Università degli Studi di Padova

Master Degree in Computer Science Engineering

Academic Year 2010/2011

Dynamic Constraint Satisfaction approach to Hospital Scheduling Optimization

Thesis director:	Prof. Silvana Badaloni
Thesis co-director:	Ph.D. Francesco Sambo
Student:	Marco Ventilii (matriculation: #604088)

GRAZIE DI CUORE...

...alla professoressa Silvana Badaloni, per aver creduto in me quando nemmeno io lo facevo e per avermi sempre spinto a dare il massimo.

...al dottor Francesco Sambo, per la sua incredibile disponibilità ad aiutare e per il sorriso con cui l'ha sempre fatto.

...alla mia famiglia, Rita, Elio e Francesco, per avermi sempre compreso e sopportato, senza mai, *mai* avermi fatto sentire sotto pressione in alcun modo.

...alla mia dolce Elena, per avermi fatto sorridere di gioia in questi mesi, ridandomi la serenità necessaria a continuare.

...a Enrico, Federica, Mariarosa e Silvia, insostitubili amici, per avermi assistito quando ne avevo veramente bisogno.

...ai compagni di corso: Michele, Martina, Roberta, Federico, Andrea, Paolo, Manuel, Cimut (non riesco mica a chiamarti Simone, oh!), Gianmario e il resto della classe, per aver reso speciali questi cinque anni e mezzo (e per le partite ad Unreal Tournament).

...agli amici di sempre: Filippo, Alessandro, Elisa, Manuel, Valeria, Matteo, Manuela, Edoardo, Daniele e tutti gli altri, per le innumerevoli esperienze significative condivise, che ricordo sempre con un gran sorriso, e per l'affetto che mi avete regalato finora.

...ai murialdini, specialmente Jacopo, Marco, Michele, Gaetano, e ai disertori, Lele Andrea e Carlo per aver reso il Murialdo un posto felice e pieno di zanzare.

...e infine ad Angelica, che mi ha insegnato che una delle conquiste più importanti per una persona è trovare il coraggio di volersi bene.

HEARTFELT THANKS...

...to Prof. Silvana Badaloni, for believing in me even when I didn't and for always pushing me to do my best.

...to Ph.D Francesco Sambo, for his incredible willingness to help and for the smile with which he has always done it.

...to my family: Rita, Elio and Francesco, for having always understood me and tolerated me, without never, ever put myself under pressure in any way.

...to my sweet Elena, for the smile of joy that was on my face in these months, giving me the serenity which I needed to carry on.

...Enrico, Federico, Mariarosa and Silvia, irreplaceable friends, for having assisted me when I really needed to.

...to classmates: Michele, Martina, Roberta, Federico, Andrea, Paolo, Manuel, Cimut (sorry, I really can't call you Simone), Gianmario and the rest of the class, for making these five and a half years special (and for Unreal Tournament matches).

. . . to the friend of all time: Filippo, Alessandro, Elisa, Manuel, Valeria, Matteo, Manuela, Edoardo, Daniele and everyone else, for the many significant experiences we shared, always remembered with a big smile, and for the love you have given me so far.

. . . to the college buddies, especially Jacopo, Marco, Michele, Gaetano, and to the deserters Lele , Andrea e Carlo, for making the Murialdo a happy place, full of mosquitoes.

. . . to Angelica, who taught me that one of the most important achievements for a person is to find the courage to love himself.

. . . to the reader, of course, for tolerating my English. I apologize from the start, buddy.

CONTENTS

1 INTRODUCTION

A hospital is a very complex environment: if we think how many people and resources are involved to maintain this facility fully operative we can easily lose the count while just listing the nurses. And all these people and resources, which includes the staff, the machineries, even the rooms of the hospital itself, always cooperate in order to to administer the right treatments to the patients while trying to maintain a good overall performance and, why not, trying also to satisfy patients requests in term of organization, personalized services and so on. Such a complex environment is surely difficult to handle: up to now the management policies of hospitals have established an acceptable quality model, which has become a standard de facto for Minimum Service Levels (**MSL**s).

Based on this we can immediately focus our attention on the core of the *Hospital Scheduling Optimization* problem: a hospital with an incoming stream of patients, each one needing a personalized medical care treatment, must optimize the way it serves them, trying to administer efficiently the majority of the treatments needed by all of them, keeping in mind that a fixed MSL has to be granted. With these premises it is not difficult to think that a human manager can be outperformed by a fully automatized software: after all the mission-critical goal is clear and easily explainable from the mathematical point of view : the treated patient throughput has to be maximized.

Let focus first our attention on each medical treatment a patient needs, considered as a single process; we know that this process requires some resources to be allocated (a patient requires at least a doctor to proceed toward his medical treatments and, in some cases, also certain devices or machineries are needed) and, once allocated, they need to be retained for some time while making progress to the end. With the same approach we can decompose a medical treatment in basic sub-treatments, each one to be performed by a different specialist, with precedence constraints between them.

With the similarities we have just established the scheduling problem archetype clearly fits in our case.

Such a complex environment can not avoid to arouse interests among the scientific community: a great number of studies has, in fact, already started to appear in the scientific literature, each one analyzing a single problem from the hospital operative cycle and, in some cases, a resolutive approach is proposed. Examples of these correlated studies are a nurse rostering optimization method[18] which can account staff preferences in the planning process, which is a common open problem in hospitals, or a study on how to improve the survival ratio of out-of-hospital cardiac arrests by optimizing existing defibrillation programs [17].

However, while most of the single parts of the hospital are already accounted in separate studies, the real challenge has become interesting only recently: a single hospital can aspire to optimize its whole structure by managing its resources in a way to improve the performances in term of served patient throughput and general satisfaction of patients and staff members. The studies in this sense are still rare among literature but they are all recent [25, 8, 23, 12] and propose resolutive methods we will analyze later in this thesis.

We choosed to use the *Dynamic Constraint Satisfaction Problems* (**DCSP**) as resolutive framework for a number of reasons.

First of all we must take into account that an hospital is a dynamic environment. An agent who finds itself in a hospital can easily expect a wide variety of changes, even if he does not interact with anything: an emergency could happen, medics can be picked away from their rooms and called for consultation, machinery can stop working, a patient's conditions can worsen, giving him/her a bigger priority on the scheduling, and so on.

Therefore, any resolutive algorithm we may propose to optimize the patient throughput must be able to perceive and handle such changes in a clever way.

Secondly, despite a wide literature on scheduling problems has been written over time, conventional approaches are not useful to our problem: according to [2, 19], in fact, the Hospital Optimization problem is a NP-Hard problem and can not be resolved within a reasonable timeframe with these methods.

Artificial intelligence, on the other hand, presents a wide variety of method to approach an intractable problem, like this one. In particular, constraint networks are a versatile, high-level instrument which allow very detailed modeling construction, while permitting different types of implementations.

In this thesis we will see how a DCSP-based method can substantially produce a good scheduling for every patient while retaining a sufficient performance level to make this scheduling approach useful for actual hospital organization.

Specifically:

- In chapter 2 we will see more about the hospital optimization problem
- In chapter 3 we will discuss about the current solutions proposed to this problem
- In chapter 4 we will learn about the DCOP approach
- Through chapter 5 we will examine strictly the solution proposed by this thesis
- In chapter 6 we will show some results obtained with the proposed solution, comparing them with the standard scheduling policies now adopted in hospitals.
- In Chapter 7 conclusions are drawn

2 PROBLEM

In this chapter we will analyze the reality that make the problem itself interesting: the hospital and the medical reality around it.

After all why may a hospital want to optimize its activity planning? And, even if there was the desire to do so, how can this goal be achieved?

To answer the first question we must take a closer look to the hospital environment and to the medical processes that take place into it and only after that we can give an answer also to the second question, by remarking the key aspects of the problem.

2.1 PROBLEM DOMAIN

A hospital is a concept which is easy to imagine: it is simply a place where unhealthy people enter, after a few standard procedures (like taking an appointment, being subject to a diagnostic exam, and so on) they receive the correct treatment for their illness and, finally, they leave.

The structure of the hospital itself, however, is really complex: a hospital is organized in various departments, which have a remarkable organizational and decision-making autonomy. A perfect example of this autonomy is given by the possibility for a specialist to fix directly an appointment to one of his (or her) patients, rather than simply prescribe the appropriate treatment. Such departments will obviously pursue their own (local) goals while optimizing their patient flow, which are probably different from the hospital (overall) goals.

Such a segmented and distributed nature of the hospital environment naturally affects the flow of the patients into the structure, knowing that each patient can interact with more than one department during their stay in the hospital, due to the prescriptions they were given. This implies that a widespread communication network among the various departments is required just to try to optimize the flow of the patients, in order to achieve an optimal use of the available resources.

2.1.1 Patients

Patients are very heterogeneous, because their requests and characteristics can vary widely.

For example, we have patients who reserved a visit in advance (they are defined as *"clinic patients"*) and those who need an emergency assistance (defined as *"urgent patients"*), arriving directly to the ER; another taxonomy of the incoming patients consists into dividing them between patients who need to be hospitalized (*"inpatients"*) and patients who can go home after the needed treatment (or treatments) have been given (*"outpatients"*).

In addition, patients clearly differ by the urgency level of their treatments and by their willingness (or possibility) to wait inside the hospital for their turn to be treated.

In particular this last aspect of the patient has a remarkable influence on the treatments planning: the patient is a required resource for all of his/her appointments,after all. The timespan that the patient can spend in hospital depends on various reasons, which may include professionals (a worker has to request a period off to his boss) or logistics (a person who travel often will surely have difficulties with the treatment planned at his hospital) restrictions. Furthermore, the time available for the patient to wait for his treatment may, obviously, be restricted for medical reasons.

In the Italian health care system, for example, medical prescriptions[1] can be associated with a certain priority level, which implies a restricted due date for the proposed treatment[2]. The specialist doctors which perform these treatment will subsequently compile a report of the actual severity of the patient conditions, to give it as a feedback to the general practitioner (**GP**) who fixed the original priority level, and, as already mentioned, they can handle the rest of the patient medical course, by prescribing further necessary treatments and fixing their appointments directly.

Hospital emergency room (**ER**) behaves differently: patients that arrive at ER does not pass by a GP and has to be diagnosed as soon as possible. This process can be speeded up if an ambulance is called for the emergency: a summary description is given to the centralized phone center, which coordinates ambulance exits, and some staff members can be sent directly to the patient location with the ambulance itself; this permits a preliminary diagnosis on the ambulance and gives to the ER a short time period (of at least 10 minutes) for preparations. After the diagnosis is made an emergency code is associated to the patient[3]and upon this code a due date for the patient can be determined. In the ER these dates are usually very strict, thus the other hospital departments can be influenced by these arrivals in various ways: a doctor

1 In the Italian health care system, medical prescriptions are given by the general practitioner, thus they are not treated directly by the ER (source : http://http://www.salute.gov.it/)

2 for example a **U** priority level (which stands for **U**rgent) means that the treatment must be given in the next 24 hours after the prescription delivery. A **D** level (which stands for **D**elayable), on the contrary, sets a due date of 30 days from the prescription delivery for diagnostic examinations and of 60 days for instrumental examinations. The lower assignable priority level is the **P**rogrammed one, which fix the due date to 3-6 months for all the examinations except the oculistic ones, which have a due date of 18 months. (source : http://http://www.salute.gov.it/)

3 Italian health care system provides four codes, each one correspondents to a color. Ordered from the less urgent one they are white (not urgent at all), green, yellow and red (life-threatening emergency). (source : http://http://www.salute.gov.it/)

may be called for an advice by the ER or an urgent X-Ray scan is committed; these event, therefore, are capable of breaking up all the previous planning with their prioritary resource requests[4].

2.1.2 Resources

Hospitals continuously aim to improve their patient-oriented care. They want to provide their patients with high service levels. However, the demand for health care is increasing, and more patients must be treated with the same capacity. High efficiency on resources is necessary for high service levels. Traditional approaches to logistical improvement are usually not suited to the medical domain. The distributed authority in hospitals makes improvements involving many departments difficult to implement. Furthermore, scheduling decisions must be made depending on the individual patient's specific attributes. Efficient scheduling of patient appointments on expensive resources is a complex and dynamic task [26].

Hospital resources are many: ranging from CT and MRI scanners, to hospital beds, to attending staff. A resource is typically used by several patient groups with different properties. There are groups of inpatients (admitted to the hospital) and outpatients (not admitted), with different levels of urgency. The total hospital resource capacity is allocated to these groups, explicitly or implicitly. Either way, due to fluctuations in demand, this allocation must be flexible to make efficient use of the resources. To allocate hospital resources, electronic calendar-systems are widely applied. However, they are mostly just storing the patient appointments.[22]

The resources of a hospital comprise the staff, the machinery and the rooms, and all of them are limited resources with a finite capacity: some of them, like the staff and many machineries, can participate only to one appointment at time, while a room may accommodate more peoples simultaneously.

In particular, the hospital staff is a remarkably valuable resource: the solution to our problem can not be as simple as hiring more staff but, on the other hand, the doctors surely can not be asked to work twice as many hours as planned. Neither they can be asked to work frantically only to respect the schedule, because a similar request may lower the quality of the given treatments, with consequent inconveniences to patients.

The hospital staff, thus, should and must take benefits from the reorganization of the planning method.

Machineries, on the other hand, require qualified staff to work but, apart from this, they are a more flexible resource compared to the staff.

It is, clearly, very difficult that a patient will be available in the middle of the night for a simple physiotherapy session (and it is even more difficult to find a doctor willing to do such a

4 Usually an engaged resource can not be released in hospital environment *but* a preemption is not excluded, even if it happens rarely.

session), but those devices that are useful for emergencies (like X-Ray fluoroscopies or CAT-scan machines) can potentially operate 24/7, with the necessary precautions, and often they do so, since it is likely that a qualified team is always ready to supervise those machineries, alternating the shifts with other members of the staff.

A medical device is usually quite difficult to move around the hospital, thus we can reasonably associate a device with the room that houses it. It is possible for a room, though, to participate in more appointments at the same time, while a medical device can not attend more than one patient simultaneously. With this assumption we can establish a many-to-one correspondence between the machine and the room it is contained into; this means that the model can merge machinery and rooms into a single entity, explained in the next subsection.

2.1.3 Definitions

Given the intrinsic complexity of this problem the first step is the definition of the key terms.

PATIENT. A patient is a person who asks the hospital for a medical treatment. We have already discussed the aspects and the behaviour of patients.

TREATMENT. A treatment is given by a member of the staff and may require particular medical devices.

WORKPLACE. A workplace is a space where a treatment can be administered. A workplace represent the many-to-one correspondence already mentioned just above, in Section 2.1.2; a workplace, then, can hold more than one device and is therefore able to participate in the administration of various types of treatment. A workplace, however, holds a scarce resource (i.e.: the machinery) and thus is considered, in turn, a scarce resource. Due to the rarity of preemptive requests, which always are consequent to high-level decisions, our model will not permit any workplace preemptions.

APPOINTMENT. An appointment is the request by a patient for a certain treatment. The hospital must try to satisfy this request by assigning to this appointment a particular time instant in which the appropriate resources (doctors, machineries, etc. . .) will be available and the requested treatment can acquire those resources and be administered.

