for users. Moreover, they presented also an interpreter and a reasoner for this language, and discussed the strengths and weaknesses of natural languages to be used as a the basis for knowledge acquisition and representation.

Concerning ASP, Erdem and Yeniterzi (2009) proposed BIOQUERYCNL, a CNL for biomedical queries, and developed an algorithm designed to automatically encode a biomedical query expressed in this language as an ASP program. BIOQUERYCNL is a subset of Attempto CNL and it can represent queries of the form *Which symptoms are alleviated by the drug Epinephrine?* (we refer the reader to Chapter 3 of (Erdem and Yeniterzi, 2009) for more queries). Later on, BIOQUERYCNL was also used as a basis to generate explanation of complex queries (Öztok and Erdem, 2011). The main difference with our approach is that CNL2ASP does not cover query answering and is not specialized on one particular application context.

Baral and Dzifcak (2012) proposed a CNL specific for solving logic puzzles. The CNL was split into two sets of sentences, namely *Puzzle Domain data* and *Puzzle clues*. The former plays a similar role of our explicit domain definitions (see Section 7.1), whereas Puzzle clues can be seen as the logic rules to solve the puzzle. As in our case, the CNL was then automatically converted into ASP rules.

Lifschitz (2022) showed the process of translating the English sentence “A prime is a natural number greater than 1 that is not a product of two smaller natural numbers." into executable ASP code.

Schwitter (2018) defined the language PENG^{ASP}, a CNL that is automatically converted into ASP. Albeit some aspects of PENG^{ASP}'s grammar rules are present in the grammar of our CNL, the latter is geared more towards the formal definition of combinatorial problems in a natural-feeling and unambiguous way that is also reliably predictable in its translation to ASP, choosing words that would stand out easily during reading and with an easily deducible meaning from the given context; this meant sacrificing some of the naturalness of PENG^{ASP}. In addition, the grammar of PENG^{ASP} is designed for allowing a conversion from the CNL to ASP and then back in the other direction, whereas in CNL2ASP this possibility is not yet available, although the language has been designed in such a way that it should be possible to make it viable. Another feature that is available in PENG^{ASP} is the possibility to express queries, which is not possible in our CNL. However, our CNL presents some features that, to best of our knowledge, are not available in PENG^{ASP}, such as explicit definitions, and positive strong constraints, that we found to be useful for specifying real-world problems in a natural way. Moreover, it should be noted that the implementation of PENG^{ASP}, as well as a binary executable, is not yet public, whereas the implementation of CNL2ASP is open source and publicly available. As an example of the differences with our CNL, we report a comparison with the CNL for specifying the graph coloring problem used by PENG^{ASP} (Figure 5 of (Schwitter, 2018)[1]).

```
1 The node 1 is connected to the nodes 2 and 3.
2 The node 2 is connected to the nodes 1 and 3.
3 The node 3 is connected to the nodes 1 and 2.
4 Red is a colour.      Green is a colour.      Blue is a colour.
5 Every node is assigned to exactly one colour.
6 It is not the case that a node X is assigned to a colour and a node Y is
    assigned to the colour and the node X is connected to the node Y.
```

[1]Since PENG^{ASP} is not publicly available we could not compare the two languages on the other problems used in our paper.

Characteristic	CNL2ASP	λ-Based	BIOQUERYCNL	PENG^{ASP}
Simple sentences	Y	Y	Y	Y
Modifying clauses	Y	Y	Y	Y
Comparative clauses	Y	Y	Y	Y
Conjunction/disjunction clauses	Y	N	Y	Y
Conditional sentences	Y	N	N	Y
Negated sentences	Y	Y	N	Y
Cardinality constraints	Y	N	Y	Y
Aggregates	Y	N	N	U
Temporal sentences	Y	Y	N	Y
Preferences	Y	N	N	Y
Queries	N	Y	Y	Y

Table 10.2 Comparison of the linguistic features of CNL2ASP, λ-based (Baral and Dzifcak (2012)), BIOQUERYCNL (Erdem and Yeniterzi (2009)), and PENG^{ASP} (Schwitter (2018)). Yes (Y) means the construct is supported, No (N) means that the construct is not supported, Unknown (U) means that there is no evidence that the construct is supported nor unsupported.

There are two main differences with our CNL presented in Chapter 7. The first one is that our CNL must use variables (i.e., `X` in our example) also to specify the connections, whereas the one of PENG^{ASP} does not need it. In our case, sentence at line 1 would create the atom `connected_to(1,2,3)`. Secondly, the last sentence is expressed in a negative form in case of PENG^{ASP}, which is similar to the concept of constraint in ASP, whereas our CNL uses a positive sentence which is similar to the concept of clause in propositional logic. Moreover, the PENG^{ASP} and the CNL2ASP methodologies differ in the way the sentences are processed before being unified with the grammar rules. First of all, the grammar rules for PENG^{ASP} are specified with a Definite Clause Grammar (DCG), while in our solution the grammar is defined in Extended Backus-Naur Form (EBNF). Moreover, our tool builds a sort of syntax tree for handling the internal structure of the sentences before rewriting them into ASP. While in PENG^{ASP}, after the collection in the DCG, a chart parser is used to extract the information needed for the translation and this information can be parsed and passed to the users to help with completing the sentence.

