Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Magistrale in Ingegneria Informatica

A genetic algorithm for dynamic scheduling in emergency departments with priorities

Candidato:
Filippo Bergamin
Matricola 620736

Relatore:
Prof Silvana Badaloni

Correlatore:
Dot. Francesco Sambo

Anno Accademico 2011-2012

Un grazie speciale, specialmente a... tutti!

Grazie a chi mi ha aiutato e non ha mai chiesto nulla in cambio.
Grazie a chi mi ha ascoltato sempre e sempre capito.
Grazie a chi mi ha sopportato con le mie sfuriate e i miei problemi.
Grazie a chi con me ha condiviso questi cinque anni.
Grazie a chi con me ha condiviso fantastici momenti.
Grazie a chi mi ha avvicinato al cielo... sin quasi a poterlo sfiorare.

Senza di voi non sarei mai arrivato fin qui... non adesso... non così!
Grazie... grazie a tutti... grazie di cuore!

Un ringraziamento particolare...
Alla professoressa Silvana Badaloni per questa splendida opportunità.
Al dottor Francesco Sambo per i suoi preziosissimi consigli.

Grazie Albino, Franca, Marta e Stefano, la mia stupenda famiglia.
Mai come in questo periodo, mi avete supportato, convinto, spinto...

... grazie!

Abstract

A hospital is a very complex environment and its management is a hard task. The point we explore in this thesis is the management of the patient flow for the Emergency Department. The main contribution of the thesis is twofold: on the one hand, we design a model of the Emergency Department as close as possible to the real environment; on the other hand, we design a genetic algorithm for finding the optimal schedule of patients' care. The choice of this approach is due to both the dynamic nature of the environment and the tight constraints on computational time, which favour the use of an any-time algorithm. We show through simulation that the model is sound and that the genetic algorithm is effective for the scheduling problem and could be easily applied to a real Emergency Department.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nel nostro Paese la tutela della salute come diritto fondamentale dell'individuo ed interesse della collettività prevista dall'articolo 32 della Costituzione è garantita, nel rispetto della dignità e della libertà della persona umana, attraverso il Servizio sanitario nazionale. [1]

The hospital service is one of the most important services: the outcome of a well organized institution is the boost of the health level of the people. A hospital is a very complex environment and its management is a hard task: the number of departments, the differences in treatment structure, the variability of quantity and requirements of the patients, the large number of human and structural resources involved are only a part of the problem of the management.

Fixing our attention only on the management of the staff and the resources it is blatantly the big number of variables considered to make them work together efficiently. All these resources have to cooperate to obtain the best service both for the quality of treatments and for the utilization of the resources: this gives both image and economic advantages. Nowadays in particular the economical aspect, but also the image one, are fundamental in those countries (like Italy) in which the National Health Service is part of the National offer.

This complex environment is clearly difficult to handle and there are several studies, both from health care institutions and the scientific community, that analyse some aspects of the hospital operative cycle in order to improve different aspects.

Examples of the scientific community studies are nurse or physicians scheduling optimization methods [3, 4, 5, 6] which can account staff preferences in the planning process, which is a common open problem in hospitals. There are also studies on how to improve the survival ratio of patients, for example optimizing the de-fibrillation programs as explained in [7].

The point we will explore in this thesis is the management of the patient flow for a particular department: the Emergency Department. This is an example of great dynamism in hospital structures: there are not appointments so the number of patients,

[1] `http://www.salute.gov.it/ministero/sezMinistero.jsp?label=principi`

the arrival distribution, the urgency level are not known. There are some studies, like [8], that tries to make a prediction model in order to plan in advance the resource utilization, but the results show the great unpredictability of this environment.

Despite the importance that studies of this problem may have, up to now they have received mild attention in literature.

Nowadays, some studies, like [9, 2, 1], try to model the Emergency Department environment in order to obtain improvements for example in the quality of the service provided or in the operative condition of the staff. Other studies, instead, start from the scheduling of the resources: this is the case of several proposals on nurses or physicians scheduling, like [4, 3, 5].