ASSIGNMENT. An assignment is the set of details needed to satisfy an appointment request. In particular an assignment is composed by

- **A starting time** for the treatment to be administered

- **A workplace** in which the treatment will be administered

This information is given for the only purpose to locate, in time and space, a slot in which the resources needed by the requested treatment are available. If an appointment is provided with these data it is said that the appointment has been *assigned*.

2.1.4 Particular issues

Festivities have also to be considered with great attention: during holidays and Saturdays every non-urgent outpatient activity is suspended and the staff present into the hospital is greatly reduced. This means that the emergencies can still be treated but on Mondays an increased flux of requests is expected and will heavily influence the appointments that have already been assigned.

2.2 COMPUTATIONAL ASPECTS

This section will focus on the scheduling optimization process, examining the key issues which will arise during the implementation step.

The first thing that one encounters when seeking medical assistance in a hospital is a schedule: the scheduled medical professionals to consult, time-slots for possible diagnostic or therapeutic machines, and availability of simple resources like examination rooms. Depending on the available capacity, these schedules may be more or less congested. In particular, in countries like The Netherlands and Greece, demand regularly exceeds capacity and substantial waiting lists exist for many medical procedures. [25] However, even in these situations, it is not the case that resources are always used at high efficiency. Medical professionals report many schedule inefficiencies. Effective scheduling algorithms should decrease waiting lists significantly, while increasing hospital efficiency [27, 5].

2.2.1 Optimization level vs Usefulness

Due to the distributed nature of a hospital, departments have local objectives and scheduling policies [23]. The problem of scheduling a mix of patients with varying properties has to be solved locally, while hospital-wide performance depends on how departments interact with each other. To make efficient use of the resources we have already delineated an appointment-based systems which, despite its simplicity, reflects with great fidelity the one actually used in reality, although in current practice the actual scheduling is often done by hand.

However, a scheduling algorithm that aims at optimizing the hospital scheduling must satisfy, first of all, a basic requirement:

the whole optimization process must be done in an amount of time short enough to make the solution still useful.

In other words, the computation time of the algorithm *must not* exceed the unit chosen to quantize the time flow[5]; the solutions, otherwise, become useless for the planning process.

2.2.1.1 *NP-Hardness.*

This problem is referable to the more general "open shop" problem [4].

In open shop problems, all jobs (here patients' appointments set) consist of as many activities (the appointments contained int the aforementioned set) as there are resources.

Our patient scheduling problem therefore corresponds to an open shop problem with processing times including values of zero: $O|p_{ij} = \{0, sat_{m_{ij}}\}|\sum_i C_i$ (standard scheduling notation), where p_{ij} is the processing time of activity a_{ij} , and $sat_{m_{ij}}$ is the standard activity time of resource m_{ij} [25].

The general open shop problem is known to be NP-Hard[10, 14] thus we can not expect that optimal solutions are obtainable within a reasonable time for this problem either.

This obviously hardens the challenge of restricted times already explained in the previous paragraph.

2.2.2 Environment

A clear aspect of our problem is that the hospital environment is clearly dynamic.

A rational scheduler that has to plan a temporal sequence of appointments acceptance must react to the changes that can occur during the hospital activity period. In particular we can observe that the frequency of these changes is unpredictable: the environment is, therefore, a stochastic dynamic environment.

Due to the nature of our environment we can state that these change are unpredictable but gradual: it is unlikely that a mass patient resignation happens at the same time of a big emergency; therefore we can assume that if we would take two snapshot of the environment's state, one before and the other after a certain change, we should be able to spot a very small number of differences.

A solution found, thus, is not stable in time but a new solution is very likely to be constructible basing on the information contained in the old one.

5 In our simulations, explained in chapter 6, this time unit is the quarter of an hour, which is enough to organize the appointments with a significant precision. See also Definition 5.1.

3 STATE OF THE ART

This section will present the state of the art for the problem of Hospital Scheduling Optimization.

Unfortunately, as already mentioned, in the literature similar works are still rare: while we can find a lot of studies on open problems related to the health care system there are still only a few approaches for this kind of problem.

We will briefly examine each of these approaches, analyzing its underlying idea, its structure and summarizing any pros and cons.

3.1 DISTRIBUTED CONSTRAINT OPTIMIZATION FOR MEDICAL APPOINTMENT SCHEDULING

The study described in [12] is the one that comes closest to our approach in terms of resolving approach: in fact it builds a multi-agent approach on an extension of *Constraint Satisfaction Problems* (CSP) settings[1] which is more suitable to collaborative problem solving, called *Distributed CSP*.

3.1.1 Description

This study starts by defining a very detailed model: this is a model which is very close to actual reality and covers a wide range of attributes of patients and resources. Also, constraints, variables and domains are rigorously described, covering most of the significant aspects of hospital reality.

This model, being so rich in detail and fitting for a constraint satisfaction approach, is the one we adopted, and our approach will be based upon it. Both the approach and the model, thus, are described in detail in chapter 5 and will not be referred in this section anymore.

Once defined the model, the agents which operate into it are also defined and, finally, the criteria adopted by the agents for local optimization can be described. With this reasoning, the model is further enriched with details, resembling even more the actual nature of the hospital: a decentralized, distributed set of departments, whit a great decision-making autonomy.

After this the study goes on describing a distributed solving approach used to build the solution by collaborating agents: in particular this method uses the *Multi-Phase Agreement Finding* (**MPAF**) [11] algorithm to incrementally assign a value to the

1 CSPs will be described in chapter 4 of this work

variables, paying attention not to violate any of the constraints specified by the model.

In the two subphases of this algorithm (which are a proposal phase and an assignment phase) *optimization objects* are involved in the process: these objects encapsulate CSP knowledge about optimization criteria and strategies and have the ability to transform themselves to declarative CLP expressions, which actually build a piece of a CLP problem declaration.

The final step of the algorithm is to collect and organize all these pieces, scattered among the various agents, and recompose the whole CLP problem, which will be given to an off-the-shelf constraint solver.

3.1.2 Pros and cons

This study, as already stated, presents a realistic and detailed model of the hospital reality, including variables and constraints, which suits perfectly a CSP-based approach. With the additional information contained in the model this study aims to distribute the problem in a way very similar to the one actually used in real hospitals, although it still requires a central solver to perform the last computation step.

However, this work misses a critical point in the modelling phase: the hospital environment described in this study, in fact, is *not* a dynamic environment. Such a modelling choice relieves the algorithm from the burden of having to handle those particular events that characterize an actual hospital environment. The appointments, thus, can be assigned to certain workplaces in the appropriate time slots, respecting the optimization criteria, but they can not be modified after that. This makes this approach of limited utility for our final goal, apart from the model: as already mentioned, in fact, the model used in this study reflects the reality with a high fidelity level, therefore is the one we used.

3.2 MULTI-AGENT PARETO APPOINTMENT EXCHANGE (MPAEX)

The Multi-agent Pareto Appointment EXchange (**MPAEX**) [25] approach is a distributed multi-agent system designed by Ivan Vermeulen et al. capable of managing the scheduling of an hospital dynamic environment.

3.2.1 Description

The main idea beyond this approach is that every patient is represented by an agent that tries to exchange its reserved time slots (obtained with the appointment acceptance) with one of the other agents following the marketplace model, which is an efficient way to distribute scarce resources: in fact similar models integrate well in dynamic environments and minimize the required communications between participants, because only the price-quotations must be exchanged in order for the bids to take place.

In this way a patient will exchange his appointments only if this reflects his preferences; an exchange of time slots, thus, will be done only if none of the parts will suffer a satisfaction loss[2].

In such a framework, the real patients are induced to cooperation by the instructions of their software agents.

This method is conceived over the knowledge of the consult-diagnostic-consult cycle, represented in figure 1, which includes various "consultation checkpoints" that allow the doctor to decide any other test to be administered to the patient, basing on the set of results obtained up to that moment.

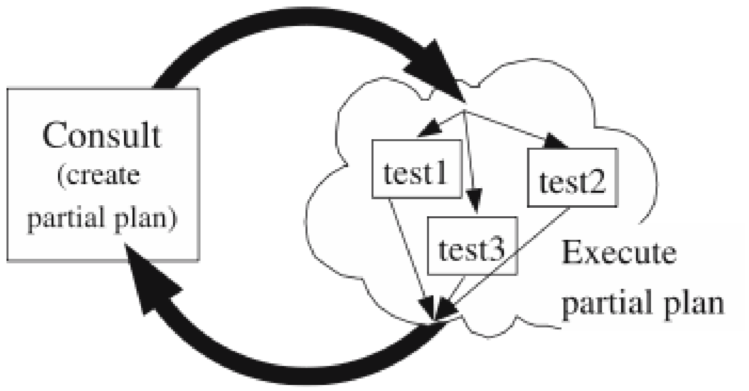


Figure 1: Consult-diagnostic(s)-consult cycle

This cycle suggests an incremental building of patient's treatments, divided among different days, but with a single partial plan fixed at once. The target of this approach is to minimize the patient waiting time, defined as the period of time between the final activity of the partial plan and the creation of the partial plan itself; this means that only the final appointment of the plan can actually change the patient's satisfaction, thus every other appointment can be moved freely without worsen nor improving the patient status.

This implies that a patient agent will always try to move the last appointment of the plan, in order to optimize the patient scheduling, asking the correspondent resource agent a way to reduce the plan's completion time. Resources agents will then propose an alternative time slot, starting from the earliest possible time slot, and the patient agent will try to exchange his/her time slot with the patient-agent occupying that time slot. The deal will be accepted if neither patient suffers a worsening of its situation with respect to its preferences (i.e. its completion time will not increase). If not accepted, the patient agent will request another prospective time slot from the resource agent, and will continue

2 In economics such a "nobody-worse" improvement is called a "Pareto improvement", which gives the name to the algorithm

doing so until there are no more prospective time slots, or a proposed exchange is accepted. The process is repeated for all patient agents iteratively, until no exchanges can be made any more.

This method is made to improve an already existing schedule, which is compiled by a very simple algorithm: for every unscheduled activity in the partial plan a time slot is asked to the correspondent resource, which is promptly given; if this time slot is not conflicting with any other assignment it is accepted by the patient's agent otherwise the resource agent is prompted for a new time slot.

3.2.2 Pros and cons

This decentralized approach is able to manage admirably a distributed system in a dynamic environment and his Pareto-improvement-based behaviour will surely facilitate its adoption by the patients and the hospital staff.

The method, unfortunately, lacks of an efficient emergency management: in an emergency case every patient will have his or her schedule's score worsened, thus will never accept an exchange with the urgent patient's agent.

This can easily be solved by forcing a time slot exchange with the non-urgent patient's agent, even if it worsen up its schedule. This agent will subsequently try to improve its scheduling with the usual algorithm. This case, however, is not covered in the article.

This approach, therefore, is surely good for scheduling outpatients with restricted availability set but any emergency management must be studied separately.

3.3 COORDINATED HOSPITAL PATIENT SCHEDULING

This works [8] by Keith Decker and Jinjiang Li presents another multi-agent system, based on the Generalized Partial Global Planning (**GPGP**) approach that preserves the existing human organization and authority structures, while providing better system-level performance (increased hospital unit throughput and decreased patient stay time).

3.3.1 Description

Generalized Partial Global Planning (GPGP) is an approach to coordinated problem resolution centered on task's environment. The basic idea is that each agent constructs its own local view of the structure and relationships of its intended tasks. This view may then be augmented by information from other agents, and it may change in other ways dynamically over time. The

GPGP approach uses a set of individual coordination mechanisms to help to construct these partial views, and to recognize and respond to particular task structure relationships by making commitments to other agents. These commitments result in more coherent, coordinated behavior. No one coordination algorithm will be appropriate for all task environments, but by selecting from a set of possible coordination mechanisms we can create a wide set of different coordination responses.

The set of intended task is represented by using a particular task structure representation language (**TÆMS**), which allows the formulation of specification of dynamically changing and uncertain task characteristics that effect an agent's preferences for some state of the world, including tasks with hard or soft deadlines (which have not been included in the model, despite the GPGP's capability of handling them.). A **TÆMS** specification also indicates relationships between local and non-local tasks or resources that effect these agent preference characteristics. An agent using the GPGP approach provides a planner or plan retriever to create task structures that attempt to achieve agent goals, and a scheduler that attempts to maximize utility via the choice and temporal location of basic actions in the task structure. Each GPGP mechanism examines the changing task structure for certain situations, such as the appearance of a particular class of task relationship, and responds by making local and non-local commitments to tasks, possibly creating new communication actions to transmit commitments or partial task structure information to other agents. The set of coordination mechanisms is extensible, and any subset or all of which can be used in response to a particular task environment situation

To pursue the objectives listed in the study an extension to the GPGP approach has been made, including a new coordination mechanism able to handle mutually exclusive resource relationships[3].

The new mechanism uses a simple multi-round but not multi-stage negotiation process which is not optimal but has good "flow" properties since at least one agent is free to stop meta-level communication and begin domain work at each round. When several agents try to use the same non-sharable resource at overlapping times, only one agent can actually get the resource and execute its work. The others who failed to get the resource waste this time unit and this effort. The idea behind the resource-constraint coordination mechanism is that when an agent intends to execute a resource-constrained task (i.e. the task is scheduled), it sends a directed bid of the time interval it needs and the local priority (expressed as the effect on local utility) of its task (we will describe how this is computed later). After a communication delay, it knows all the bids given out by the other agents at the same time as its own bid. Since all the agents who bid will have the same information, if they all use the same commonly

3 This new mechanism, like the other GPGP mechanisms, can be applied to any problem with the appropriate resource relationship.

accepted rule to decide who will get the time interval, they can get the same result on this round of bidding. The agent who won will keep its schedule and execute that task at the time interval it bid, and everyone else will mark this time interval with a `DON'T` commitment and never try to execute a related resource constrained task in it unless the owner gives it up. All the agents who did not get their time intervals at this round will then reschedule and bid again, while continue to monitoring the just missed time-slot, in case that the agent who won that may give it up.

3.3.2 Pros and cons

The study actually proves that cooperating agents perform better than simple (competing) agents from an overall point of view. GPGP, also, can handle a wide variety of relation types, which can model task uncertainty, precedence constraints, preferences, and so on, adding value to the final optimized scheduling.

However, the overall performance of this approach is worse than the one of a centralized system.

The study begins modeling the hospital environment but this model presents a couple of heavy restrictions:

1. No redundant resources are modeled. This implies that two interchangeable resources are different for the model, thus they can not be swapped between patients which require a treatment that can be administered by both of them.

2. Deadlines are not included into the model: this is a really dangerous assumption because without deadlines (hard or soft) a patient can go on starvation[4]. Also by not having any deadline constraint associated to a treatment it is impossible to estimate an accurate metric of treatment urgency, which has to be separately specified.

Also, the study focus more on remarking GPGP versatility than on design of an effective hospital scheduling optimization algorithm, beginning with the strict model used that is just explained, which may not fit well in an actual hospital reality.

3.4 OPTIMIZATION OF ONLINE PATIENT SCHEDULING WITH URGENCIES AND PREFERENCES

This study done by Vermeulen et al. [23] considers the online problem of scheduling patients with urgencies and preferences on hospital resources with limited capacity. To solve this complex scheduling problem effectively the problem has been split into the following sub problems: determining the allocation of capacity

4 A job which, for its low priority, is always postponed and is never served for a long time is called to be on *starvation*[13]. This is an undesirable case and, to be avoided, a commonly taken action is to raise the starving job priority

to patient groups, setting dynamic rules for exceptions to the allocation, ordering time slots based on scheduling efficiency, and incorporating patient preferences over appointment times in the scheduling process, by dynamically controlling a trade-off between scheduling most efficiently and fulfilling patient preferences.