We also mention that some of the sentences used in the CNL presented in this paper are inspired by the Semantics of Business Vocabulary and Business Rules (SBVR) (Bajwa et al., 2011; The Business Rules Group, 2000), which is a standard proposed by the Object Management Group to formally describe complex entities, e.g., the ones related to a business, using natural language.

In Table 10.2 we present a comparison of the features of the different CNLs translating to ASP. In particular, we want to highlight the constructs that are covered by the CNLs in order to ease the usage of the tool and to be more adherent to natural language. We considered the same constructs considered in Schwitter et al. (1995) plus temporal sentences and new constructs specifically adopted for ASP: cardinality constraints, aggregates and preferences.

10.7 Fairness in Scheduling problems

The approach in Cappanera et al. (2023b) exploits ASP to address the problem of fairness and preference satisfaction in the mid-term scheduling of medical staff within a hospital network.

The paper Ala et al. (2021) addresses the appointment scheduling problem in hospitals, utilizing a fairness-oriented approach to optimize resource allocation and ensure equitable access to medical services. In Ala et al. (2022), the authors propose a mixed-integer linear programming model for patient appointment scheduling in hospitals. Their model prioritizes patients based on their overall health status and incorporates fairness as a key consideration to minimize patient waiting times and improve healthcare service equity. The authors of Masroor et al. (2024) explore real-world patient arrival patterns and multiple allocation techniques to understand the impact of optimized allocation on the fairness of the experience for patients. The authors also investigate how tuning machine learning hyperparameters can balance optimization with fairness considerations. Finally, the work Ferrara (2023) provides a comprehensive survey on fairness and bias in AI, addressing their sources, impacts, and mitigation strategies.

Chapter 11

Conclusions

In this thesis, we presented contributions to two different areas. The first one is our contributions to the Scheduling problems in Digital Health. In particular, we presented solutions to three problems using Answer Set Programming (ASP), a declarative programming language that is often used in complex scheduling problems. The problems we solved are the PRE-OPERATIVE ASSESSMENT CLINIC (PAC), the OPERATING ROOM SCHEDULING (ORS), and the NUCLEAR MEDICINE SCHEDULING (NMS) problem. For all these problems, we presented the proposed solution and tested them using realistic data for the PAC problem and real data for the ORS and the NMS. The analysis of the results confirms that our solutions are able to reach good performances and, when the comparison is possible, improve the solutions of the hospitals. Moreover, we compared our solutions to other logic-based formalisms. The second area in which we presented our contributions is related to Explainable Artificial Intelligence (XAI) and the Fairness problem when using AI-based solutions. Concerning XAI, we defined the theoretical aspects and presented a tool that increases the explainability of ASP solutions through the inspection of the solutions. The tool, E-ASP, allows us to identify the set of rules justifying a solution. Moreover, E-ASP is able to provide explanations over aggregates via a stepping-through approach, enhancing its utility in complex scenarios. To increase, instead, the ASP readability and interpretability, we presented CNL2ASP. CNL2ASP allows the translation of a Controlled Natural Language to an ASP encoding, allowing non-expert users to get an encoding, without knowing the ASP syntax. Moreover, we presented three examples of how such a tool can be used with real-world problems and performed an analysis of the usability of the tool. Finally, to address the problem of Fairness often introduced using AI solutions, starting from the NMS problem and exploiting the modularity of ASP, we proposed an encoding that can be modularly added to the original one

to reduce the biases of the AI solution in such a problem. To confirm the viability of such a solution we performed an experimental analysis to assess the quality of our solution.

For future works, we would like to improve the quality of the solutions to the scheduling problems by studying possible solutions implementing alternative approaches to ASP programming, such as Logic-Based Bender's Decomposition (Benders (2005); Cappanera et al. (2023a); Heching and Hooker (2016); Hooker (2004); Hooker and Ottosson (2003); Rahmaniani et al. (2017)) or making use of Machine Learning techniques to improve the performances of the ASP solutions. Another problem we want to study is the development of a general framework to solve a broad set of scheduling problems, exploring the common parts of these problems and the modularity of ASP. Moreover, to increase the interpretability and explainability of our solutions, we would like to explain the ASP solutions through the usage of natural language sentences via the CNL.

References

Abedini, A., Ye, H., and Li, W. (2016). Operating room planning under surgery type and priority constraints. *Procedia Manufacturing*, 5:15–25.

Abels, D., Jordi, J., Ostrowski, M., Schaub, T., Toletti, A., and Wanko, P. (2019). Train scheduling with hybrid ASP. In *LPNMR*, volume 11481 of *Lecture Notes in Computer Science*, pages 3–17. Springer.

Akhavizadegan, F., Ansarifar, J., and Jolai, F. (2017). A novel approach to determine a tactical and operational decision for dynamic appointment scheduling at nuclear medical center. *Computers & Operations Research*, 78:267–277.