Our purpose is to manage the patient flow not starting from an appointment based system, like [10, 11, 12], that is not suitable for an Emergency Department. It is worthwhile to observe that patient scheduling at the Emergency Department differs from the classical scheduling problem because of the unpredictability of the patients' flow.

Looking for a model in literature, we have found not many: in our opinion the management of this very dynamic environment requires a model with a great similarity to the reality. But, for example, the possibility to insert into the model every kind of resources, or to correlate the resources in different tasks and other important aspects are many times overlooked in other works.

In chapter 4 we will show the model and its similarity to the reality. We will also talk about its simplicity that permits to cover a great number of aspects. The main entities are the horizon, the patients, the resources and the tasks: the horizon is the time span in which the scheduler can assign a start time to all the patient tasks. All the couples of patient-task are tied by resources availability and time constraints.

Summarizing, this problem can be classified as a constrained optimization problem in which the variables are the starting time for the tasks of all the patients while the constraints involve the availability and the use of the resources, priorities and deadlines of the patients.

In chapter 5 we will show our strategy which use an any-time algorithm: the possibility to obtain partial solution satisfies the necessity to keep low the computational time. We have noticed some similarities with our problem and the timetabling one: the biggest difference is the dynamical environment. So, starting from the concepts explained by Alberto Colorni, Marco Dorigo and Vittorio Maniezzo in [13], we will show that a genetic algorithm in which we have designed a new and well made structure for the chromosomes, a crossover operator and two mutation operators gives us some advantage.

In chapter 6, we will show the architectural choices both for the model representation and the genetic algorithm implementation and we will explain our choice to use `C++` to realize the software.

Finally, we will show the outcome of our tests which highlights the goodness of our choice to use a genetic algorithm to solve the problem. The low computational time and the low average cost of the solutions in output make our prototype suitable to be applied in a real hospital environment.

Chapter 2

The Problem

In this chapter we will introduce the domain of the problem. To do this it is important to introduce briefly the agents that act in the Emergency Department. Thus we will present resources and patients, together with some problems of interest.

2.1 The problem domain

The Emergency Department is a hospital department which provides medical and surgical care to patients, with illness and/or injuries, arriving to the hospital in need of immediate care. The service is characterized by a high variability of patient arrivals, depending both on time and on randomness (e.g., accidents, weather, epidemics). Notwithstanding arrival characteristics, it shall satisfy hard requirements for what concerns quick (sometimes immediate) response, also in case of patient congestion. The waiting time (that elapses between the patient arrival and the start of his process of care) is heavily influenced by the quantity and capability of the resources, but also by their optimal utilization.

2.1.1 Patients

As mentioned above, patients are very heterogeneous, because their requests and characteristics can vary widely. Another point is this: patients that arrive at Emergency Department do not pass by a general practitioner and has to be diagnosed as soon as possible. Obviously, if a car accident happens, 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. So, at least for urgencies, there are some minutes to prepare the resources but this is not taken for granted.

Triage

After the diagnosis is made an emergency code is associated to the patient. The process that brings from the diagnosis to the code is called Triage. This procedure

assigns an order to the patients. In Italy, according to a conference of the 25th October 2001 between the State and the Regions, the possible codes for the Triage are four.

- **Red Code** It is assigned to patients for which there is a physical breakdown of one of the three vital functions (respiratory function, cardiovascular function or nervous function).
- **Yellow Code** It is assigned to patients with great problems and with the possibility of an alteration of the vital functions.
- **Green Code** It is assigned to patients with problems that do not involve vital functions.
- **White Code** It is assigned to patients that could be managed by a physician of general medicine.

The red code patients have got the priority over all other patients that are waiting: they should start their treatments immediately. For yellow patients, the waiting time should be at most 10 minutes, but in those cases in which they have to wait, their conditions should be reconsidered every 5-15 minutes. Green patients start their treatments after all the red and yellow patients. Their conditions should be reconsidered every 30-60 minutes. White patients do not have a maximum waiting time but to calm down the patient, some personal should reconsider their conditions every 60-120 minutes.