3.4.1 Description

The model used in this paper covers different urgency levels, trying to schedule every patient on time (i.e.: before the due date provided by the urgency description) and a minimum access time (`mat`) is set for non urgent patients, such as they can not have an appointment before that time has passed from the patient arrival.

In particular patients flow is modeled as a Poisson process with intensity λ. Each patient p belongs to a patient group $g_p \in G$ according to a patient-group distribution D_G. The urgency of a patient is given by its group $u_p = U(g_p)$, with u_p the number of days between the arrival day and due-date. Minimum access time for non-urgent patients is given by `mat` in days. Resource capacity is C, the number of time slots on each working day. The performance measure is based on the service levels of patient groups. Service level SL_g is the fraction of patients in group g scheduled on time (before or on their due date). To aggregate scheduling performance over groups the minimum service level (MSL) is used, which is defined as MSL $= min(SL_0, \ldots, SL_{|G|})$, which aims at a high performance (close to 1) for each group.

The approach to each of the sub problems listed before is parametrized: this will allow a subsequent optimization by searching the parameters' space. The method is based upon the classic First Come First Served scheduling method by hybridizing it with the Balanced Utilization (**BU**) scheduling method: while a FIFO scheduling tends to become saturated during intense activity periods scheduling patients based on a Balanced Utilization policy results in any available time slots being spread out evenly over days, which increases the chances of them being beneficial for overflow from other groups. To combine the two orderings, available time slots ts are ordered via a weighted sum of two normalized values ($w_{g,d} = 0$ equals an FCFS ordering, while $w_{g,d} = 1$ equals a BU ordering):

$$FCFSBU(ts) = (1 - w_{g,d})FCFS(ts) + (w_{g,d})BU(ts)$$

$$FCFS(ts) = \frac{\text{rank of } ts \text{ in FCFS ordering}}{\text{total number of time slots}}$$

$$BU(ts) = \frac{(\text{utilization of day of } ts) - (\text{lowest utilization})}{(\text{highest utilization}) - (\text{lowest utilization})}$$

where we consider time slots and utilization of days before the due date. If no time slots are available to scheduling the patient before the due date a pure FCFS scheduling policy is applied, scheduling the patient to the earliest available time slot.

The key issue, now, is finding the optimal value of $w_{g,d}$ for each patient group and weekday, which involves two other parameters: the number $a_{g,d}$ of time slots allocated to patient group g for weekday d and $t_{g,d}$, which quantify the tolerance for capacity allocation overflow of patients of the patient group g for weekday d.

In this model patient preferences are modelled with a boolean preference model: a patient is scheduled either to a preferred time slot or to a non-preferred time slot. This is motivated by exclusion: the alternative of quantifying preferences is hard because it is difficult for patients to put values on preferences. Moreover, it is also hard to compare preferences values between patients, who will surely have subjective metrics to evaluate their desires. The overall objective, after this model improvement, is now a weighted combination of scheduling performance (**MSL**) explained before and patient preferences fulfillment (**PP**), the fraction of non-urgent patients that are scheduled to a preferred time slot:

$$O = (\beta) * MSL + (1 - \beta) * PP$$

By fixing a value for β a hospital department can set a preferred combination of objectives.

As already mentioned the optimization step consists in searching for optimal value into the parameter space; this is done by an Estimation of Distribution Algorithm [3, 9] (**EDA**), which is able to find good combinations of values for the 50 parameters of the model in a reasonable time (< 24 hours) for a specific scenario[5].

3.4.2 Pros and cons

The approach is very ingenious and allows the user to outline a long-term scheduling while managing emergency (by reserving an appropriate number of time slots in advance for urgencies) and patient preferences (by accepting tradeoffs between efficiency and patient satisfaction).

The only drawback is its "preventive" behaviour, because the parameter optimizations are based upon predictive models. If a situation worse than the expectations happens a simple prioritized FIFO policy is used, which is clearly suboptimal.

Similar situations are, however, unlike to happen often, thus the work have a great practical value.

3.5 COMPLEMENTARY WORKS

In this section we will examine a couple of works by Vermeulen et al. which do not resolve the hospital scheduling optimization problem, focusing instead on other aspects of the medical procedures left uncovered by our work.

5 Note that in practice the parameter values should be updated only as often as a few times per year.

These articles are cited to complete our overview of the scientific approach to shceduling in health care systems; in particular with these two works we have identified some sort of "optimization suite", that can handle efficently most of the scheduling problem instances spread in the whole health care system.

3.5.1 Decentralized online scheduling of combination-appointments in hospitals

This multi-agent system is designed to handle the outpatients in a general medical facility without incoming emergencies (i.e.: not an hospital but, maybe, a private facility or a local outpatients' department) [24].

The model, in fact, excludes emergencies arrival and precedence constraints between activities with the initial assumptions.

It also uses the same consult-diagnostic(s)-consult cycle represented in figure 1, while considering that a patient can have only a partial plan active at a time.

In particular this algorithm tries to increase the patient satisfaction by scheduling "combination appointments", which are diagnostic sessions in which two or more activities of a patient's partial plan are performed within the same day. A similar appointment allows the patient to come to the outpatients' department a reduced number of times, thus minimizing his/her effort to follow the medical procedures.

To measure the actual performances of the algorithm two metrics are considered

AVERAGE MINIMUM SERVICE LEVEL. Given that the minimum service level MSL_d of a department d is the same metric defined in section 3.4 the *average minimum service level* (**aMSL**) of all departments is nothing more than the arithmetic mean of all the MSLs :

$$aMSL = \frac{\sum_{d \in D}}{|D|}$$

COMBINATION APPOINTMENTS RATIO (**CA**). This metric is defined as the fraction of all partial plans with two or more activities all scheduled within their schedule window, that are successfully scheduled as a combination appointment.

With these metrics the pursued overall multi-objective O is definable as

$$O = max \begin{bmatrix} CA \\ aMSL \end{bmatrix}$$

The algorithm tries to achieve a multi-objective optimization by maximizing the just defined O value, by exploring the Pareto front of (CA,$aMSL$) solutions. To explore the set of Pareto fronts a cost function is defined and time slots are ordered in ascending order according to this cost (or, if two time slots have equal cost, the earliest time slot is considered to be cheaper).

This approach is really good to manage scheduling of outpatients' departments, because the model is really detailed and fits perfectly the needs of this kind of facility.

The lacks of emergency management, unfortunately, makes this approach useless for our work.

3.5.2 Adaptive Optimization of Hospital Resource Calendars

This work focus upon the resource scheduling calendars, instead of patient scheduling [26].

This is a very complex problem that can not be approached with classical methods (queuing theory will not provide any analytical answer and modeling it as a Markov decision problem returns a state space of unsolvable size), thus the approach here described was designed from scratch.

Despite the fact that this study is focused on CT-scan calendar management the explained method can be easily generalized, due to the intrinsic complexity of CT-scans reservations. A CT-scan, indeed, is a medical test literally tailor-made to the patient: the various technical values of the scan has to be calculated based on patient's biological parameters; there are some preliminary treatments that must be given in special occasions, like injection of intravenous contrast fluids or the administering of a tranquilizer medicine, for children or claustrophobic patients, and so on. However these patients can still be classified with a general taxonomy system: we have *urgent* patients, *clinical* patients and, finally, *outpatients*.

A resource calendar, in actual health care system, is prepared by making a long-term schedule (months), based on patients expectation and hospital policies; then the scheduling is tweaked with medium-term (weeks) adjustments, due to known future events, like holidays, planned maintenance of machines, etc... and, finally, last small adjustments are performed to make the short-term scheduling to fit any given optimization criteria.

The approach described in this work is designed to handle the short-term adjustments and, additionally, it can optimize the opening hours of resources as medium-term adjustment.

Knowing that non-urgent reservations are made with at least two days' notice and that an urgency has to be scheduled within one to three days the algorithm works on a 4-day scheduling window, trying to reserve time slots in advance basing on current reservations and expectations of urgent patients load.

The plan is to divide the available time of the resource in time-slots; each of these time-slots will be associated with a certain priority-class, relative to the patient taxonomy: outpatients are scheduled in their time slots (which can be further divided basing on an eventual needing of intravenous contrast, for a deeper calendar detail level) To reserve the optimal quantity of time slots for urgencies (i.e.: enough, but not too much), however, this predictive approach is not enough; the algorithm, thus, virtually divides urgent capacity while scheduling, spreading it on the following days. With this precaution urgencies can be handled by switching time slots of different capacity class, if necessary. This means, shortly, that urgencies can bump non-urgent patients reservations in the next days if no time slots are available. This is applied recursively for clinic patients, which can bump outpatients time slots. The algorithm pay attention that no reservations can fail their deadlines or, if no deadline are provided, can starve (i.e.: not being served indefinitely because of a very low priority).

The medium-term adjustment (open hours optimization) is performed basing on a fixed OH_w parameter, which sets the total amount of opening hours on week w. The actual optimization is performed before the beginning of week $w - 1$ with a standard bi-directed search method, searching for the smallest OH_w that has a minimum performance level. This approach takes into account that on each day the closing time can not be reduced further than the latest appointment already scheduled in the partially filled-in calendar.

This approach is a very generalizable approach, with initially strict assumptions which can be relaxed to fit other resource models. It is also capable of mid-term optimizations, like proposing new opening hours to the resource manager, as well as managing short-term adjustments.

Unfortunately, this approach requires that a scheduling has to be already made, so this can not be useful for our problem at all.

4 RESOLUTION METHODS

In this chapter we will examine the mathematical foundations of the solution proposed in this thesis.

As already said we focus on a CSP-based approach to find the desired solution, so we'll see how we can model a dynamic environment using constraint networks.

The language of constraint networks was originally designed to express static problems, thus the real challenge will be maintaining the consistency of our model while the environment changes, in order to ensure a coherent response to any query which can be posed anytime about the environment.

4.1 CSP

A Constraint Satisfaction Problem (**CSP**) [15] is defined up by a set of variables $V = \{V_1, \ldots, V_n\}$ and a set of constraints $C = \{C_1, \ldots, C_m\}$, with every variable V_i having a domain $D_i \neq \emptyset$. Every constraint C_i involves a subset of the variables, specifying allowed combination of values for these variables; therefore the following statement holds:

$$\text{Let be } V \supseteq V' = \{V_1, \ldots, V_l\}.\ C_h \subseteq D_1 \times D_2 \times \cdots \times D_l$$

We say that a state X of the CSP is defined from the *assignment* of a value to some (or all) of its variables, such that $\{V_1 = x_1, \ldots, V_h = x_h : x_i \in D_i\ \forall\ i$.

An assignment X is *consistent* if and only if satisfies every constraint of the CSP. Thus $X \in C_i\ \forall\ i \iff X$ is consistent.

An assignment X is *complete* if and only if it involves all the variables of the CSP.

A solution for a CSP, thus, is given by a *consistent, complete* assignment of values to its variables.

A CSP, thus, is a high-level concept framework used to model a problem and lends itself to a wide range of implementation approaches; the most natural one is a constraint network, which is the mere translation of the just explained CSP definition into a graph[6]. The variables become the nodes of the graph, with their respective domains still associated to them, and, on the other hand, the constraints become the edges of the graph, with an end in each node representing one of the variables involved by the constraint, encapsulating the list of allowed assignment for the involved variables subset.

4.2 DYNAMIC CSP

The notion of *dynamic CSP* (**DCSP**) has been introduced to represent dynamic situations. A DCSP is a sequence of CSPs, where each one differs from the previous one by the addition or removal of some constraints. It is indeed easy to see that all the possible changes to a CSP (constraint or domain modifications, variable additions or removals) can be expressed in terms of constraint additions or removals.

To solve such a sequence of CSPs, it is always possible to solve each one from scratch, as it has been done for the first one. But this naive method, which remembers nothing from the previous reasoning, has two important drawbacks:

1. **Inefficiency**, which may be unacceptable in the framework of real time applications (planning, scheduling, etc.), where the time allowed for replanning is limited;

2. **Instability** of the successive solutions, which may be unpleasant in the framework of an interactive design or a planning activity, if some work has been started on the basis of the previous solution.

The original method of belief mantainment proposed in [7], together with the definition of DCSP, was designed to act on the constraint network itself, keeping in mind the undesirable limits of the trivial resolution process.

The algorithm is invoked when the constraint network is altered, due to a change in constraints or variables reflecting an actual change happened in the dynamic environment, and it consists of two different phases.

The first one is a phase called *support propagation*, in which the change propagates through the network, notifying what happened to the variables, and if after this the network is in a contradiction state the second phase, called *contradiction resolution* begins.

In this subsequent phase the variable who detected the contradiction tries to enumerate the other variables that have to be reassigned in order to restore consistency and starts with these variables a recursive solution process of the contradiction.

The other existing methods can be classified in three groups:

1. **Heuristic methods**, which consist of using any previous consistent assignment (complete or not) as a heuristic in the framework of the current CSP [20].

2. **Local repair methods**, which consist of starting from any previous consistent assignment (complete or not) and of repairing it, using a sequence of local modifications (modifications of only one variable assignment). The original method proposed by Dechter in [7] is among these ones.

3. **Constraint recording methods**, which consist of recording any kind of constraint which can be deduced in the framework of a CSP and its justification, in order to reuse it in

the framework of any new CSP which includes the same justification[16].

The methods of the first two groups aim at improving both efficiency and stability, whereas those of the last group only aim at improving efficiency.

A little apart from the previous ones, a fourth group gathers methods which aim at minimizing the distance between successive solutions[1].

4.2.1 Local Changes Algorithm

A special consideration has been reserved to the *local changes* algorithm, because it was designed to tackle an objective similar to the one we have[1], i.e.: manage a scheduling in a dynamic environment while altering as little as possible the already planning.

The *local changes* algorithm [21] was formalized among some studies for the French Space Agency (CNES) which aimed at designing a scheduling system for a remote sensing satellite (SPOT).

In this problem, the set of tasks to be performed evolved each day because of the arrival of new tasks and the achievement of previous ones. One of the requirements was to disturb as little as possible the previous scheduling when entering a new task.

For solving such a problem, the following idea was used: it is possible to enter a new task t if and only if there exists for t a location such that all the tasks whose location is incompatible with t's location can be removed and entered again one after the other, without modifying t's location.

In terms of CSP, the same idea can be expressed as follows: let us consider a binary CSP; let A be a consistent assignment of a subset V of the variables; let v be a variable which does not belong to V; we can assign v, i.e., obtain a consistent assignment of $V \cup \{v\}$, if and only if there exists a value val of v such that we can assign val to v, remove all the assignments (v', val') which are inconsistent with (v, val) and assign these unassigned variables again one after the other, without modifying v's assignment. If the assignment $A \cup \{(v, val)\}$ is consistent, there is no variable to unassign and the solution is immediate.

Note that it is only for the sake of simplicity that we are considering a binary CSP: the proposed method can deal easily with general n-ary CSPs.

With such a method, for which we use the name *local changes* and which clearly belongs to the second group (local repair methods), solving a CSP looks like solving a fifteen puzzle problem: a sequence of variable assignment changes which allows any consistent assignment to be extended to a larger consistent one.