Ala, A., Alsaadi, F. E., Ahmadi, M., and Mirjalili, S. (2021). Optimization of an appointment scheduling problem for healthcare systems based on the quality of fairness service using whale optimization algorithm and nsga-ii. *Scientific Reports*, 11(1):19816.

Ala, A., Simic, V., Pamucar, D., and Tirkolaee, E. B. (2022). Appointment scheduling problem under fairness policy in healthcare services: fuzzy ant lion optimizer. *Expert systems with applications*, 207:117949.

Alade, O. M. and Amusat, A. O. (2019). Solving nurse scheduling problem using constraint programming technique. *arXiv preprint arXiv:1902.01193*.

Alami, H., Gagnon, M.-P., and Fortin, J.-P. (2017). Digital health and the challenge of health systems transformation. *MHealth*, 3:31.

Alviano, M., Amendola, G., Dodaro, C., Leone, N., Maratea, M., and Ricca, F. (2019a). Evaluation of disjunctive programs in WASP. In *LPNMR 2019*, volume 11481 of *LNCS*, pages 241–255. Springer.

Alviano, M., Amendola, G., Dodaro, C., Leone, N., Maratea, M., and Ricca, F. (2019b). Evaluation of disjunctive programs in WASP. In Balduccini, M., Lierler, Y., and Woltran, S., editors, *LPNMR*, volume 11481 of *LNCS*, pages 241–255. Springer.

Alviano, M., Bertolucci, R., Cardellini, M., Dodaro, C., Galatà, G., Khan, M. K., Maratea, M., Mochi, M., Morozan, V., Porro, I., and Schouten, M. (2020). Answer set programming in healthcare: Extended overview. In *IPS and RCRA 2020*, volume 2745 of *CEUR Workshop Proceedings*. CEUR-WS.org.

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

Alviano, M., Dodaro, C., Fiorentino, S., Previti, A., and Ricca, F. (2023a). ASP and subset minimality: Enumeration, cautious reasoning and MUSes. *Artif. Intell.*, 320:103931.

Alviano, M., Dodaro, C., Leone, N., and Ricca, F. (2015). Advances in WASP. In *LPNMR*, volume 9345 of *LNCS*, pages 40–54. Springer.

Alviano, M., Dodaro, C., and Maratea, M. (2017). An advanced answer set programming encoding for nurse scheduling. In *AI*IA*, volume 10640 of *LNCS*, pages 468–482. Springer.

Alviano, M., Dodaro, C., and Maratea, M. (2018). Nurse (re)scheduling via answer set programming. *Intelligenza Artificiale*, 12(2):109–124.

Alviano, M., Ly Trieu, L., Cao Son, T., and Balduccini, M. (2024). The xai system for answer set programming xasp2. *Journal of Logic and Computation*, page exae036.

Alviano, M., Trieu, L. L. T., Son, T. C., and Balduccini, M. (2023b). Advancements in xASP, an XAI System for Answer Set Programming. In *Proc. of CILC*, volume 3428 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Amendola, G., Dodaro, C., Leone, N., and Ricca, F. (2016). On the application of answer set programming to the conference paper assignment problem. In *AI*IA*, volume 10037 of *Lecture Notes in Computer Science*, pages 164–178. Springer.

Ansótegui, C., Pacheco, T., and Pon, J. (2019). Pypblib.

Arias, J., Carro, M., Chen, Z., and Gupta, G. (2020). Justifications for Goal-Directed Constraint Answer Set Programming. In *Proc. of ICLP Technical Communications*, volume 325 of *EPTCS*, pages 59–72.

Aringhieri, R., Landa, P., Soriano, P., Tànfani, E., and Testi, A. (2015). A two level metaheuristic for the operating room scheduling and assignment problem. *Computers & Operations Research*, 54:21–34.

Bajwa, I. S., Lee, M. G., and Bordbar, B. (2011). SBVR business rules generation from natural language specification. In *AI for Business Agility*. AAAI.

Balduccini, M. (2011). Industrial-size scheduling with ASP+CP. In Delgrande, J. P. and Faber, W., editors, *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, volume 6645 of *Lecture Notes in Computer Science*, pages 284–296. Springer.

Balduccini, M., Gelfond, M., Watson, R., and Nogueira, M. (2001). The USA-Advisor: A case study in answer set planning. In *LPNMR*, volume 2173 of *LNCS*, pages 439–442. Springer.

Baral, C. (2003). *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Baral, C. and Dzifcak, J. (2012). Solving puzzles described in english by automated translation to answer set programming and learning how to do that translation. In *Proceedings of KR*. AAAI Press.

Béatrix, C., Lefèvre, C., Garcia, L., and Stéphan, I. (2016). Justifications and blocking sets in a rule-based answer set computation. In *Proc. of ICLP Technical Communications*, volume 52 of *OASIcs*, pages 6:1–6:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Benders, J. (2005). Partitioning procedures for solving mixed-variables programming problems. *Computational Management Science*, 2(1).

Brain, M. and De Vos, M. (2005). Debugging logic programs under the answer set semantics. In *Proc. of Answer Set Programming, Advances in Theory and Implementation Workshop*, volume 142 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., and Woltran, S. (2007). That is illogical captain! the debugging support tool spock for answer-set programs: system description. In *Proc. of SEA*, pages 71–85.