2.1.2 Resources

The resources of a hospital include the staff, the machineries and the rooms. All of this resources are limited both in time and capacity: some of them, like staff, can accomplish only a patient at a time, some others, like rooms, may accommodate more people simultaneously. A resource is typically used by several patients with different properties and requirement: the resource has to give the right response to all this requests.

It is also true that to manage a peak in patient flow it is not possible to increase the resource quantity or to request to the staff to work frantically. In the first scenario, the cost of the solution is too high while in the second the quality of service decreases consistently. The hospital staff should take benefits from the reorganization of the planning method.

To allocate hospital resources, there must be both an appointment management system and a random flow management system. It is also important to understand that some resources are useful during day shifts (e.g., rehabilitation rooms), while others are necessary 24/7 (e.g., CAT-scan machines).

Chapter 3

State of the art

Scheduling problems are largely studied. Only in Hospital field there are several different applications that could involve nurses, physicians and patients. It is important to underline that patient scheduling at the Emergency Department differs from the classical scheduling problem because of the unpredictability of the flow of patients.

Scheduling deals with the allocation of the patients to the treatments over time, by respecting precedence, duration of the treatments' sequence and incompatibility constraints, in order to achieve the optimal using of resources or the optimal accomplishment of tasks.

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

3.1 Algorithms for Hospital scheduling

A study of the Health Service of Lazio Region, [14], gives some indicators that are of interest in order to increase the quality of service. An interesting parameter is the mean time in which a patient overstays in an Emergency Department: for Lazio Region, the range is between 41 and 385 minutes. In our opinion an intelligent management of the patient flow could reduce noticeably this time.

As some other studies confirm, the management of the inpatient flow in an Emergency Department gives great advantages both for the quality of the cares and for the resources utilization. Some approaches to the problem, like [15, 9, 2], start from a model to simulate the environment and to define the right resource allocation. Some other studies, [16, 17], instead, consider the resources as a precondition and they try to schedule the patients in order to optimize factors like the resource utilization or the patient mean waiting time.

In literature several papers on resource scheduling in Hospital environment can be found. Some of the most considered topics are nurse scheduling [3, 4, 5] and physician scheduling [6]. For patient scheduling, instead, greater attention is devoted to the management of appointments, also according to statistical data of the emergency flow, in

order to prevent overloads due to sudden peaks of emergencies and this is an hard task because the number of variables and constraints is very high.

As mentioned above, we are interested in problems of direct emergency flow management. The complexity of these problems lies in the dynamism of the process and in the structure of the Emergency Department, as stated in [17].

In the following sections, we will present four studies that will serve as a reference for our approach.

3.1.1 A Dynamic Patient Scheduling at The Emergency Department in Hospitals

The work, explained in [17], was made by Amani Daknou, Hayfa Zgaya Slim Hammadi of the Ecole Centrale de Lille and Hervé Hubert of the Institute of Engineering in Health department of the Universitè de Lille 2. The study analyses the problem with a wider view: it starts from the creation of the record for a patient, in which a specific algorithm makes a diagnosis and generates the treatment plan, to reach the scheduling of the patient.

Architecture

The proposal is a multi-agent based architecture capable of handling all patients of Emergency Department in order to minimize their waiting time as well as the costs of care, with the respect of the quality of care. The model aims to make possible the identification of a medical actor available at a given time and to assign to him, depending on both of the patients' flow and the medical practices, a set of tasks.

Their system is based on the interaction of five types of software agents: the Home Agent (HA) that ensures the patient's host and the creation of medical records, the Identifier problem Agent (IdA) that receives medical problems and identifies the skills needed for the treatment, the Scheduler Agent (SA) that makes the assignment of MSA agents to medical teams, the Monitor Agent (MA) that is in charge of the patient's become and the Mobile Staff Agents of medical team (MSA) that provides the treatment and the following of the patient.

At a time t, the arrival of a patient requires the creation of a HA and triggers the creation of an IdA, a SA, a MA and a number of MSAs. These same agents are available if they will take care of new patients otherwise they are created on request: after a period of inactivity $\Delta\,\mathrm{inf}$, i.e. in the absence of patients, the agents will be destroyed and they will be automatically created if it is necessary.