The corresponding algorithm can be described as follows:

1 see paragraph 5.2.1 for a more detailed analysis of the optimization criteria

```
local-changes(csp)
// Main call to resolve whole CSP from scratch
    return lc-variables($\emptyset, \emptyset$, variables(csp))

lc-variables($V_1, V_2, V_3$)
// Choose a variable from the unassigned set and
// call lc-variable to assign it
// $V_1$ is the set of assigned and fixed variables
// $V_2$ is the set of assigned and not fixed variables
// $V_3$ is the set of unassigned variables
    if $V_3 = \emptyset$
    then return success
    else let $v$ be a variable chosen in $V_3$
        let $d$ be its domain
        if lc-variable($V_1, V_2, v, d$) = failure
        then return failure
        else return lc-variables($V_1, V_2 \cup \{v\}, V_3 - \{v\}$)

lc-variable($V_1, V_2, v, d$)
// Try to assign the variable passed in the parameters.
// Use lc-values to verify the consistency of possible assignments
    if d=$\emptyset$
    then return failure
    else let $val$ be a value chosen in $d$
        save-assignments($V_2$)
        assign-variable($v, val$)
        if lc-value($V_1, V_2, v, val$) = success
        then return success
        else unassign-variable($v$)
            restore-assignments($V_2$)
            return lc-variable($V_1, V_2, v, d - \{val\}$)

lc-value($V_1, V_2, v, val$)
// Verify the consistency of current assignment and, if necessary,
// try to deassign other variables in order to achieve it
// and call lc-variables on the new unassigned variables.
    let be $A_1$ = assignment($V_1$)
    let be $A_{12}$ = assignment($V_1 \cup V_2$)
    if $A_1 \cup \{(v, val)\}$ is inconsistent
    then return failure
    else if $A_{12} \cup \{(v, val)\}$ is consistent
    then return success
    else let $V_3$ a non empty subset of $V_2$ such that
        let $A_{123} = assignment(V_1 \cup V_2 - V_3)$
        $A_{123} \cup \{(v, val)\}$ is consistent
        unassign-variables(V3)
        return lc-variables($V_1 \cup \{v\}, V_2 - V_3, V_3$)
```

Local changes algorithm pseudocode

Correctness, termination and completeness of the algorithm and the three procedures are granted by the following theorems, proven in [21]:

Theorem 4.1. *If the CSP csp is consistent (resp.inconsistent), the procedure call lc(csp) returns success (resp. failure); in case of success, the result is a consistent assignment of csp's variables.*

Theorem 4.2. *Let V_1 and V_2 be two disjunct sets of assigned variables and let V_3 be a set of unassigned variables; let be $V = V_1 \cup V_2 \cup V_3$; let be $A_1 = assignment(V_1)$. If there exists (resp. does not exist) a consistent assignment A of V, such that $A \downarrow_{V_1} = A_1$[2], the procedure call lc-variables(V_1, V_2, V_3) returns success (resp. failure); in case of success, the result is a consistent assignment of V.*

Theorem 4.3. *Let V_1 and V_2 be two disjunct sets of assigned variables; let v be an unassigned variable; let d be its domain; let be $V = V_1 \cup V_2 \cup \{v\}$; let be $A_1 = assignment(V_1)$; if there exists (resp. does not exist) a consistent assignment A of V, such that $A \downarrow_{V_1} -A_1$, the procedure call lc-variable(V_1, V_2, v, d) returns success (resp. failure); in case of success, the result is a consistent assignment of V*

Theorem 4.4. *Let V_1 and V_2 be two disjunct sets of variables; let v be an unassigned variable; let val be one of its possible values; let be $V = V_1 \cup V_2 \cup \{v\}$; let be $A_1 = assignment(V_1)$; if there exists (resp. does not exist) a consistent assignment A of V, such that $A \downarrow_{V_1 \cup \{v\}} = A_1 \cup \{(v, val)\}$, the procedure call lc-value(V_1, V_2, v, val) returns success (resp. failure); in case of success, the result is a consistent assignment of V.*

2 Let A be an assignment of a subset V of the CSP variables and be $V' \subseteq V$; the notation $A \downarrow_{V'}$ designates the restriction of A to V'.

5 SOLUTION

In this chapter we will see how the Dynamic CSP solution approach, already seen in chapter 4, has been implemented.

We will start from examining the mathematical model of the problem and we will move on to the implementation of the *local changes* algorithm, also seen in chapter 4.

5.1 MODEL

It is really hard to make a rigorous mathematical formulation of the Hospital Optimization problem. Luckily, we can find in [12] a strict mathematical model which resemble the hospital reality in a truly accurate way.

Definition 5.1. HORIZON. *A* horizon $T = \{0, \dots, \hat{t}\}$ *is a finite set of integers that represents the set of possible starting times for appointments. Given a constant number nd of starting times per day,* $day(t) = t \div nd + 1$, $Day(T) = \{day(t) | t \in T\}$.

In our model the minimum allowed difference between two consecutive values of a horizon, referred as time unit, time quantum *or* temporal quantum *is the quarter of an hour. This is enough to organize the appointments with a significant precision*

Definition 5.2. WORKPLACE.
A workplace *is a pair* $w = (AT, T_{avail})$. $AT = \{(at, \Delta t_{dur}, \Delta t_{change})\}$ *is the set of appointment types provided by the workplace with identifier* $at \in \mathbb{N}$, *the duration* $\Delta t_{dur} \in T \backslash \{0\}$ *and the required buffer time* $\Delta t_{change} \in T \backslash \{0\}$ *between two appointments of this type.* $T_{avail} \subseteq T$ *is the set of starting times on which the workplace is available, also called "workplace availability set".*

Definition 5.3. DIAGNOSTIC UNIT. *A* diagnostic unit *is a triple* $u = (W, AT, \hat{r}_{staff})$. *W is the set of workplaces in this diagnostic unit,* $AT = \{(at, \hat{r}_{day}) | (at, \cdot, \cdot) \in w.AT \wedge w \in u.W\}$ [1] *is the set of appointment types provided by the diagnostic unit with identifier* $at \in \mathbb{N}$ *and the maximum number of these appointments per day* $\hat{r}_{day} \in \mathbb{N}^+$. $\hat{r}_{staff} \in \mathbb{N}^+$ *is the maximum number of staff resources available for parallel appointments.*

Definition 5.4. APPOINTMENT. *An* appointment *is a pair* $ap = (at, as_\lambda)$. $at \in \mathbb{N}$ *is the appointment type identifier and* as_λ *is the actual reservation.*

Definition 5.5. RESERVATION. *A* reservation *is a triple* $as = (t_{start}, \Delta t_{dur}, w)$ *where* $t_{start} \in T$ *is the starting time,* $\Delta t_{dur} \in T \backslash \{0\}$

1 *x.y* denotes the projection of the structure *x* onto the component with name *y*.

is the duration and w is the workplace where the treatment required is administered.

The act of accepting a patient request is formalized by associating a reservation to the appointment which represent the request itself. An appointment that has been accepted by associating it with a reservation is called an assigned appointment*. This definition is helpful while managing reschedulings, which are reservations change processes that* unassign *(i.e. remove the reservation from) and subsequently* reassign *(i.e. associate a different reservation to) an appointment.*

Definition 5.6. PATIENT. *A* patient *is a triple p = (AP,bef,T_{avail}). AP is the set of appointments of the patient, bef : $2^{AP \times AP} \to \{0,1\}$ is the partial order relation among appointments, $T_{avail} \subseteq T$ is the set of starting times on which the patient is available.*

Definition 5.7. CLINIC. *A* clinic *is a pair cl = (P,U). P is the set of patients, U is the set of diagnostic units.*

We see that this model encompasses all the major aspect of an hospital reality. To exploit this model to our algorithm we must adapt this mathematical formulation to the CSP framework.

This model is weak on two fundamental aspects to be fully usable by a CSP-based algorithm: as already said a CSP must present a set of variables and a set of constraints.

5.1.1 Variables

By examining the logic of the model itself we can understand that, for the purpose of writing an adequate solver, many of the definitions are destined for mere registry purpose. Considering that this is, first of all, a scheduling problem we can focus our attention on definitions 5.4 and 5.5.

At a first glance the appointments are a natural choice for the variable role, while the reservations will become the values that those variables can assume. Logically speaking this is true: an appointment has to be scheduled in a given workplace, and the patient, once the treatment is started, stays in there (i.e.: he *holds* that resource, which is mutually exclusive, for himself) until he is done.

This reasoning, unfortunately, will not fit in the model we have just formulated: an appointment, here, contains only the information about the appointment aspects; the data of its assignment are actually stored in the reservation entity.

This choice made on model definition leads to a single conclusion: the reservations will be the variables of our CSP, while their domains are all the possibles triples which can fit the definition 5.5.

5.1.2 Constraints

A major lack in the model regards the constraints, instead.

No constraints have been specified up to now, and a real CSP have many of them. Without the appropriate constraints our scheduler could choose to put more than one patient in the same CT-scan machine, or ask a patient to receive two different treatments in two different rooms of the hospital simultaneously.

Constraints, then, are not only useful to include in the model personal preferences, aiming to find better solutions among the acceptable ones, but are also necessary to symbolize even the most basic law of the ambient in which the problem is located (the hospital, in our case).

We can divide the constraint that act in the Hospital Scheduling Optimization problem in two different categories: constraints which act on patients and constraints which acts workplaces.

A summary listing of these constraints, specifying a brief description and the entity which are involved, follows:

- Constraints which act on patients.
  - **Availability** : a patient must be physically present for a treatment;
  - **Partial Order** : if some appointments of a patient have precedence constraints on the other this must be specified;
  - **Non-overlap** : a patient can not receive two or more treatments at the same time;
- Constraints which act on workplaces.
  - **Appointment type sufficiency** : a workplace can administer only the treatments which are covered by its abilities;
  - **Availability** : a workplace must be operative to administer a treatment;
  - **Duration** : the duration of a treatment depends on the equipment used to administer it, thus is set by the workplace chosen for the administration.
  - **Change Time** : medical devices might need an inactivity period (in order to cool off, heat up, etc. . .) between two appointments of the same type.
  - **Non-overlap** : a workplace can not administer two treatments at the same time.

As we can see all the constraints represents basic laws of the hospital environment that must be specified in the DCSP, in order to exclude solution which can be unfeasible into reality. In the following sections we will examine more accurately and formalize these constraints.

5.1.2.1 *Patients' constraints*

In this section are listed the constraints which act on the patients. We will see, as just mentioned, that some constraints are meant to model the basic law of reality (i.e.: a patient can not be in two different places at the same time).

Availability.

Like release and due dates on Job Shop Scheduling problem a patient enters the clinic at a given moment and should leave the clinic in a subsequent instant.

$\forall ap \in p.AP : \dots$

$\dots : \left(ap.as_{\lambda}.t_{start} \in p.T_{avail} \wedge ap.as_{\lambda}.t_{start} + ap.as_{\lambda}.\Delta t_{dur} \in p.T_{avail}\right)$

Such constraint is, usually, a soft constraint in a real hospital: a patient availability, in this case, is open-ended and covers the rest of the scheduling horizon. Nevertheless, in our model this is considered an hard constraint: this choice came after a lot of considerations about it.

Reasons why patients availability is changed to a hard constraint

1. An emergency must be served as soon as possible, and the size of its availability horizon is inversely proportional to its criticity level: thus an emergency case has, usually, a very restricted set of available start time for treatments. In this case the availability can *not* be considered a soft constraint.

2. Choosing a hard policy for this constraint will preserve a patient from the risk of being never served: when the availability set is progressively reduced (which happens naturally because the advancing of time literally erodes, starting from the first element, the patient's availability set) the presence of an hard constraint automatically raises the priority of all the appointments of this patient, helping them to exit the starvation status.

3. In some medical systems, like the Italian one, some medical prescriptions have a given priority level, which means that the appropriate treatment must be given before a predetermined due date. In this case, thus, this is naturally a hard constraint.

4. Many worker patients, for not-urgent treatments, have actually an open-ended temporal availability. This availability set, however, is really strict if examined day-by-day: a patient can have only the morning available for treatments on Monday, maybe nothing on Tuesday, only the afternoon on Wednesday and so on. And if this kind of constraint is broken the patient might be criticized on his job place, despite the medical nature of his lateness.

Partial order.

A possible partial order in appointment exists for medical reason and has to be obeyed anyway. We can formalize this constraint as follows:

$$\forall ap_1, ap_2 \in p.AP : (ap_1, ap_2) \in p.bef \Rightarrow \dots$$

$$\cdots \Rightarrow ap_1.as_\lambda.t_{start} + ap_1.as_\lambda.\Delta t_{dur} < ap_2.as_\lambda.t_{start}$$

Non-overlap.

The patients are, physically speaking, symbolized as a scarce resource that has to be available for each appointment. In the model we use a patient can participate in only one appointment at a time, so the appointments of the same patient must not overlap with each other.

Given a patient p with his appointments $p.AP = \{ap_1, \dots, ap_s\}$ we can formalize this constraint using the more general *cumulative* constraint[2]:

$$cumulative\big((ap_1.as_\lambda.t_{start}, \dots, ap_s.as_\lambda.t_{start}), \dots$$

$$\dots, (ap_1.as_\lambda.\Delta t_{dur}, \dots, ap_s.as_\lambda.\Delta t_{dur}), \underbrace{(1, \dots, 1)}_{s}, 1\big)$$

5.1.2.2 *Workplaces' constraints*

In this section we will see the constraints acting on every workplace.

Appointment Type Sufficiency.

Every appointment assigned to a workplace w must be an appointment of a type that workplace can actually provide. This can be formalized this way:

$$\forall ap \in AP_w : (ap.at, \cdot, \cdot) \in w.AT$$

Availability.

Like what expressed in the homonymic patients constraint, a workplace w has to be available for all the appointments assigned to it

$$\forall ap \in AP_w : ap.as_\lambda.t_{start} \in w.T_{avail}$$

Duration.

Assigning an appointment of a certain type to a workplace w directly determines the duration of the appointment, according to the specification given by $w.AT$. We formalize it with the following assumption:[3]

$$\forall ap \in AP_w : ap.as_\lambda.\Delta t_{dur} = (head\{(ap.at, \cdot, \cdot) \in w.AT\}).\Delta t_{dur}$$

Change time.