Brewka, G., Eiter, T., and Truszczynski, M. (2011). Answer set programming at a glance. *Communications of the ACM*, 54(12):92–103.

Buccafurri, F., Leone, N., and Rullo, P. (2000). Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860.

Cabalar, P. and Fandinno, J. (2016). Justifications for programs with disjunctive and causal-choice rules. *Theory Pract. Log. Program.*, 16(5-6):587–603.

Cabalar, P., Fandinno, J., and Fink, M. (2014). Causal graph justifications of logic programs. *CoRR*, abs/1409.7281.

Cabalar, P., Fandinno, J., and Muñiz, B. (2020). A system for explainable answer set programming. In *Proc. of ICLP Technical Communications*, volume 325 of *EPTCS*, pages 124–136.

Cabalar, P. and Muñiz, B. (2023). Explanation graphs for stable models of labelled logic programs. In Arias, J., Batsakis, S., Faber, W., Gupta, G., Pacenza, F., Papadakis, E., Robaldo, L., Rückschloß, K., Salazar, E., Saribatur, Z. G., Tachmazidis, I., Weitkämper, F., and Wyner, A. Z., editors, *Proceedings of the International Conference on Logic Programming 2023 Workshops co-located with the 39th International Conference on Logic Programming (ICLP 2023), London, United Kingdom, July 9th and 10th, 2023*, volume 3437 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., and Schaub, T. (2020a). ASP-Core-2 input language format. *Theory and Practice of Logic Programming*, 20(2):294–309.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., and Schaub, T. (2020b). Asp-core-2 input language format. *Theory Pract. Log. Program.*, 20(2):294–309.

Cappanera, P., Gavanelli, M., Nonato, M., and Roma, M. (2022). A decomposition approach to the clinical pathway deployment for chronic outpatients with comorbidities. In Amorosi, L., Dell'Olmo, P., and Lari, I., editors, *Optimization in Artificial Intelligence and Data Sciences*, pages 213–226, Cham. Springer International Publishing.

Cappanera, P., Gavanelli, M., Nonato, M., and Roma, M. (2023a). Logic-based Benders decomposition in answer set programming for chronic outpatients scheduling. *Theory and Practice of Logic Programming*, 23(4):848–864.

Cappanera, P., Gavanelli, M., Nonato, M., Roma, M., Fabbri, N., Feo, C., and Soverini, R. (2023b). Fairness-driven mid-term scheduling of medical staff at a hospital network.

Cardellini, M., Nardi, P. D., Dodaro, C., Galatà, G., Giardini, A., Maratea, M., and Porro, I. (2021). A two-phase ASP encoding for solving rehabilitation scheduling. In Moschoyiannis, S., Peñaloza, R., Vanthienen, J., Soylu, A., and Roman, D., editors, *Proceedings of the 5th International Joint Conference on Rules and Reasoning (RuleML+RR 2021)*, volume 12851 of *Lecture Notes in Computer Science*, pages 111–125. Springer.

Cardellini, M., Nardi, P. D., Dodaro, C., Galatà, G., Giardini, A., Maratea, M., and Porro, I. (2024). Solving rehabilitation scheduling problems via a two-phase ASP approach. *Theory Pract. Log. Program.*, 24(2):344–367.

Caruso, S., Galatà, G., Maratea, M., Mochi, M., and Porro, I. (2021). An ASP-based approach to scheduling pre-operative assessment clinic. In Bandini, S., Gasparini, F., Mascardi, V., Palmonari, M., and Vizzari, G., editors, *Proc. of AIxIA 2021 - Advances in Artificial Intelligence - 20th International Conference of the Italian Association for Artificial Intelligence, Revised Selected Papers*, volume 13196 of *Lecture Notes in Computer Science*, pages 671–688. Springer.

Chrpa, L., Dodaro, C., Maratea, M., Mochi, M., and Vallati, M. (2023). Comparing planning domain models using answer set programming. In Gaggl, S. A., Martinez, M. V., and Ortiz, M., editors, *Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings*, volume 14281 of *Lecture Notes in Computer Science*, pages 227–242. Springer.

Clark, P., Harrison, P., Jenkins, T., Thompson, J. A., and Wojcik, R. H. (2005). Acquiring and using world knowledge using a restricted subset of english. In *Proceedings of FLAIRS*, pages 506–511. AAAI Press.

Correll, D., Bader, A., Hull, M., Hsu, C., Tsen, L., and Hepner, D. (2006). Value of Preoperative Clinic Visits in Identifying Issues with Potential Impact on Operating Room Efficiency. *Anesthesiology*, 105(6):1254–1259.

COSTANTINI, S. and PROVETTI, A. (2005). Normal forms for answer sets programming. *Theory and Practice of Logic Programming*, 5(6):747–760.

Di Gaspero, L. and Urli, T. (2014). A cp/lns approach for multi-day homecare scheduling problems. In Blesa, M. J., Blum, C., and Voß, S., editors, *Hybrid Metaheuristics*, pages 1–15, Cham. Springer International Publishing.