Algorithm

The main goal is to minimise the health state adjusted stay time of the patients, which is equivalent to an overall minimisation of suffering for the patients. The SA agent will apply a 3 Phase Reactive Scheduling Algorithm 3P-RSA which provides scheduling of operations care needing multiple skills. The main objective of the proposed method

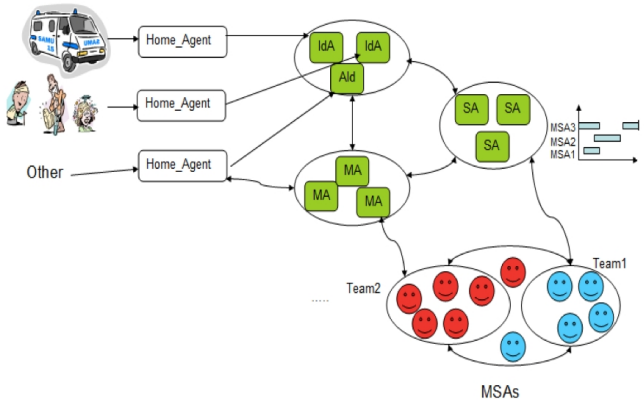


Figure 3.1: The multi-agent architecture.

of scheduling is to give the possibility to insert a new patient in scheduling which is in progress. The authors assume that material resources (the treatment rooms and equipment) are still available.

- **Phase 1: Scheduling**
  The first phase consists in making an initial scheduling of patients attending the Emergency Department at the moment t according to a priority rule based on algorithms of lists: among the available medical tasks, the task with highest priority is affected. The priority rule is dynamic and is estimated using intervention deadline before which operation must be allocated to the available medical staff.

- **Phase 2: Insertion**
  The second phase consists in exploring the feasibility of introducing patients arriving at the moment $t + \delta t$ in scheduling already established and with the highest priority. Firstly, the algorithm looks at the medical tasks of the new patient: if their insertion does not compromise the feasibility of the already established sequence, the algorithm ends in this second phase. However, if the insertion of emergency induced exceeded the scheduled time of others emergencies it is necessary to apply a method of rescheduling.

- **Phase 3: Rescheduling**
  The third phase aims to seek rescheduling of emergency allowing pre-emption for medical tasks in progress and that have a lower priority than the one that has just arrived.

The variables used are the following:

$$\mathrm{PB}_j(t) \quad \text{indices the urgency j, } 1 \leq j \leq \mathrm{PB} \text{ at moment } t \tag{3.1}$$

$$n_j \quad \text{is the number of ordered operation} \tag{3.2}$$

$$\mathrm{O}_{i,j} \quad \text{represent the operation } i \text{ for the urgency j} \tag{3.3}$$

$$\mathrm{P}_m,\ m \in \{1, \ldots, \mathrm{M}\} \quad \text{is the person assigned to } \mathrm{O}_{i,j} \tag{3.4}$$

$$\mathrm{d}_{i,j,k} \quad \text{is the duration of the assignment} \tag{3.5}$$

$$\mathrm{C}_k,\ k \in \{1, \ldots, \mathrm{K}\} \quad \text{is the skill mastered} \tag{3.6}$$

$$\theta_k \in [0, 1]\ \forall \mathrm{P}_m \quad \text{is the degree of knowledge} \tag{3.7}$$

$$dl_{i,j} \quad \text{is the deadline for the operation } i \text{ of the urgency } j \tag{3.8}$$

The resolution of this problem requires the allocation of each operation $\mathrm{O}_{i,j}$ to the person with the appropriate C_k and a minimum level of knowledge θ_k. Then, it must calculate both its start date $t_{i,j}$ and its end date $tf_{i,j}$ of execution. The *Objective Function* aims to minimize the waiting time between the arrival moment of the patients receiving at the ED and the first medical consultation.

Considerations

The tests proposed by the authors of this work show that their approach is able to reduce significantly the waiting time between home and the first consultation of 100 patients in ED. Anyway there are some lacks. Surely it provides a complete model but it assumes that the structural resources are available at any time. In addition, all the machineries (i.e. CAT-scanners, ...) are not considered, or better, they are inserted in the structural set.