Change times might be necessary between appointments of the same types. This is motivated by technical requirements of

2 Given a tuple $(t_1, \dots, t_n)$ of integer starting times $t_i \in T$,a tuple $(\Delta t_1, \dots, \Delta t_n)$ of integer durations $\Delta t_i \in T \backslash \{0\}$, a tuple $(r_1, \dots, r_n)$ of integer resources $r_i \in \mathbb{N}$ and an integer resource threshold $\hat{r} \in \mathbb{N}^+$, *cumulative*$((t_1, \dots, t_n), (\Delta t_1, \dots, \Delta t_n), (r_1, \dots, r_n), \hat{r})$ is specified by

$$\forall k \in \big[\min_{i \in [1,n]} \{t_i\}, \max_{i \in [1,n]} \{t_i + \Delta t_i - 1\}\big] : \sum_{t_j \le k \le t_j + \Delta t_j - 1} r_j \le \hat{r}.$$

3 the *head* operator returns the first element of a set

diagnostic devices used in that kind of appointments: they might need to cool down or heat up before another treatment can be given.

$$\forall\ ap_1, ap_2 \in AP_w, ap_1 \neq ap_2, \Delta t_{change} = (head\{(ap.at, \cdot, \cdot) \in \in w.AT\}).\Delta t_{change}\ :\ ap_1.at = ap_2.at \Rightarrow \ldots$$

$$\ldots\ \Rightarrow ap_1.as_\lambda.t_{start} + ap_1.as_\lambda.\Delta t_{dur} + \Delta t_{change} \leq ap_2.as_\lambda.t_{start} \vee \vee\ ap_2.as_\lambda.t_{start} + ap_2.as_\lambda.\Delta t_{dur} + \Delta t_{change} \leq ap_1.as_\lambda.t_{start}$$

Non-overlap.
Like the patients a workplace can only participate in one appointments at a given time. We model this assumption by using the *cumulative* constraint again:

$$cumulative((ap_1.as_\lambda.t_{start}, \ldots, ap_t.as_\lambda.t_{start}), \ldots$$

$$\ldots, (ap_1.as_\lambda.\Delta t_{dur}, \ldots, ap_t.as_\lambda.\Delta t_{dur}), \underbrace{(1, \ldots, 1)}_{t}, 1)$$

5.2 ALGORITHM

The dynamic CSP solution approach is perfect for modeling our environment: an incoming patient adds a single variable and some constraints to the problem, without altering the already present ones, while the leaving of a patient, on the contrary, removes a variable and the constraints associated to it, de facto relaxing the set of constraints of the problem.

These incremental changes, distributed in time, are exactly like what has been described in section 4.2, so we already have many tested methods to preserve the model coherency.

Such evolution of the problem can be effectively tracked in particular with the *local canghes* algorithm, seen in paragraph 4.2.1. This algorithm fits perfectly our work environment, granting the progressive solution stability and the computation efficiency which are mandatory in a planning problem like the one we are solving.

5.2.1 Optimization criteria

An hospital has a clear objective: maximize the number of treated patients in a certain period of time.

Therefore, given a set of patients we can say that an hospital aims at to treat the maximum number of these patients, with this number depending on the circumstances. What we seek, thus, is a schedule that allows every patient to be treated respecting his time restrictions. Unfortunately this is not always possible: some patients might have requests that are incompatible with the constraints specified above; one or more of these patients, thus, must be *rejected*, in order to treat all the others without any constraint violation. In such cases, thus, we say that the schedule we want is the one which rejects the lesser number of patients among the ones in the set; we call this schedule as a *feasible schedule*.

With such an advanced framework, actually, we can aspire more than find just a feasible schedule with the given tasks; the goal we pursue, then, is to find a *good* feasible schedule. But when, exactly, a schedule is a *good* schedule?

As already specified in paragraph 2.2.1 we have many metrics to evaluate the goodness-score of a schedule but here we focus, in particular, on a precise aspect of the incremental schedule generated from the algorithm: the number of rescheduling operations[4] performed.

In an environment like this a reschedule operation is not costless: most of the rescheduled appointments imply a discomfort for a patient and extra-work for the hospital staff, which have to rearrange its personal planning to follow the changes of the global schedule. In the worst case a rescheduling of an appointment which is not urgent implies a phone call to the patient, informing him that his appointment has been rescheduled, propagating the planning change outside the hospital environment (which usually implies even more discomfort for the rescheduled patient).

With this approach we will try to schedule all the incoming patient, with all the objective already specified in paragraph 2.2.1 while attempting to minimize the number of this kind of operations used during the process. This means that the scheduler has to accept the largest number of patients up to the hospital capability while trying to alter the current schedule as little as possible.

5.2.2 Implementation guidelines

Referring to the algorithm described in paragraph 4.2.1 the structure of our implementation is quite similar: each unassigned appointment has its record created but no reservation record are associated to it. This, in a CSP context, is represented by an unassigned variable, actually creating the empty reservation record; this variable will be subsequently assigned by the *local changes* algorithm, preserving the solution consistency all the time.

These guidelines grant that the resolutive algorithm will be complete, correct and will always terminate if invoked, for what already proven with theorems 4.1, 4.2, 4.3 and 4.4.

5.2.3 Heuristics

To improve the performance of the algorithm in terms of usefulness of the solution found some heuristics have been used during the implementation. The key idea behind the use of these heuris-

4 A rescheduling operation is the act of unassigning a variable and then reassigning another value to it, different from the previous one.

tic is to improve the performance achieved by two instructions, given by the *local changes* algorithm, which are not specified in a strict way, leaving a certain degree of freedom to the implementer.

Let see in detail what we can use and where:

5.2.3.1 *Fail-First Heuristic*

Into the *lc–variables* subroutine there is a single line saying

let v be a variable chosen in V_3

where V_3 is the set of currently unassigned variables. In terms of hospital scheduling this line represents the act of choosing an appointment to schedule among the unassigned ones.

A good approach, in AI problem solving, is to randomly choose the variable v among all the variables of the set V_3; such method, as a matter of fact, give the algorithm a good capability of avoiding the local minima present in the search space without loss of performance, assuming that choosing a variable or another will not hurt the performances.

In our reality, however, we can expect a bit of heterogeneity in variables weight: for example, a patient can be a young student or a worker. The first one will probably not have any urgent tasks to attend in the next few days of his scholastic career, so it will have a wide range of available starting times to receive his treatment; on the contrary, if the given patient is a worker with an important deadline his needs are more urgent that the one of the student.

This particular urgency is reflected by the actual size of the availability set T_{avail}, described in definition 5.6, so a scheduling policy that would favor patients with restricted availability set, scheduling them before the ones with a wider T_{avail} set, will probably grant a higher level of satisfaction by the patients and will almost certainly improve the performance of the scheduling process[15]: a variable with a strict domain is, after all, more likely to become unassignable if we try to assign a value to it at the end of the process, compared to a variable with a wider domain, so if this situation is avoided it is unlikely that we should resort to a backtracking.

5.2.3.2 *Min-Conflicts*

This heuristic tries to help minimizing the number of rescheduling needed to insert a new appointment in the time table. [15]

In particular we use this heuristic while executing this line of *lc-variable* subroutine:

let val be a value chosen in d

where d is the domain of a variable we are trying to assign; note that we are improving a choosing-from-a-set instruction here too.

The idea behind this heuristic is based on the fact that if an appointment needs to be reassigned it means that it has been previously unassigned for some reason, and this is just the type of operations we try to minimize.

In particular we must unassign an appointment only for a single, although very generic, reason: a constraint has been violated. Some of the constraints seen in the previous section can be violated even by one appointment only, like the **availability** constraint of patients and workplaces (there is no need for a second appointment to violate this constraint) but this kind of violation is usually blocked in advance by preventing a similar assignment to be made at all. An appointment, thus, is not unassigned for *any* constraint violation but only for a constraint which can be violated *after* this appointment's assignment, and this can happen only when another appointment is assigned to an already reserved time slot.

A freshly assigned appointment can break a constraint in three ways:

1. It tries to use a scarce resource already taken by another appointment, infringing the **non-overlap** constraint of that resource (patient or workplace)

2. It tries to access a free workplace without waiting that the appropriate devices are ready for the treatment, being just used for a previous appointment, violating the **change time** constraint of that workplace

3. Its reservation breaks the **partial order** relation between his patient's appointments

In such cases we have no choice but to unassign one appointment or the other and to try to reassign it. We can not exclude one of the options in advance, because both of them can lead to the optimal solution, but we surely recognize that it is better to avoid a similar operation.

We say that two appointments a_1 and a_2 *conflict* if and only if a_1 breaks one or more constraints involving a_2. This definition is useful while evaluating in advance the implications of a single reservation.

From this reasoning we have a guideline to follow while evaluating which value to assign to a variable, among the ones present in its domain: the less conflicts this variable generates if a certain value is assigned to it, the less variables will be consequently unassigned and, therefore, the less reassignment will be globally done during the entire process.

It is also worth mentioning that a single reservation can conflict with more than one already assigned appointment at the same time, implying that all of them will be unassigned in the next step of the algorithm.

This heuristic, therefore, change the value-picking process slightly: let be v the variable we are trying to assign and let d be its domain

1. for each value $val \in d$ assume assigning val to v

2. for each already assigned variable v' evaluate if there are any conflict between the two variables
3. if a conflict is present count that as one

In the end the *min-conflict* heuristic favors the value that will generate the minimum number of conflicts if assigned to v, thus the one who obtained the lesser score during the evaluation process.

5.2.4 Data Structure

The *local changes* algorithm is a local search algorithm[15]: it takes decisions based only on local-gathered informations while trying to find a solution, and if no solution is found it can backtracks its steps on the search tree until a new option, not previously taken, can be picked.

Knowing this we must choose a data structure capable of facilitating this process and, given that there is no need to evaluate informations that are no locally accessible, the best choice is to use a tree reproducing the actual search tree. To build the search tree as-is, however, we must consider that the decision taken are not uniform: we pick, alternately, an appointment to schedule and a slot on the time table to insert the appointment into it. This means that our search tree must be a bicoloured tree: let's say that the nodes are black and red, we'll have that a black node will encapsulate an appointment to be scheduled and, for each possible reservation with which that appointment can be scheduled, a red node encapsulating that reservation will be added as children. This particular pair of nodes represent a pair (appointment, reservation) or, from a CSP point of view, an actual variable assignment, being a pair (variable,value).

Every red node, on the other hand, will have black children that will encapsulate all the appointments still to be scheduled after the partial assignment currently evaluated, composed by the set of pairs (variable,value) we found along the path from the current node back to the root.

Nota bene: this implies that the V_3 set used by the subroutines *lc-variables, lc-variable* and *lc-value* in the *local changes* algorithm is de facto embedded in every red node of the search tree. This will come useful later on, while discussing the embedding of this structure into the algorithm.

The resulting structure looks like the one pictured in figure 2. Note that at the top of the tree there is a dummy red node used as a root: this is mandatory because, knowing that the children of a red node are the elements of the V_3 list, we may have more than one unassigned variable at the beginning, so the tree must have more than one black node just from the first level of exploration.

A similar data structure allows a quick backtracking in case of unsuccessful exploration and also allows, for each node we visit,

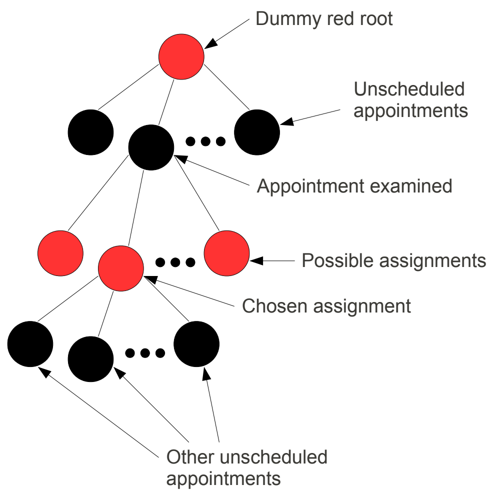


Figure 2: Bicoloured search tree

a clear look on the available option to carry on the exploration of the decision tree. Minding that the objective is to embed this structure in the *local changes* algorithm we can advantage ourselves in various ways by exploiting the topological structure of the tree. Apart from the memory-preserving strategies (i.e.: expand only the nodes we try to explore, collapse the subtrees we're not interested to explore, etc. . .) a clever implementation choice is to encapsulate into the black nodes also the V_1 and V_2 sets required by the three subroutines of the algorithm. This way we have a perfect backtrackable situation and we can improve the performance in term of computation time even further, by translating the recursive algorithm into an iterative version: the information which were saved into the stack with recursive calls now are already memorized in the tree nodes or into the tree structure itself, having the V_1 and V_2 set saved into the black nodes and the V_3 set implicitly memorized into the children list of every red node.

Considering the information encapsulated into the tree and knowing the behaviour of *local changes* algorithm we can conclude that a path from the root to a red leaf means that the algorithm managed to find a reassignment for a certain number of variables (maybe all of them) of the CSP. Such a path will be composed by a certain number of pairs $(variable, value)$ that make up a partial reassignment for the CSP.

Contrariwise, a black leaf means that an unassigned variable with an empty domain exists; if that's the case the *lc-variable* subroutine will return a *failure* while examining that leaf.

We can state the following theorem:

Theorem 5.1. *Let be a CSP with a set of variables* $V = V_1 \cup V_2 \cup V_3$*, with* V_1, V_2, V_3 *being respectively*

1. *the set of assigned but not deassignable variables of the CSP*

2. *the set of assigned and deassignable variables of the CSP*

3. *the set of unassigned variables of the CSP*

and let $V_1 \cup V_2$ be consistent with the CSP[5].

Let T be the bicoloured tree built with the method just explained in this paragraph.

If a path from the root to a red leaf exists we can assign every value encapsulated into a red node on this path (except the root) to the variable encapsulated into his black node father.

This way we obtain a consistent assignment for the original CSP.

Otherwise there's no complete and consistent assignment for this CSP.

Proof. The proof of this theorem follows directly from the correctness of local changes algorithm, already proven by theorems 4.1, 4.2, 4.3 and 4.4. □

5.2.5 Stochastic Exploration

In local search problems we travel in the solution space, moving from a solution to another near one, trying to find an optimal solution among the ones we are passing by. In order to do this we specify the "moving primitives", to permit the exploration of the solution space and an evaluation metrics used by the local search algorithm to see which is the most promising direction to take, step by step.

A clear example of a local search algorithm is the hill-climbing algorithm [15]:

1. for each solution reachable from the current one evaluate its score with a given an optimality metric.

2. travel to the solution who reaches the highest score among the evaluated ones.

This method is really efficient because every decision is taken evaluating a small portion of the problem space, which is why this algorithm class is called *local repair* algorithms, as mentioned in section 4.2, but has a main weakness: it may stops in a local minimum.

A clever move is to apply a Random Walk strategy[6] to enhance our algorithm's ability of escape such minim. The Random Walk strategy was designed primarily for SAT[6] problems, combining a random search with a greedily computed bias. In other words, given a variable v that we are trying to assign we can estimate a probability distribution for this variable such that the more a value x would be favored by our local repair algorithm during the greedy evaluation the more this value is likely to be picked.

5 Mind that with the three points of this definition the following statement holds: $V_i \cap V_j = \emptyset \ \forall \ i, j : i \neq j$

6 SAT, or *propositional satisfiability problem*, is a NP-Complete problem. It's famous because is the first known example of NP-Complete problem [6]

Applying a *stochastic local repair* algorithm to our problems, thus, imply that the decisions have to be made with a weighed random policy. In other words, when an option has to be chosen, every alternative will be associated with a probability proportional to its score.

The only decisions we made are the two we have already analyzed once noticing the "degree of freedom" left in the local changes algorithm description: how to choose an unscheduled appointment to be scheduled and how to choose a possible slot to assign to this appointment.

The two aforementioned heuristic, Fail-First and Min-Conflicts (see Section 5.2.3) can be exploited to estimate the goodness of every option: an appointment is more attractive the more restricted its domain is, and an reservation is more likely to be chosen if it creates less conflicts if accepted.

Given that O is the set of options for a given choice the remaining steps are to associate a probability p to each of the available options such that $\sum_{o \in O} p(o) = 1$ and pick the current option randomly using the probability distribution just computed.

5.2.6 Algorithm Pseudocode

This section shows the pseudocode of the algorithm we've just described.