Dingess, M. and Truszczynski, M. (2020). Automated aggregator - rewriting with the counting aggregate. In *Proceedings of ICLP Technical Communications*, volume 325 of *EPTCS*, pages 96–109.

Dodaro, C., Galatà, G., Grioni, A., Maratea, M., Mochi, M., and Porro, I. (2021). An ASP-based solution to the chemotherapy treatment scheduling problem. *Theory and Practice of Logic Programming*, 21(6):835–851.

Dodaro, C., Galatà, G., Khan, M. K., Maratea, M., and Porro, I. (2019a). An ASP-based solution for operating room scheduling with beds management. In Fodor, P., Montali, M., Calvanese, D., and Roman, D., editors, *Proceedings of the Third International Joint Conference on Rules and Reasoning (RuleML+RR 2019)*, volume 11784 of *Lecture Notes in Computer Science*, pages 67–81. Springer.

Dodaro, C., Galatà, G., Khan, M. K., Maratea, M., and Porro, I. (2020). Solving operating room scheduling problems with surgical teams via answer set programming. In Baldoni, M. and Bandini, S., editors, *AIxIA 2020 - Advances in Artificial Intelligence - Revised Selected Papers of the 19th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2020)*, volume 12414 of *Lecture Notes in Computer Science*, pages 204–220. Springer.

Dodaro, C., Galatà, G., Maratea, M., and Porro, I. (2018). Operating room scheduling via answer set programming. In *AI*IA*, volume 11298 of *LNCS*, pages 445–459. Springer.

Dodaro, C., Galatà, G., Maratea, M., and Porro, I. (2019b). An ASP-based framework for operating room scheduling. *Intelligenza Artificiale*, 13(1):63–77.

Dodaro, C., Gasteiger, P., Musitsch, B., Ricca, F., and Shchekotykhin, K. M. (2015a). Interactive debugging of non-ground ASP programs. In *Proc. of LPNMR*, volume 9345 of *LNCS*, pages 279–293. Springer.

Dodaro, C., Gasteiger, P., Reale, K., Ricca, F., and Schekotihin, K. (2019c). Debugging non-ground ASP programs: Technique and graphical tools. *Theory Pract. Log. Program.*, 19(2):290–316.

Dodaro, C., Leone, N., Nardi, B., and Ricca, F. (2015b). Allotment problem in travel industry: A solution based on ASP. In *RR*, volume 9209 of *LNCS*, pages 77–92. Springer.

Dodaro, C. and Maratea, M. (2017). Nurse scheduling via answer set programming. In *LPNMR*, volume 10377 of *LNCS*, pages 301–307. Springer.

Edward, G. M., Das, S. F., Elkhuizen, S. G., Bakker, P. J. M., Hontelez, J. A. M., Hollmann, M. W., Preckel, B., and Lemaire, L. C. (2008). Simulation to analyse planning difficulties at the preoperative assessment clinic. *BJA: British Journal of Anaesthesia*, 100(2):195–202.

Eiter, T. and Geibinger, T. (2023). Explaining answer-set programs with abstract constraint atoms. In *Proc. of IJCAI*, pages 3193–3202. ijcai.org.

El-Kholany, M. M. S., Ali, R., and Gebser, M. (2023). Hybrid asp-based multi-objective scheduling of semiconductor manufacturing processes. In Gaggl, S. A., Martinez, M. V., and Ortiz, M., editors, *Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings*, volume 14281 of *Lecture Notes in Computer Science*, pages 243–252. Springer.

El-Kholany, M. M. S., Gebser, M., and Schekotihin, K. (2022). Problem decomposition and multi-shot ASP solving for job-shop scheduling. *Theory Pract. Log. Program.*, 22(4):623–639.

Erdem, E., Gelfond, M., and Leone, N. (2016). Applications of answer set programming. *AI Magazine*, 37(3):53–68.

Erdem, E. and Yeniterzi, R. (2009). Transforming controlled natural language biomedical queries into answer set programs. In *Proceedings of the BioNLP Workshop*, pages 117–124. Association for Computational Linguistics.

Faber, W., Pfeifer, G., and Leone, N. (2011). Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298.

Falkner, A. A., Friedrich, G., Schekotihin, K., Taupe, R., and Teppan, E. C. (2018). Industrial applications of answer set programming. *Künstliche Intelligenz*, 32(2-3):165–176.

Fandinno, J. and Schulz, C. (2019). Answering the "why" in answer set programming - A survey of explanation approaches. *Theory Pract. Log. Program.*, 19(2):114–203.

Ferrara, E. (2023). Fairness and bias in artificial intelligence: A brief survey of sources, impacts, and mitigation strategies. *Sci*, 6(1):3.

Ferschl, M., Tung, A., Sweitzer, B., Huo, D., and Glick, D. (2005). Preoperative Clinic Visits Reduce Operating Room Cancellations and Delays. *Anesthesiology*, 103(4):855–859.

Fox, M. and Long, D. (2003). PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20:61–124.

Frank, S. (2000). Digital health care—the convergence of health care and the internet. *The Journal of ambulatory care management*, 23:8–17.