There is another lack in the model: for medical staff it is not declared a shift system. Probably it is assumed that at any time there are the same quantity of staff so the change time is not considered.

The last point is the pre-emption of all the tasks: this is both a strong point and a weakness. It allows to increase the feasibility of the scheduling but, in reality the tasks that can be pre-empted are a minimal part: it is unthinkable that a surgeon would leave an operating room to start another surgical intervention that is more urgent.

3.1.2 A knowledge-based scheduling system for Emergency Departments

In the study proposed in [18] a knowledge-based reactive scheduling system is proposed to answer the requirements of Emergency Departments. The algorithm includes detailed patient priority, arrival time, flow time and doctor load. The main aim is to determine the patients who have higher priorities initially, and then minimize their waiting times. To achieve this aim, physicians and the other workers related to them can use an interactive system. This study was performed by scheduling of ED patients to minimize patients' waiting times and balance doctor loads. Because of the dynamic environment

of ED, a knowledge-based system is proposed for entering data, evaluating the priorities, developing and updating the schedules of patients in ED.

Algorithm

As mentioned above, the aim is to take care of urgent patients as soon as possible. Nevertheless the patient's arrival time has importance. The priority system, named *A-F*, is divided in 6 categories: A for airway problems, B for respiratory problems, C for cardiac problems, D for neurological problems, E for trauma or extremity problems and F for triage 3 patients that are not urgent. These categories are evaluated using Analytic Hierarchy Process (AHP [19]) approach by an expert doctor. The evaluation factor is defined as risk and the pair-wise comparison is given in 3.1.

	A	B	C	D	E	F
A		2	2	3	7	9
B			2	3	7	9
C				2	7	9
D					7	9
E						7
F						

Table 3.1: Pairwise comparison: compare the relative importance with respect to risk.

When patients are admitted to ED, their data are entered to the system. Initially, patient's name, surname, ID, arrival time, priority (category: A, B, C, D, E or F) and estimated duration are used. The other required data can be added and updated later.

At the beginning, it is assumed that there are m doctors and n patients in the system. During the process, doctors take patients according to their workload.

Before starting with the explanation of the algorithm, it is necessary to know the meaning of some used variables.

$$f[j] \quad \text{is the flow time of } j^{th}\text{doctor's schedule} \tag{3.9}$$

$$f^*[j] \quad \text{is the schedule with minimum flow time} \tag{3.10}$$

$$n(f^*[j]) \quad \text{is the number of schedule with minimum flow time} \tag{3.11}$$

$$load[j] \quad \text{is the doctor load, and it is weighted by the patient's category} \tag{3.12}$$

$$load^*[j] \quad \text{is the schedule with minimum load} \tag{3.13}$$

As for equation 3.9, all the formulas are referred to the j^{th} physician. The proposed algorithm has got five steps.

Step 1

Rank patients in descending order based on categories (sorting algorithm)

Step 2

Rank patients in ascending order of arrival times within same categories (sorting

algorithm for arrival times within same categories to apply first come first served approach)

Step 3

Rank patients to minimize flow time by using minimum processing time first approach within same categories and arrival times

REPEAT

Step 4

Select the doctor schedule with minimum flow time, $f^*[j]$
IF $n(f^*[j]) = 1$ THEN
Assign the next patient to the schedule with minimum flow time
ELSE
Select the doctor schedule with minimum load, $load^*[j]$
Assign next patient to the schedule the schedule with minimum load

Step 5

$load^*[j] = load^*[j]+$ the i^{th} patient's load

UNTIL remaining patients becomes zero

Considerations

The primary aim of the work is to minimize the waiting times of patients according to their priorities. The results highlight that for patients who have higher priority, lower waiting time is provided. But also, the load of staff is considered and this factor can improve the quality of service. Indeed, motivation of staff can be increased because of the balanced loads.

This algorithm sounds like a greedy algorithm, thus sharing the limitations of this category of algorithms in situation of great uncertainty. Nevertheless, authors state that this kind of knowledge-based system can be used if arrival, property and duration of patients are uncertain.