Methods with a trivial implementation have only their prototype listed and private attributes of classes are not shown.

The behaviour of the *chooseCandidateIndex* method is explained after the algorithm source code.

```
Code block start
```

```
interface _Node
{
 //mark this node as already expanded
 public void setAsExpanded();
 //return true if setAsExpanded() has already been
 //invoked on this node
 public boolean hasBeenExpanded();
 //return true if the node has no children
 public boolean isLeaf();
 //return the x-th child of the children list
 public _Node getChild(int x);
 //return the children list of this node
 public Collection<_Node> getChildren();
 //return the father of this node
 public _Node getFather();
}
class RedNode implements _Node
{
 //return the reservation encapsulated into this red node
 public Reservation getReservation();
 //Build one black node for each given appointment with
 //the given V1 and V2 set encapsulated into them too.
 //Then add these black nodes to the children list.
 public addToChildren(Collection<Appointment> apps,
Collection<Appointment> V1, Collection<Appointment> V2);
}
class BlackNode implements _Node
{
 //return the appointment encapsulated into this
 //black node
 public Appointment getAppointment();
 //return a collection of the appointments encapsulated
 //in the brothers of this node
 public Collection<Appointments> getBrothers();
 //remove the given child from the children list
 public void removeChild(RedNode target);
 //build the children list of this node: each RedNode
 //encapsulate a possible Reservation for the
 //appointment contained in this BlackNode
 public void buildChildrenList();
 //return the set V1 encapsulated into this node
 public Collection<Appointment> getV1();
 //return the set V2 encapsulated into this node
 public Collection<Appointment> getV2();
}
/**
 This is the method which is called to resolve the problem
instance
 @param newApp : the collection of appointments to be
scheduled
 @param oldApp : the collection of appointments already
scheduled
 @return a collection of reservations, each one to be assigned to
its appointment, which compose the schedule change, or null if a
feasible schedule does not exist.
*/
public Collection<Reservation>
askForSchedulingProposal(Collection<Appointment> newApp,
Collection<Appointment> oldApp)
{
 //allocate V1 and V2 sets for blacknode buildings
 //let suppose that collections can be instantiated
 Collection<Appointment> V1 = new Collection();
 Collection<Appointment> V2 = new Collection();
 V2.addAll(newApp);
 //build the first level of the tree
 RedNode root = new RedNode();
 root.buildChildrenList(newApp, V1, V2);
 root.setAsExpanded();
 return exploreTree(root);
```

```
}
/**
 This method is called to explore the tree, searching for a solution
 @param root : the start node for the tree exploration process
 @return a collection of reservations, each one to be assigned to
its appointment, which compose the schedule change, or null if a
feasible schedule does not exist.
*/
private Collection<Reservation> exploreTree(RedNode root)
{
 _Node currNode = root;
 //set the exploration cycle
 boolean goOn = root.hasBeenExpanded && !root.isLeaf();
 while(goOn)
 {
  //choose what child is to be visited next
  int index = visitNode(currNode);
  //if index < 0 there is an interesting case
  if(index<0)
  {
   RedNode wrongNode = null;
   //if is a red node...
   if(currNode instanceof RedNode)
   {
    //if it is a leaf a consistent assignment has been found: exit
the cycle
    if(currNode.isLeaf())
     goOn = false;
    //otherwise there is an appointment with empty domain into
its children
    else
    {
     //mark this reservation to be removed
     wrongNode = currNode
    }
   }
   //otherwise we are in an empty black node: the reservation to
be deleted is its red father
   else
   {
    wrongNode = currNode.getFather();
   }
     //if there is a reservation to remove...
   if(wrongNode != null)
   {
    //backtrack to the previous blacknode
    currNode = wrongNode.getFather();
    //if there's no previous blacknode the problem is unsolvable
from the first level
    if(currNode == null)
     goOn = false;
    else
    {
     //otherwise remove the reservation which will bring to an
empty-domain variable
     currNode.removeChild(wrongNode)
     //and choose a new assignment with the new cycle iteration
    }
   }
  }
  else
  {
   //otherwise we move to the chosen child
   currNode = currNode.getChild(index);
  }
 }
 //check if exploration has been successful
 if(currNode == null)  //exploration has failed
  return null;
 else  //build the reservations collection
  return buildRetCollection(currNode);
}
/**
 This method build up the collection of reservations which
compose a scheduling change by exploring the tree backward,
starting from a red leaf and going to the root
 @return the reservations collection or null if this method is
invoked on the tree root
*/
private Collection<Reservation> buildRetCollection(RedNode
leaf)
{
 if(leaf.getFather() == null)
  return null;
 Collection<Reservation> ret = new Collection();
 while(leaf.getFather() != null)
 {
  ret.add(leaf.getReservation());
  leaf = (RedNode) leaf.getFather().getFather();
 }
 return ret;
}
/**
 this method will visit the given node and choose the next
children to visit, if there are any.
 Also, it will expand the chosen child, preparing it for the
upcoming visit
 @param target : the node to be visited
 @return the index of the children chosen to be explored
*/
private int visitNode(_Node target)
{
 //retrieve the children list
 Collection<_Node> candidates = target.getChildren();
 //build additional data structures for candidates ranking
 Iterator<_Node> iter = candidates.iterator();
 int[] score = new int[candidates.size()];
 int index = -1;
 //verify it is not empty
 if(candidates.isEmpty())
  return index;
 //if is a RedNode choose the appointment to be scheduled on
the next step
 if(target instanceof RedNode)
 {
  //Examine each appointment to assign a score to it
  for(int i=0; i<score.length; i++)
  {
   BlackNode curr = (BlackNode) iter.next();
   //if needed build its children list
   if(!curr.hasBeenExpanded())
   {
    curr.buildChildrenList();
    curr.setAsExpanded();
   }
   //the score is equal to the size of its appointment's domain,
which is equal to the size of its children list
   score[i] = curr.getChildren().size();
   //if the BlackNode has no children this reservation can not
bring to a feasible schedule
   //signal failure, adopting a Fast-Fail strategy
   if(score[i]==0)
    return -1;
  }
  //choose the index of the candidate to visit
  index = chooseCandidateIndex(scores, scores.length);
  //the chosen child is already expanded
  //do not collapse the unvisited nodes: in a time-vs-memory
tradeoff time must be favored
 }
 else //otherwise choose the reservation to assign to the
appointment encapsulated in this node
 {
  //fetiching V1 and V2 lists
  BlackNode target2 = (BlackNode) target;
  Collection<Appointment> V1 = target2.getV1();
  Collection<Appointment> V2 = target2.getV2();
  //conflicts of each assignment will be saved here
  Collection<Appointment> conflicts[] = new
Collection<Appointment>[score.length];
  int nElem = 0;
  //examine all the candidates
  for(int i=0; i<score.length; i++)
  {
   RedNode curr = (RedNode) iter.next();
   Collection<Appointment> currConf = new Collection();
   //for each candidate verify that there is no conflict with the V1
set
   currConf = findConflicts(curr.getReservation(),V1);
   //if there is no conflict the reservation can be candidated
   if(currConf.isEmpty())
   {
    //test the reservation for conflicts with V2...
    currConf = findConflicts(curr.getReservation(),V2);
    //...and save these conflicts in the collection
    conflicts[nElem] = currConf;
    //save the number of conflicts in the score value
    scores[nElem] = currConf.size();
    //increment the saving sentry
    nElem++;
   }
   //otherwise this reservation is ignored
  }
  //choose the index of the candidate to visit
  index = chooseCandidateIndex(scores,nElem);
  //if candidates generate no conflicts add the appointment
encapsulated into this node to V2
  if(conflicts[index].isEmpty())
   V2.add(target2.getAppointment());
  //otherwise add it to V1
  else
   V1.add(target2.getAppointment());
  //build the children list for the chosen candidate
  RedNode cand = (RedNode) target2.getChild(int index);
  //add the old unassigned appointments...
  cand.addToChildren(target2.getBrothers(),V1,V2);
  //...and the recently unassigned ones
  cand.addToChildren(conflicts,V1,V2);
 }
 //return the index of the chosen candidate
 return index;
}
/**
 Returns all the appointments of the given collection which
conflict with the given reservation.
 This is a trivial method: a nested cycle tests all appointments for
each constraint, saving them into the return collection if needed,
thus it's not shown.
 @param res : the reservation to be examinated
 @param test : the appointments to be tested for conflicts
 @return a collection X of the appointments which conflicts with
the given reservation. Mind that X ⊆ test.
*/
```

private Collection<Appointment> findConflicts(Reservation res, Collection<Appointment> test);
/**
 This method examine the given score sequence and choose an index from it.
 The behaviour of this method is defined outside of this source.
 @param scores : an array with the score value for each candidate, in the same order in which candidates have been examinated
 @param nElem : the number of elements contained into the scores array. The array, in fact, may not be full.
 @return the index of the candidate chosen among the available ones.
*/
private int chooseCandidateIndex(int[] scores, int nElem);

```
Code block end
```

We have purposely left unexplained the behaviour of the *chooseCandidateIndex* method in the code block.

This has been done because this method is what really differentiates the deterministic scheduling from the stochastic scheduler.

In fact the deterministic scheduler use the heuristics explained in Section 5.2.3, which means that the behaviour of the unexplained method is equal to the one of the following expression[7]

$$\arg\min_{index}\{score[index]\} \;:\; index \in [0, score.length-1]$$

On the other hand, the stochastic scheduler uses the passed *scores* array to compute an appropriate probability distribution for the index to be returned: in the implementation used for the experiments performed in Chapter 6 the probability for each index to be picked was directly proportional to its goodness for the appropriate heuristic (i.e. Min-Conflict for RedNode picking step and Fail-First for BlackNode picking step).

The code, as shown, has great modularity in this sense: other strategies to choose the options, in a deterministic or probabilistic way (like *simulated annealing* [6]), can be implemented by simply overriding the *chooseCandidateIndex* method.

5.2.7 Summary

A stochastic exploration of the solution space clearly do not ensure that a solution better than the one we can find with the deterministic search policy will be obtained. The resolving algorithm, in fact, is not meant to use only stochastic explorations: this approach is actually meant to be an extension to the already designed deterministic algorithm.

7 *Argmin* stands for the *argument of the minimum*, that is to say, the set of index for which the value of the given expression attains its minimum value.

To get the best overall performance, thus, we combine both methods: given a reschedule to be done, first a deterministic solution can be found and subsequently a finite number of stochastic explorations can be made; the limit of these explorations can be set in advance, but being the *stochastic local search* an anytime algorithm it is advisable to let the stochastic explorations run as much as possible, i.e. until the beginning of next time quantum, when new appointments are likely to be rescheduled.

Every stochastic run has the chance to improve the solution found, following the terms explained in paragraph 5.2.1; if that is the case we simply mark this solution as the best one and go on with the next run.

A transaction system is useful to manage the sequence of assignment generated by the stochastic run: every time the solution is improved we must note down that solution but we can **not** apply the assignment before all the stochastic runs are finished. This means that the changes will be committed only as the final step of the algorithm, updating the reservations just before the start of a new temporal quantum.

6 RESULTS

This section will presents the results of our experiments.

A standard benchmark procedure to compare the performance of our solution with those already explained in chapter 3 does not exists in the literature.

These comparisons, therefore, are made with the queue model we can infer from our exam of the hospital environment, which is very close to a First-In-First-Out scheduling policy without any possibility, for the scheduler, to reschedule already scheduled patients with a priority code equal or higher than the one which is currently trying to insert.

The solvers who participate in this simulation, thus, are the FIFO-like queue manager just described, and the algorithm described in chapter 5. The full algorithm is composed by a deterministic exploration of the tree and subsequent stochastic explorations trying to improve the solution just found. In order to fully test this algorithm a first simulation will be done, for each experiment, using only the deterministic share of the algorithm and in the second one the stochastic exploration will be included.

The experiments we made are of two different types: offline scheduling and online scheduling.

The first type aims to simulate a rescheduling request for a certain number of appointments at the same time, with relaxed time limits. This usually happens in response to a change known in advance, as a planned maintenance session for a device.

The second type of experiments, on the other hand, try to simulate the accesses to the hospital by the patients among a prolonged time period. This is the usual situation we have at the hospital bottlenecks, like the diagnostic center in the ER [23].

The hospital reality modeled includes 5 resources which offers 5 kind of different treatments. For resources that offers the same treatment, obviously, there's the possibility for the patient to use any of these resources to receive his or her treatment.

The computer used for this experiments has an Intel Core2 Duo T7100 CPU with 2 gigabytes of RAM memory and, during the experiments, every process had the whole hardware reserved for its computation. The prototype of the scheduler using the algorithm described in Section 5 has been implemented using the Java language and has been tested extensively in order to verify its robustness and its reactivity. Even with a Java implementation, which is known to be disadvantageous in terms of execution speed, the imposed time limits have been always respected.

6.1 METRICS

To evaluate the efficiency of each solver, however, we must define a set of metrics, in order to quantify the performance for the comparison.

The metrics we use are the following, listed in descending order by the priority level associated to their optimization:

1. Refused Patients percentage (**Ref%**): this metric is computed by counting the number of appointments which could not be scheduled and subsequently dividing this quantity by the number of appointments which we tried to schedule.

   We aim at keeping this metric, obviously, as low as possible, which is the primary goal of our algorithm. The differences between other metrics will be considered significant only in case of equality of results obtained in this metric.

2. Relative Solution Size (**Size%**): this is the number of rescheduling operations performed while computing the current solution divided by the number of appointments actually scheduled. This value is, obviously, always equal or greater than 1 and minimizing it helps the hospital organization and improve the patient satisfaction. This metric is calculated only for successful scheduling attempts.

3. Resource Utilization percentage (**RU%**): this metrics is computed by counting, for each resource, the number of time slots in which the resource is engaged and subsequently dividing this number by the cardinality of its restricted availability set. A restricted availability set of a workplace w it is a proper subset of its availability set $T^r_{avail} \subseteq T_{avail}$ such that $\forall t \in T^r_{avail}$ exists an assignment a scheduled in a fixed day such that:

   See definition 5.2 for the "availability set" definition

   $$a.w = w \wedge (a.t_{start} \leq t \vee a.t_{start} + a.\Delta t_{dur} \geq t)$$

   In this way we consider into the computation only the period encompassed by the arrival of the first patient and the leaving of the last patient for each given day

4. Running time ($\mathbf{T_{elab}}$): this is the actual time needed for every solution's computation, in seconds. It is not an important information, given its dependency on the hardware used for computation, but knowing that all the simulations has been run on the same conditions we can obtain interesting information by comparing these results one with each other. The FIFO policy will, obviously, have the lower of these values but to achieve this result it sacrifices all the other aspects considered by more sophisticated algorithms.

6.2 OFFLINE SCHEDULING

This kind of simulation is made with a complete knowing of the set of appointments. This is unlikely to happen during high-intensity activity period but a similar process can be launched at night to optimize the appointments scheduled for the next day (or days).

Different problem aspects are applied to test the algorithm robustness: various appointments load, restricted or unrestricted computation time allowed and presence of various appointments that can not be rescheduled.

To have significant data for our results and knowing that a significant portion of the computation is stochastic in nature, 20 different problem instances have been sequentially given to the solvers.

In this section we will summarize the average results obtained for each metric by each solver.