Fuchs, N. E. (2005). Knowledge representation and reasoning in (controlled) natural language. In *Proceedings of ICCS*, volume 3596 of *LNCS*, pages 51–51. Springer.

Fuchs, N. E. and Schwitter, R. (1995). Specifying logic programs in controlled natural language. *CoRR*, abs/cmp-lg/9507009.

Galatà, G., Maratea, M., Marte, C., and Mochi, M. (2024). Rescheduling master surgical schedules via answer set programming. *Progress in Artificial Intelligence*.

Galatà, G., Maratea, M., Mochi, M., Morozan, V., and Porro, I. (2021). An asp-based solution to the operating room scheduling with care units. In Benedictis, R. D., Maratea, M., Micheli, A., Scala, E., Serina, I., Vallati, M., and Umbrico, A., editors, *Proceedings of the 9th Italian workshop on Planning and Scheduling (IPS'21) and the 28th International Workshop on "Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion" (RCRA'21) co-located with AIxIA 2021*, volume 3065 of *CEUR Workshop Proceedings*. CEUR-WS.org.

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

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Wanko, P. (2016). Theory solving made easy with clingo 5. In *ICLP (Technical Communications)*, volume 52 of *OASICS*, pages 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Gebser, M., Kaufmann, B., and Schaub, T. (2012). Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*, 187:52–89.

Gebser, M., Leone, N., Maratea, M., Perri, S., Ricca, F., and Schaub, T. (2018a). Evaluation techniques and systems for answer set programming: a survey. In Lang, J., editor, *IJCAI*, pages 5450–5456. ijcai.org.

Gebser, M., Obermeier, P., Schaub, T., Ratsch-Heitmann, M., and Runge, M. (2018b). Routing driverless transport vehicles in car assembly with answer set programming. *Theory and Practice of Logic Programming*, 18(3-4):520–534.

Gebser, M., Pührer, J., Schaub, T., and Tompits, H. (2008). A meta-programming technique for debugging answer-set programs. In *Proc. of AAAI*, pages 448–453. AAAI Press.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference and Symposium , Seattle, Washington, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press.

Gelfond, M. and Lifschitz, V. (1991). Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Comput.*, 9(3/4):365–386.

Giunchiglia, E., Maratea, M., and Mochi, M. (2023). A simple proof-theoretic characterization of stable models. In Benedictis, R. D., Castiglioni, M., Ferraioli, D., Malvone, V., Maratea, M., Scala, E., Serafini, L., Serina, I., Tosello, E., Umbrico, A., and Vallati, M., editors, *Proceedings of the the Italian Workshop on Planning and Scheduling, RCRA Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy (IPS-RCRA-SPIRIT 2023) co-located with 22nd International Conference of the Italian Association for Artificial Intelligence AIxIA 2023, November 7-9th, 2023, Rome, Italy*, volume 3585 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Gunning, D., Stefik, M., Choi, J., Miller, T., Stumpf, S., and Yang, G.-Z. (2019). Xai—explainable artificial intelligence. *Science Robotics*, 4(37):eaay7120.

Gurobi Optimization, LLC (2021). Gurobi Optimizer Reference Manual.

Hamid, M., Nasiri, M. M., Werner, F., Sheikhahmadi, F., and Zhalechian, M. (2019). Operating room scheduling by considering the decision-making styles of surgical team members: A comprehensive approach. *Computers & Operation Research*, 108:166–181.

Harnett, M. P., Correll, D., Hurwitz, S., Bader, A., and Hepner, D. (2010). Improving Efficiency and Patient Satisfaction in a Tertiary Teaching Hospital Preoperative Clinic. *Anesthesiology*, 112(1):66–72.

Heching, A. and Hooker, J. (2016). Scheduling home hospice care with logic-based benders decomposition. In *Integration of AI and OR Techniques in Constraint Programming: 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29-June 1, 2016, Proceedings 13*, pages 187–197. Springer.

Heshmat, M. and Eltawil, A. (2021). Solving operational problems in outpatient chemotherapy clinics using mathematical programming and simulation. *Annals of Operations Research*, 298:1–18.

Hooker, J. (2004). A hybrid method for planning and scheduling. In *International Conference on Principles and Practice of Constraint Programming*, pages 305–316. Springer.

Hooker, J. and Ottosson, G. (2003). Logic-based benders decomposition. *Mathematical Programming*, 96(1):33–60.

Huggins, A., Claudio, D., and Pérez, E. (2014). Improving resource utilization in a cancer clinic: An optimization model. In *IIE Annual Conference and Expo 2014*.

Ignatiev, A., Morgado, A., and Marques-Silva, J. (2019). RC2: an efficient maxsat solver. *J. Satisf. Boolean Model. Comput.*, 11(1):53–64.

Janhunen, T. and Niemelä, I. (2016). The answer set programming paradigm. *AI Mag.*, 37(3):13–24.

Kaminski, R., Romero, J., Schaub, T., and Wanko, P. (2023). How to build your own asp-based system?! *Theory Pract. Log. Program.*, 23(1):299–361.