Another evident lack is the model: in our opinion it is very limited. In a Hospital the staff is clearly the most important resource, and the aim to balance its load is significant. But there are a lot of other resources that give time constraints which can not be ignored. To give an example: an x-ray machine can be used by more than one technician. If one of this technicians is using the machine, the other can not execute a task that involve an x-ray exam.

3.1.3 Dynamic Constraint Satisfaction approach to Hospital Scheduling Optimization

It is the work of Marco Ventilii, [20], and it is the starting point of this thesis. For the scheduling algorithm he uses a Dynaminc CSP approach.

The model

The solution proposed in this work is substantially composed by seven entities.

Horizon A *horizon* $T = \{0, \ldots, \hat{t}\}$ 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.

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".

Diagnostic Unit A *diagnostic unit* is a triple $u = (W, AT, \hat{r}_{staff})$. W is the set of workplaces in this diagnostic unit, $AT = \{(at, \hat{r}_{day}) | (at, \cdot, \cdot) \in w.AT \wedge w \in u.W\}$ 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.

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.

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

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

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

Algorithm

The solution use a Dynamic CSP approach. To model this kind of algorithms it is necessary to define the existing variables and constraints.

Variables Ventilii says that the variables are the reservations because an appointment contains only the information about the appointment's aspects. The data of its assignment are actually stored in the reservation entity.

Constraints There are several constraints that involve both patients and resources. 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 others 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 that 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 it 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.

Dynamic CSP It is used a Dynamic CSP: 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 problem's constraints.

The chosen optimization criteria is to schedule the largest number of patients up to the Hospital capability maintaining the current schedule as unchanged as possible. To do this the proposal is to use two types of heuristics: *Fail-First* and *Min-Conflicts* heuristics.

Considerations

The results are divided by two categories: off-line scheduling and on-line scheduling. For the first category, tests report that the performances are very good both for light workloads and heavier workloads.

For the second category, as the workload increases, the scheduling converges to the one obtained with a standard FIFO.

In our opinion, the model proposed is good for the appointment management. For the purpose of manage an emergency flow, a better representation of the reality is requested.

3.1.4 Short-Term Forecasting of Emergency Inpatient Flow

This work, presented in [8], in our opinion is useful to clarify the high uncertainty climate that characterizes the Emergency Department environment. The proposal of the authors is to modulate the appointment flow using a model of the emergencies flow. Elective patients are booked days or weeks in advance, whereas emergency patients arrive in an unplanned fashion, beyond the control of the Hospital, and driven by factors such as illness patterns, time of day, and socio-demographic effects. If electives are booked assuming a certain level of emergency demand, and it turns out to be an underestimate, then the two streams conflict. This implies a management of the emergencies: if in some periods of the year there are more probability of peak in emergencies, the reduction of the appointment rate allows to have more free resources.

The models are applied to three years of daily data. By measuring their mean square error in a cross-validation framework, they find that emergency admissions are largely random, and hence, unpredictable whereas emergency occupancy can be forecast using a model combining regression and autoregressive integrated moving average (ARIMA) model, or a seasonal ARIMA model, for up to one week ahead.

Models

Patients arriving at the ED are seen first by a triage nurse who ranks them by urgency, and later by a doctor who assesses and initiates treatment of their complaint. Some will then be discharged or transferred directly from the ED, a few seriously ill patients will die in the ED, and the rest will be admitted to a ward.

For the study, the authors use different models in order to make a comparison between the obtained results. Some brief introductions to the various models are presented in what follows. The entire ones are visible in [8] at pages 381-384.

- **Moving Average Forecasting**
  In the k-step moving average model the one day ahead forecast F_{t+1} is the average of the last k observations in a sequence Y consisting of T observations. Setting $k = 1$ is equivalent to forecasting the same value for tomorrow as is observed today.

- **Single Exponential Smoothing**
  It is related to the moving average method but recent observations are weighted more highly than earlier ones.