6.2.1 Simulation n.1

In this simulation a low quantity (50) of patients have been inserted with no already scheduled appointments and a restricted computation time for deterministic and stochastic explorations of 30 minutes.

The results are summarized in Table 1

Metrics	FIFO	Det. only	Full
Ref %	30,62%	0%	0%
Size%	100%	121,92%	117,78%
RU %	99,29%	84,43%	72,14%
T_{elab}	0,1329	69,1015	77,6164

Table 1: Restricted, low-load offline performance

We can immediately notice a remarkable improvement on rejection appointments ratio compared to the one obtained by the standard scheduling policy and, with the stochastic extension described in Section 5.2.5, subsequent improvements of the relative solution size found with the deterministic exploration are obtained.

This experiments clearly highlight that the algorithm responds as desired, optimizing its objectives in the correct order. The algorithm, in fact, ranks the solution found by examining first the fraction of rejected appointments and, only afterward, the Relative Solution Size. The Resource Utilization Ratio, which is third in our priority ranks, is clearly neglected while trying to minimize the number of reschedulings needed.

Seeing the already very good results obtained with this simulation there will be no simulations with more computation time allowed because performance can only improve in term of Relative Solution Size, which is a secondary objective.

6.2.2 Simulation n.2

In this simulation a high quantity (150) of patients have been inserted with no already scheduled appointments and a restricted computation time for deterministic and stochastic explorations of 30 minutes.

Results obtained, as expected, are not satisfying. In this extreme scenario, in fact, the exponential nature of the problem becomes perfectly clear: a great load of work needs computation times far greater than the ones required by the low-size problem instances, and 30 minutes is simply not enough time to solve the problem instance.

The simulation has been repeated, thus, with a more relaxed computation time of 2 hours, obtaining more promising, but still not good enough, results.

To complete this heavy-load simulation we relax the time restrictions to 8 hours (one night), granting as much time as possible to the optimizers.

Metrics	FIFO	Det. only	Full
Ref %	36,5 %	24,66 %	24,33 %
Size%	100%	689,37%	764,97%
RU %	91,6 %	94,17 %	94,34 %
T_{elab}	0,2987	2880024,2	2880032,7

Table 2: Least restricted, high-load offline performance

Results listed in Table 2 are quite good: using the whole night the tree-exploring optimizers have produced results that outperform the standard scheduling policy.

We can summarize these results with three different plots, having a look at the overall trends.

The interesting value we must examine first is the refused appointments rate, which we can see in the plot represented in figure 3.

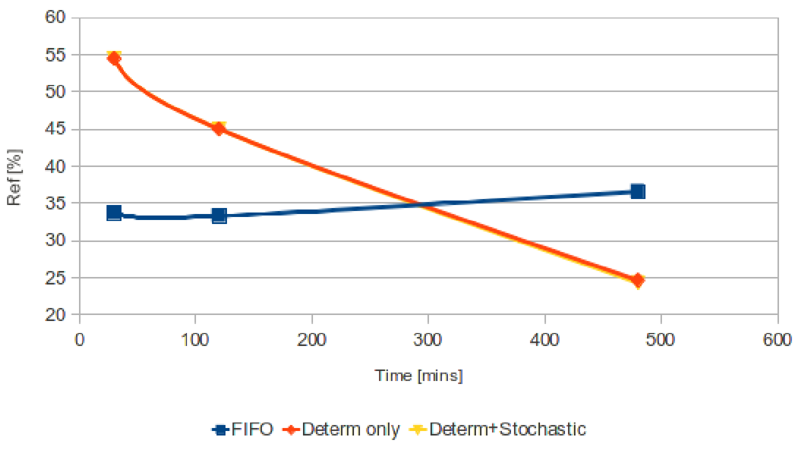


Figure 3: High-load offline performance: Ref % trend

This is surely a good result: we see that, which enough time to work on the schedule, the rejection rate drops to a more than acceptable level, remarkably improving the performance obtained by the standard schedulers and respecting time limits we fixed. For an hospital this is a great success: the lesser appointments it has to reject the more patient it can satisfy, which is the mission-critical goal of every hospital.

Furthermore, we can see how these appointments have been included into the actual planning by examining the Relative Size Ratio trend, represented in figure 4.

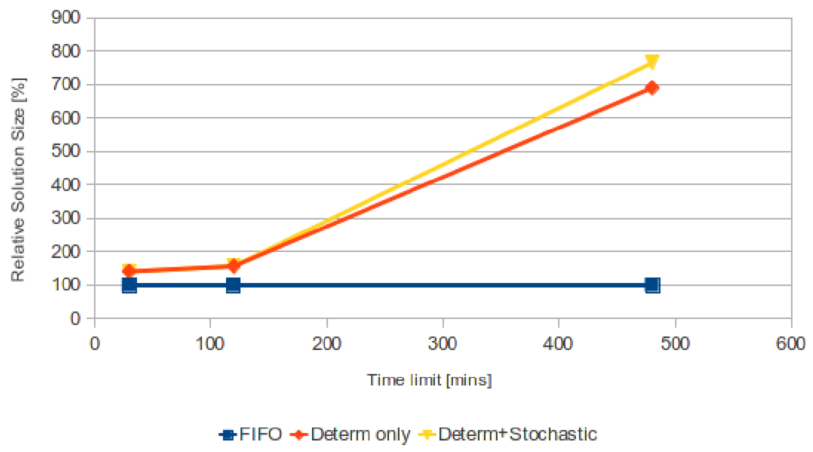


Figure 4: High-load offline performance: Size % trend

Obviously for subsequent adjustments some of already scheduled appointments have been rescheduled, and we try to minimize the number of these reschedulings. We must also keep in mind, however, that these reschedulings clearly bring the benefit of a higher patient throughput, so this is a goal which is good to pursue, but is clearly a secondary goal for the hospital objectives. A similar trend suggest that, as the scheduled patients number increase, make new appointments fit in the schedule is more and more harder: for the best patient throughput achieved the number of rescheduled appointments is nearly 8 times more than the actual inserted appointments. This is clearly influenced by the heavy-load scenario simulated in this experiments and, for the remarkable 33,33% improvement on the patient rejection rate, this is surely an affordable drawback.

Finally we can examine the resource utilization ratio trend, represented in figure 5: this trend shows clearly an aptitude of the DCSP-based optimizers to fill-in the resources more and more over time.

This plot suggest that the resource utilization ratio seems to benefit from the use of a DCSP-based scheduler much less than the rejected appointments ratio. We have to remind, however,

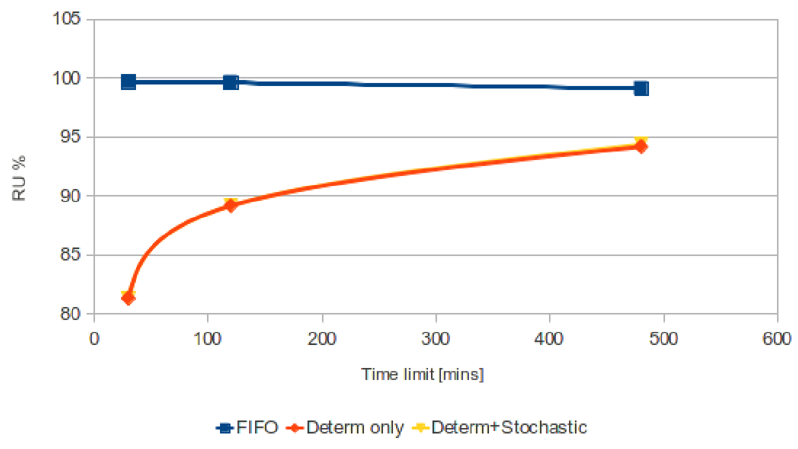


Figure 5: High-load offline performance: RU % trend

that this metric is calculated on the local intervals where the scheduler acts, i.e. this metric measure how the scheduler place the appointments only in the time periods where they *actually place* appointments. This mean that a high appointment rejection ratio will make this value rise, because the less appointment are scheduled the small is the time period where this metric is actually calculated onto.

6.2.3 Simulation n. 3

In this simulation a low quantity (50) of patients have been inserted with already scheduled appointments up to approximately 50% of the hospital capacity.

As the previous step a restricted computation time for deterministic and stochastic explorations of 30 minutes has been set for the first simulation and, by gradually relaxing it, we can obtain the results summarized in Table 3.

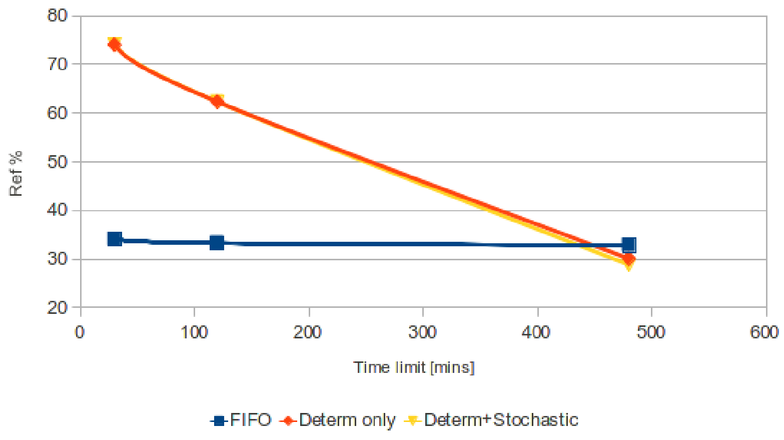
Metrics	FIFO	Det. only	Full
Ref %	32,89 %	30,12 %	28,86 %
Size%	100%	233,3 %	294,62%
RU %	91,21 %	94,42 %	95,74 %
T_{elab}	318	28800015,47	28800073,56

Table 3: Least restricted, low-load offline preloaded performance

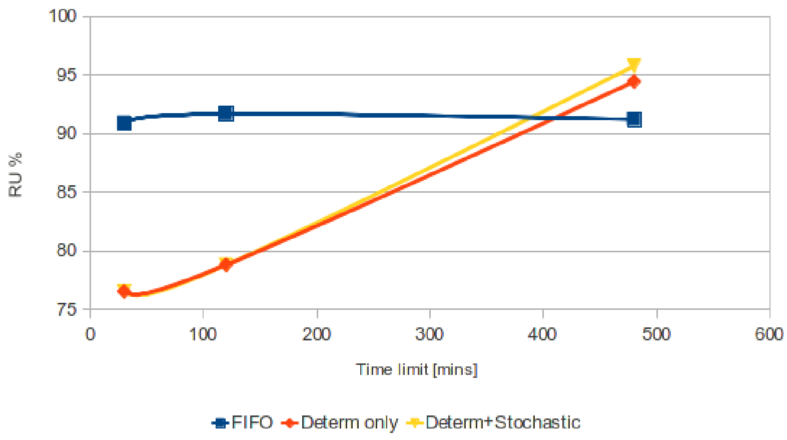
We can summarize again these results with the plots represented in figure 6.

From the plot in figure 7a we can see that the refused appointment ratio, with the appropriate computation time limit, can perform lightly better than the one obtained with the standard schedulers, allowing more patients to be inserted into the plan.

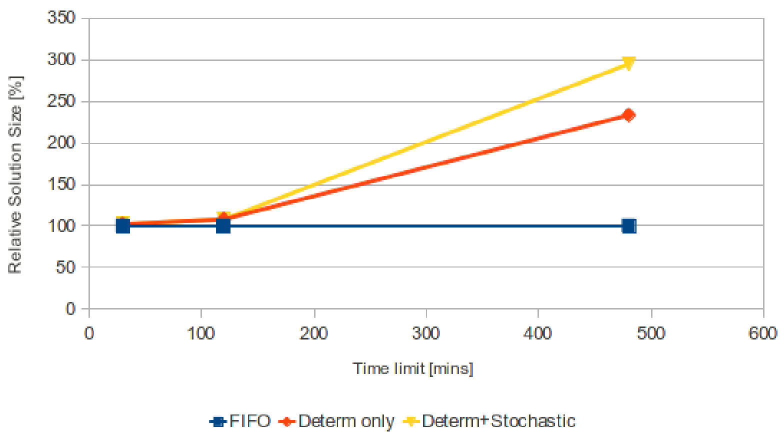
Figure 6: Low-load offline preloaded performance plots



(a) Ref % trend



(b) RU % trend



(c) Size % trend

Seeing that the time limits have been respected this is, again, a no-cost improvement, even if it is relatively small (about 12,25%)

The plot in figure 7b remarks the effectiveness of the optimization routines: with high loads of appointments to optimize the DCSP-approach can exploit the free time slots of the planning, and a new appointment can be squeezed in by rescheduling his neighbours.

The last plot of the figure show us the relative solution size. Again: to achieve the best performance a perceptible quantity of rescheduling must be done. This time, given that the size of the appointments set is about 2/3 of the one of the previous simulation, the number of rescheduling is much smaller. This remarks again the exponential nature of the problem: by increasing the size of the problem by 1/3 the number of needed rescheduling is 2,6 times higher.

6.2.4 Simulation n. 4

In this simulation we try to completely fill the hospital with our optimization routines.

A high quantity (100) of patients, thus, have been inserted with already scheduled appointments up to approximately 50% of the hospital capacity. Due to the extreme quantity of patients and to results previously obtained this simulation has been done only with an 8 hour time limit.

Metrics	FIFO	Det. only	Full
RU %	99,24 %	99,5 %	99,5 %
Ref %	36 %	26,72 %	26,72 %
T_{elab}	576	28800980	28800160
Size%	100%	475,24 %	462,78 %

Table 4: High-load offline preloaded performance

performance is similar to the one listed in table 4 with the exception that resource utilization ratios, which are naturally high due to the fact that a previous optimized scheduling already exists before the simulation start.

6.2.5 Conclusions

Performance are very good for light workloads: the Rejected Appointments Ratio clearly benefits from the adoption of a more sophisticated scheduling performance, reaching optimal performances in this case.

We can also notice a little difference between the deterministic and stochastic results: a purely deterministic method will naturally find a solution with high Resource Utilization Ratio but subsequently stochastic explorations of the decision tree can improve the Relative Solution Size, that takes precedence over the *RU%*, by rescheduling a lesser number of patient while still granting a more than acceptable resource utilization ratio.

This is a really good results: in fact we can logically assume that a patient does not like to see its appointment postponed, even if it is done to make place to an emergency life-saving operation. With the first simulation we have verified that with the stochastic explorations of the decision tree the number of postponed appointments can be reduced, and knowing that the limited time we have set was more than enough for these explorations this benefit can be obtained with no additional costs in these conditions.

The DCSP-based scheduler outperform standard schedulers with heavier workloads too: in fact every experiments clearly show a reduction of the fraction of rejected appointments reached with long running times which still respect, however, the limits we set.

The ranking of the objectives is clearly respected: the main goal of the algorithm is to minimize the Rejected Appointment Ratio and, only if this is optimal, the other metrics are considered. In particular we can see that by raising the workload size the performance of the scheduling optimizers, in term of rejection appointments ratio, converges with the one of standard scheduling policies but, in these cases, other goals are pursued once realized that the prioritary metric can not be improved further.

6.3 ONLINE PLANNING

This kind of experiment is done in order to simulate an ER system access during a prolonged period of time: the flow of incoming patients is simulated as a Poisson process and strict time restrictions are applied, in order to respect the assumption that the scheduler can only use the "dead times"[1] to perform the planning optimization.