Kuhn, T. (2014). A survey and classification of controlled natural languages. *Comput. Linguistics*, 40(1):121–170.

Landa, P., Aringhieri, R., Soriano, P., Tànfani, E., and Testi, A. (2016). A hybrid optimization algorithm for surgeries scheduling. *Operations Research for Health Care*, 8:103–114.

Li, F., Wang, H., Basu, K., Salazar, E., and Gupta, G. (2021). Discasp: A graph-based ASP system for finding relevant consistent concepts with applications to conversational socialbots. In *Proc. of ICLP Technical Communications*, volume 345 of *EPTCS*, pages 205–218.

Lifschitz, V. (2022). Translating definitions into the language of logic programming: A case study. In *Proceedings of the ICLP Workshops*, volume 3193 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Linardatos, P., Papastefanopoulos, V., and Kotsiantis, S. (2020). Explainable ai: A review of machine learning interpretability methods. *Entropy*, 23(1):18.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA. Curran Associates Inc.

Macario, A. (2010). What does one minute of operating room time cost? *Journal of Clinical Anesthesia*, 22(4):233–236.

Maratea, M., Pulina, L., and Ricca, F. (2014). A multi-engine approach to answer-set programming. *Theory and Practice of Logic Programming*, 14(6):841–868.

Marques-Silva, J. (2024). Logic-based explainability: Past, present & future. *CoRR*, abs/2406.11873.

Marques-Silva, J. and Huang, X. (2024). Explainability is not a game. *Commun. ACM*, 67(7):66–75.

Marques-Silva, J. and Ignatiev, A. (2023). No silver bullet: interpretable ML models must be explained. *Frontiers Artif. Intell.*, 6.

Martins, R., Manquinho, V. M., and Lynce, I. (2014). Open-wbo: A modular maxsat solver,. In *SAT 2014*, volume 8561 of *LNCS*, pages 438–445. Springer.

Marynissen, S., Heyninck, J., Bogaerts, B., and Denecker, M. (2022). On nested justification systems. *Theory Pract. Log. Program.*, 22(5):641–657.

Masroor, F., Gopalakrishnan, A., and Goveas, N. (2024). Machine learning-driven patient scheduling in healthcare: A fairness-centric approach for optimized resource allocation. In *2024 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 01–06. IEEE.

McCluskey, T. L. and Porteous, J. M. (1997). Engineering and compiling planning domain models to promote validity and efficiency. *Artif. Intell.*, 95(1):1–65.

McCluskey, T. L., Vaquero, T. S., and Vallati, M. (2017). Engineering knowledge for automated planning: Towards a notion of quality. In *Proceedings of K-CAP*, pages 14:1–14:8.

Mercier, L. and Van Hentenryck, P. (2008). Edge finding for cumulative scheduling. *INFORMS Journal on Computing*, 20(1):143–153.

Meskens, N., Duvivier, D., and Hanset, A. (2013). Multi-objective operating room scheduling considering desiderata of the surgical team. *Decis. Support Syst.*, 55(2):650–659.

Miller, T. (2017). Explanation in artificial intelligence: Insights from the social sciences. *CoRR*, abs/1706.07269.

Mittelstadt, B. D., Russell, C., and Wachter, S. (2018). Explaining explanations in AI. *CoRR*, abs/1811.01439.

Mochi, M., Galatà, G., and Maratea, M. (2023). Master surgical scheduling via answer set programming. *J. Log. Comput.*, 33(8):1777–1803.

Morgado, A., Dodaro, C., and Marques-Silva, J. (2014). Core-Guided MaxSAT with Soft Cardinality Constraints. In *CP 2014*, pages 564–573, Lyon, France. Springer.

Nuijten, W. W. (1994). Time and resource constrained scheduling : a constraint satisfaction approach.

Oetsch, J., Pührer, J., and Tompits, H. (2010). Catching the ouroboros: On debugging non-ground answer-set programs. *Theory Pract. Log. Program.*, 10(4-6):513–529.

Oetsch, J., Pührer, J., and Tompits, H. (2011). Stepping through an answer-set program. In *Proc. of LPNMR*, volume 6645 of *LNCS*, pages 134–147. Springer.

Olivier Roussel and Vasco Manquinho (2012). Input/Output Format and Solver Requirements for the Competitions of Pseudo-Boolean Solvers.

Oswald, J., Srinivas, K., Kokel, H., Lee, J., Katz, M., and Sohrabi, S. (2024). Large language models as planning domain generators. In *34th International Conference on Automated Planning and Scheduling*.

Öztok, U. and Erdem, E. (2011). Generating explanations for complex biomedical queries. In *Proceedings of AAAI*. AAAI Press.

Öztürkoğlu, Y. (2020). A different approach to nurse scheduling problem: Lagrangian relaxation. *Alphanumeric Journal*, 8(2):237–248.

Perri, S., Ricca, F., Terracina, G., Cianni, D., and Veltri, P. (2007). An integrated graphic tool for developing and testing dlv programs. In *Proc. of SEA*, pages 86–100.

Polleres, A., Frühstück, M., Schenner, G., and Friedrich, G. (2013). Debugging non-ground ASP programs with choice rules, cardinality and weight constraints. In *Proc. of LPNMR*, volume 8148 of *LNCS*, pages 452–464. Springer.

Pérez, E., Ntaimo, L., Wilhelm, W. E., Bailey, C., and McCormack, P. (2011). Patient and resource scheduling of multi-step medical procedures in nuclear medicine. *IIE Transactions on Healthcare Systems Engineering*, 1(3):168–184.

Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2017). The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817.

Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., and Leone, N. (2012). Team-building with answer set programming in the Gioia-Tauro seaport. *Theory and Practice of Logic Programming*, 12(3):361–381.

Saikko, P., Berg, J., and Järvisalo, M. (2016). LMHS: A SAT-IP hybrid maxsat solver. In *SAT 2016*, volume 9710 of *LNCS*, pages 539–546. Springer.

Scanu, M., Mochi, M., Dodaro, C., Galatà, G., and Maratea, M. (2023). Operating room scheduling via answer set programming: The case of ASL1 Liguria. In Dovier, A. and Formisano, A., editors, *Proc. of the 38th Italian Conference on Computational Logic (CILC 2023)*, volume 3428 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Schüller, P. (2018). Answer set programming in linguistics. *Künstliche Intelligence*, 32(2-3):151–155.

Schulz, C., Satoh, K., and Toni, F. (2015). Characterising and explaining inconsistency in logic programs. In *Proc. of LPNMR*, volume 9345 of *LNCS*, pages 467–479. Springer.

Schulz, C. and Toni, F. (2016). Justifying answer sets using argumentation. *Theory Pract. Log. Program.*, 16(1):59–110.

Schwitter, R. (2018). Specifying and verbalising answer set programs in controlled natural language. *Theory Pract. Log. Program.*, 18(3-4):691–705.

Schwitter, R., Hamburger, B., and Fuchs, N. E. (1995). Attempto: Specifications in controlled natural language. In *Proceedings of the Workshop on Logische Programmierung*, pages 151–160.

Shchekotykhin, K. M. (2015). Interactive query-based debugging of ASP programs. In *Proc. of AAAI*, pages 1597–1603. AAAI Press.

Smith, T., Evans, J., Moriel, K., Tihista, M., Bacak, C., Dunn, J., Rajani, R., and Childs, B. (2022). Cost of or time is $46.04 per minute. *Journal of Orthopaedic Business*, 2:10–13.

Stark, C., Gent, A., and Kirkland, L. (2015). Improving patient flow in pre-operative assessment. *BMJ Open Quality*, 4(1).

Syrjänen, T. (2006). Debugging inconsistent answer set programs. In *Proc. of NMR*, volume 6, pages 77–83.

Tariq, H., Ahmed, R., Kulkarni, S., Hanif, S., Toolsie, O., Abbas, H., and Chilimuri, S. (2016). Development, functioning, and effectiveness of a preoperative risk assessment clinic. *Health Services Insights*, 2016:1.

The Business Rules Group (2000). Defining business rules $\sim$ what are they really?

Ueda, D., Kakinuma, T., Fujita, S., Kamagata, K., Fushimi, Y., Ito, R., Matsui, Y., Nozaki, T., Nakaura, T., Fujima, N., et al. (2024). Fairness of artificial intelligence in healthcare: review and recommendations. *Japanese Journal of Radiology*, 42(1):3–15.

Vallati, M. and Chrpa, L. (2019). On the robustness of domain-independent planning engines: The impact of poorly-engineered knowledge. In *Proceedings of K-CAP*, pages 197–204.

Woodrum, C. L., Wisniewski, M., Triulzi, D. J., Waters, J. H., Alarcon, L. H., and Yazer, M. H. (2017). The effects of a data driven maximum surgical blood ordering schedule on preoperative blood ordering practices. *Hematology*, 22(9):571–577.

World Health Organization (2018). Classification of digital health interventions v1.0: a shared language to describe the uses of digital technology for health. Technical documents, World Health Organization.

World Health Organization (2021). *Global strategy on digital health 2020-2025*. World Health Organization.

World Health Organization (2023). *Classification of digital interventions, services and applications in health: a shared language to describe the uses of digital technology for health*. World Health Organization, 2nd ed edition.

World Health Organization (2024). *Artificial Intelligence for Health*. World Health Organization.

Xiao, Q., Luo, L., Zhao, S., bin Ran, X., and bing Feng, Y. (2018). Online appointment scheduling for a nuclear medicine department in a chinese hospital. *Computational and Mathematical Methods in Medicine*, 2018.

Zhang, J., Dridi, M., and El Moudni, A. (2017). A stochastic shortest-path MDP model with dead ends for operating rooms planning. In *ICAC*, pages 1–6. IEEE.

Zhang, Z. and Xie, X. (2015). Simulation-based optimization for surgery appointment scheduling of multiple operating rooms. *IIE Transactions*, 47(9):998–1012.

Şeyda Gür and Eren, T. (2018). Application of operational research techniques in operating room scheduling problems: Literature overview. *Journal of Healthcare Engineering*, 2018.