In this case a policy similar to the one of the previous chapter is also applied, in order to test the algorithm robustness: except for the computation time restrictions, which are always strict by assumption, various appointments load have been tested, modifying the λ parameter proportionally to the number of resources N in the Poisson process, and the presence or absence of already scheduled appointments have been considered.

6.3.1 Simulation n.1

In this simulation a low-intensity Poisson process ($\lambda = N/4$) has been simulated, with no appointments already scheduled. This is to show how all the scheduling policies works with a

1 Dead times are the time periods where all the resources are engaged and all jobs are waiting or are currently being served

low quantity of patients to be scheduled. Results are reported in Table 5.

Metrics	FIFO	Det. only	Full
Ref %	4,64 %	3,31%	3,31%
Size%	100%	100,05%	100,05%
RU %	59,79 %	59,92%	59,92%
T_{elab}	0,0105	0,2819	39,4178

Table 5: Low-intensity online insertion performance with no previous load

The first interesting information we can extract from this table is the minor contribution of the stochastic search paradigm: in such scenarios, where simpler solution are also the better ones, the deterministic explorations are sufficient to obtain a very good solution.

We can also see that the rejection rates are low for all the approaches: obviously when there is little or no need for appointment rescheduling an approach that optimize the rescheduling process is of a limited utility.

6.3.2 Simulation n.2

In this simulation a medium-intensity Poisson process ($\lambda = N/2$) has been simulated, with no appointments already scheduled.

Results are reported in table 6

Metrics	FIFO	Det. only	Full
Ref %	18,12 %	10,48%	10,48%
Size%	100%	100,8%	100,8%
RU %	76,13 %	78,55%	78,55%
T_{elab}	55,75	70,6238	612,4518

Table 6: Medium-intensity online insertion performance with no previous load

Performance is satisfying: the arriving of about N/2 patients every time quantum can be perfectly handled by the optimizers, which can reduce the refused patient ratio by over 42%.

Knowing that a treatment can last more than a single time quantum, not counting change times of workplaces, we can imagine that in more heavily loaded situations a similar arrival ratio can be unmanageable for the hospital.

6.3.3 Simulation n.3

In this simulation a medium-intensity Poisson process ($\lambda = N/2$) has been simulated, with already scheduled appointments up to approximately 50% of the hospital capacity.

Metrics	FIFO	Det. only	Full
Ref %	86,58%	84,63%	84,12%
Size%	100%	108,64%	118,12%
RU %	97,59%	98,86%	99,1%
T_{elab}	186	8124,501	16325,408

Table 7: Medium-intensity online insertion performance with low previous load

Results are summarized in Table 7

Performance here is still satisfying: due to limited hospital capability the number of refused patient is high but we can see that the optimizers can still squeeze some more patients into the scheduling, compared to standard schedulers, while respecting the fixed time limits.

There is no need to test the schedulers with more preload and the same patient arrival ratio but we can still make one last experiment to test algorithm capabilities.

6.3.4 Simulation n.4

In this simulation a high-intensity poisson process ($\lambda = N$) has been simulated, with no appointments already scheduled.

This is another high-stress situation, where the hospital will surely clogs, but our target is to see how schedulers can manage a such a great patient flow.

Results are summarized in table 8

Metrics	FIFO	Det. only	Full
Ref %	91,36 %	91,27 %	91,27 %
Size%	100%	117,28 %	117,27 %
RU %	99,87 %	99,92 %	99,92 %
T_{elab}	778	11705,376	19325,408

Table 8: High-intensity online insertion performance with no previous load

We can see we have given the schedulers a critical situation: an income ratio comparable to the number of resources is overwhelming for the hospital in a way such that we can spot no sensible differences between the schedulers' performance because there are no free slots to insert the patients into, as we can see from Resource Utilization Ratios.

In fact the incredible number of patients fill up almost every free slot even with the standard scheduling policies, and eventual stochastic improvements to deterministic explorations (-0.1% of relative solution size) are definitely not worth the computation time needed to find them.

6.3.5 Simulation n. 5

In this part we have tried to simulate a single emergency arrival: one urgent patient has been inserted with already scheduled appointments up to roughly 95% of the hospital capacity.

This simulation represents a realistic case in a heavily loaded ER department, and will test the schedule-rebuilding capability of the scheduler optimizers we designed.

Results are summarized in Table 9

Metrics	FIFO	Det. only	Full
Ref %	75%	40%	40%
Size%	100%	866,67%	841,67%
RU %	98,675%	98,925%	98,925%
T_{elab}	8,25	317,6012	496,30945

Table 9: Unestricted, high-load unitary insertion performance

We can notice immediately that the DCSP-based schedulers, here, outperforms dramatically the standard one, reducing by near a half the number of refused appointments[2]. Also, despite having allowed only a single time quantum (15 minutes) as computation time to the solvers, we can notice that a similar perturbation is handled in acceptable time (from 5 to 10 minutes). All the resource utilization ratios are high, but this is naturally given by the appointments that are already scheduled, which nearly push the hospital capability to its limits, but another information is of particular interest for us: the relative solution size ratio.

As we can see the average solution size ratio is really high for the Deterministic and Stochastic schedulers, over 8 times the initial number of appointments to insert. Examining extensively the data, however, we were able to spot a single outlier solution. By excluding this value from the data aggregation the relative solution size ratios of the explorations are put down to 500% and 472,73%, for deterministic only explorations and deterministic and stochastic explorations respectively.

6.3.6 Conclusions

These experiments are highlighted that a reactive approach like the one we presented in Section 5 is suitable to handle a continuous flow of patients requests: in every experiment was noted an improvement on the Rejected Appointment Ratio obtained with standard scheduling policies.

In particular, the improvements granted by an optimized scheduling policies are present with all the tested workloads: if the algorithm has enough space to exchange already scheduled appointments, which can also be very limited, new patients can be

2 We remark that the solvers, in order to make place to the just arrived emergency, have the ability to dump a previously scheduled appointment

squeezed into the planning. This is particularly visible in the experiments described in Section 6.3.5: even with the hospital filled to about 95% of its maximum capacity a single urgent patient can be inserted in the schedule without rejecting any other patient.

It is also clear, however, that the intensity of this improvement decreases as the workload comes close to the maximum workload sustainable by the hospital: in these cases performances of the scheduling optimizers converges to the one obtained with the standard schedulers, because no scheduler can insert a patient in a schedule already filled.

7 CONCLUSIONS AND FUTURE WORK

In this thesis, we have presented a new dynamic approach for the Hospital Scheduling Optimization problem, modelling the hospital reality in a truly accurate way.

As we have seen, hospitals, being very complex systems, may be really suitable for automatic administration: their vast structure, manageable by human agents with suboptimal performance, can be handled with more precision and objectivity by automatic algorithms.

However, hospitals are hierarchic and bureaucratic environments, in which all the procedures currently adopted as standards have been deeply tested and monitored, in order to safeguard the health of the patients who come to the hospital each day. In fact, a proposal which could be attractive from a management point of view can be disastrous, if examined with a medical eye: doctors, therefore, always have the last word in term of procedural updates.

Due to these reasons it is difficult to implement innovative ideas while trying to improve the health care system; therefore scientific approach for resolution of the Hospital Scheduling Optimization problem are still rare among the scientific literature, while there are a bit more works which analyze some of the nearly infinite other open problems related to health care system.

In this work we have had a glance to some of them, analyzing the pros and cons of each one, and then we proposed a new generalized, versatile and flexible approach, based on the *Dynamic Constraint Satisfaction Problem*s framework. Compared to other approaches already present in literature the one we propose in this thesis is a highly reactive, dynamic approach, which is built on a reality model that resembles hospital environments with a high fidelity level. This approach grants to the proposed method the ability to handle effectively the dynamic hospital environment by promptly reacting to the changes typical of this environment (emergencies, machinery maintenance sessions, and so on) which can affect the resource pool, compromising a previously built schedule.

The *local changes* algorithm, in particular, is a good base to work on: the objective it pursue is very similar to our (i.e. try to insert a new task in a feasible schedule while trying to alter the already present scheduling as little as possible) and its architecture allows us to easily adapt it to our purposes. Furthermore, the greedy heuristics we inserted into the algorithm reflects the standard procedures already used in hospital to manage the patients flow, making any actual deployment easier to integrate into the hospital.

A really good feature of this system, in fact, is its scalability, especially in term of adding or removing constraints on patient and resources: due to the nature of the *DCSP*s, when a constraint is added or removed the network automatically switches in order to find a new consistent assignment, by propagating the change event to all the variables.

By modelling the incoming patients flow as a Poisson process we've been able to test the proposed method, obtaining many promising results: as a matter of fact the system is able to manage the appointments scheduling in a more than satisfactory way, granting better results than standard scheduling policies while respecting the time limitations intrinsic in the hospital mechanism. Furthermore, the stochastic explorations of the decision tree can exploit all the remaining available time after a solution has been found with deterministic strategies: in fact the algorithm used for these explorations is an anytime algorithm, which means that the proposed solution converges to optimal solution with the increasing of the time available for computation.

Knowing this, it is clear that the available time plays a crucial role in the algorithm; the performance measured and listed in chapter 6 confirms this intuition, by showing that better solutions are found with the gradual relaxation of the time limits. This implies that an efficient implementation of the proposed algorithm can improve the performance mentioned above even further.

Some steps to obtain this efficient implementation are already almost done: the adoption of a tree-based data structure, for example, take advantage of the classic "*memory-vs-time*" tradeoff, and can permit an easy translation of the recursive version of the *local changes* algorithm to an iterative version of the same algorithm; this will surely grant remarkable performance improvements, since the algorithm itself is called on the beginning of each time quantum in order to schedule any new requests.

With the flexible modeling instruments given by the DCSP more details can be easily added to the model: making the patient able to select one or more preferred time slots, for example, will put this approach on the line with the one described in [23] in term of functionality features. In this case, the current implementation could be extended to the new problem model with promising implementation approaches from the literature, such as evolutionary algorithms [9, 23].

BIBLIOGRAPHY

[1] A.Bellicha. Maintenance of solution in a dynamic constraint satisfaction problem. In *Applications of Artificial Intelligence in Engineering VIII Vol 2 Applications and Techniques*, pages 261–274, 1993.

[2] Berger B. and Cowen L. Complexity results and algorithms for $\{<, \leq, =\}$-constrained scheduling. In *Proc. Second Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 137–147, 1991.

[3] Peter Bosman, Jörn Grahl, and Dirk Thierens. Enhancing the performance of maximum-likelihood gaussian edas using anticipated mean shift. In Günter Rudolph, Thomas Jansen, Simon Lucas, Carlo Poloni, and Nicola Beume, editors, *Parallel Problem Solving from Nature - PPSN X*, volume 5199 of *Lecture Notes in Computer Science*, pages 133–143. Springer Berlin / Heidelberg, 2008.

[4] Peter Brucker. *Scheduling Algorithms*. Springer, 2001.

[5] Marinagi C.C., Spyropoulos C.D., Papatheodorou C., and Kokkotos S. Continual planning and scheduling for managing patient tests in hospital laboratories. *Artificial Intelligence in Medicine*, 20:139–154, october 2000.

[6] Rina Dechter and David Cohen. *Constraint Processing*. Morgan Kaufmann, 2000.

[7] Rina Dechter and Avi Dechter. Belief maintenance in dynamic constraint network. In *In Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 37–42, 1988.

[8] Keith Decker and Jinjiang Li. Coordinated hospital patient scheduling. In *Proceedings of the international Conference on Multi Agent Systems*, pages 104–111, 1998.

[9] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Professional, 1989.

[10] Hoogeveen H, Schuurman P, and Woeginger GJ. Non-approximability results for scheduling problems with minsum criteria. In *Proceedings 6th international integer programming and combinatorial optimization conference. Lecture notes in computer science*, volume 1412, pages 353–366. Springer, 1998.

[11] Markus Hannebauer. How to model and verify concurrent algorithms for distributed csps. In Rina Dechter, editor, *Principles and Practice of Constraint Programming*, volume 1894

of *Lecture Notes in Computer Science*, pages 510–514. Springer Berlin / Heidelberg, 2000.

[12] Markus Hannebauer and Sebastian Muller. Distributed constraint optimization for medical appointment scheduling. In *AGENTS '01 Proceedings of the fifth international conference on Autonomous agents*, 2000.

[13] Jane W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[14] D. S. Johnson M. R. Garey and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, May 1976.

[15] Stuart Johnatan Russel and Peter Norvig. *Artificial Intelligence, a modern approach*. Prentice Hall, 2010.

[16] T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. In *In Proceedings Fifth International Conference on Tools with Artificial Intelligence, 1993.*, pages 48–55, 1993.

[17] Ian G. Stiell, George A. Wells, Brian J. Field, Daniel W. Spaite, Valerie J. De Maio, Roxanne Ward, Douglas P. Munkley, Marion B. Lyver, Lorraine G. Luinstra, Tony Campeau, Justin Maloney, Eugene Dagnone, and for the OPALS Study Group. Improved out-of-hospital cardiac arrest survival through the inexpensive optimization of an existing defibrillation program. *JAMA: The Journal of the American Medical Association*, 281(13):1175–1181, 1999.

[18] Chang-Chun Tsai and Sherman H. A. Li. A two-stage modeling with genetic algorithms for the nurse scheduling problem. *Expert Syst. Appl.*, 36:9506–9512, July 2009.

[19] Ullman. Polynomial complete scheduling problem. In *Proc. fourth Symp. Operating System Principles*, pages 96–101, 1973.

[20] Pascal Van Hentenryck and Thierry Le Provost. Incremental search in constraint logic programming. *New Generation Computing*, 9:257–275, 1991.

[21] Gérard Verfaille and Thomas Schiex. Solution reuse in dynamic constraint satisfaction problem. In *Proceedings of the twelfth national conference on Artificial intelligence*, volume 1, 1994.

[22] I. Vermeulen, S. Bohte, S. Elkhuizen, J. Lameris, P. Bakker, and J. La Poutré. Adaptive optimization of hospital resource calendars. In Riccardo Bellazzi, Ameen Abu-Hanna, and Jim Hunter, editors, *Artificial Intelligence in Medicine*, volume 4594 of *Lecture Notes in Computer Science*, pages 305–315. Springer Berlin / Heidelberg, 2007.

[23] I.B. Vermeulen, S.M. Bohte, P.A.N. Bosman, S.G. Elkhuizen, P.J.M. Bakker, and J.A. La Poutré. Optimization of online patient scheduling with urgencies and preferences. *AIME 2009*, pages 71–80, 2009.

[24] Ivan Vermeulen, Sander Bohte, Sylvia Elkhuizen, Piet Bakker, and Han La Poutr. Decentralized online scheduling of combination-appointments in hospitals. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, 2008.

[25] Ivan Vermeulen, Sander M. Bohte, Koye Somefun, and Han La Poutré. Multi-agent pareto appointment exchanging in hospital patient scheduling. *Service Oriented Computing and Applications*, 1(3):185–196, November 2007.

[26] Ivan B. Vermeulen, Sander M. Bohte, Sylvia G. Elkhuizen, Han Lameris, Piet J. M. Bakker, and Han La Poutré. Adaptive resource allocation for efficient patient scheduling. *Artif. Intell. Med.*, 46:67–80, May 2009.

[27] Jan M. H. Vissers. Patient flow-based allocation of inpatient resources: A case study. *European Journal of Operational Research*, 105(2):356–370, 1998.