- **ARIMA Models**

  - **Seasonal Regression (RegAR)**
    RegAR assumes that seasonality is deterministic. The seasonal regression model was constructed by first de-trending and de-seasonalizing the time series using a least-squares regression of the dependent variable (admissions or occupancy) on time, and on indicator variables for day of the week and month.

- **Seasonal ARIMA (SARIMA)**
  SARIMA treats seasonality as stochastically varying, rather than deterministically varying.
- **Modelling Discharge Proportions (AdmDis)**
  This approach combines models of admissions and discharges. Admissions are modelled using RegAR model while discharges are modelled by performing a least-squares regression of the proportion of discharges on occupancy.
- **Modeling LOS (AdmLOS)**
  This approach uses the RegAR for admissions and a model of patient's LOS for discharges.

Considerations

Results highlight that none of the admission models could produce useful forecasts for any forecast horizon: admissions are largely unpredictable. RegAR and SARIMA could explain most of the variance of occupancy for up to two days ahead, and some of the variance for up to one week ahead.

Why do we quote this work? The important points that this study highlights is the unpredictability of the Emergency Department environment. So every system that could be realized have to be very dynamic.

Chapter 4

Model and objective function

From the study of the relevant literature, presented in the previous chapter, emerges the significant lack of a realistic model of an Emergency Department. All considered models provide indeed a partial view of the global problem. The model explained in this chapter is characterized, in our opinion, by a proper level of similarity to the reality.

Firstly, our purpose is to create a solid mathematical model that considers the most part of the variables that characterize the problem. Obviously it does not mean that all the variables should be taken into account: it would be impossible both for the ED and the algorithmic computational complexity. Anyway, ignoring heterogeneity of the resources or considering that a patient has to work only with one of those resources or limiting the possible tasks, are model simplification that are not acceptable in our opinion. Further more, the model has to be as close as possible to the reality to permit the simulation as well as possible of all the situations.

This chapter shows the mathematical model that supports our solution. It starts from the entities to arrive to the objective function.

4.1 The new model

The basic concepts of our proposal are derived from [20], the Master Thesis of Marco Ventilii, another student of the AI Laboratory of the *Università degli Studi di Padova* , and from [10].

The main limitation of the model proposed by Ventilii in [20] is the impossibility to add more degree of freedom: the model has to be taken as is, with the possibility to work on constraints or on the scheduling algorithms. For example, there is not a real representation of the human resources and, generally speaking, there is not a distinction between the resource types, so for example it is not possible to define a skill level for a doctor: he could be available for some tasks but not for other that could be too difficult for him, despite his classification as "Doctor".

The most important lack, which was also noticed in other models, is the partially or even totally absence of a resources' classification. With a clear and robust distinction

between different types it is possible to cover a great assortment of cases (i.e. a task might require some but not other types or more than one resource for every type).

Another important point is the simplicity to model a structure. In chapter 6 we will show that, with a rather simple XML input, it is possible to model the entire structure. Thus, it is possible to create a GUI that helps a possible user modelling the hospital.

A mathematical representation

The following model is created to satisfy the request of simplicity and generality. We will begin from the definitions of Day, Horizon, Resource, Task, Treatment and Patient to proceed with the variables, the constraints and the objective function.

Day The Day makes a one-to-one correspondence between the discrete time representation of the algorithm and the real time ones. Given a dimension QD of the time quantum in minutes, a Day is defined as follows.

$$\text{Day} = \{t_0, \ \dots \ , t_n\} \quad , \quad t_i \in \mathbb{N}, \ \forall i \in [0, n] \quad , \quad n = \frac{24 \cdot 60}{QD}$$

Horizon The Horizon represents the temporal space in which it is possible to share the inpatient flow. An Horizon is defined as a subset of a Day that contains a given (and constant) number of elements. So, given a dimension $HDIM$ and a starting time t_i with $0 \leq i \leq n$

$$\text{H}_i = \{t_i, t_{i+1}, \ \dots \ , t_{i+HDIM}\} \quad , \quad t_j \in Day \quad , \quad \forall j \in [i, i + HDIM]$$

Resource It is the representation of any type of the resources for the hospital. Conceptually it has got an unambiguous identifier, a type and a temporal availability.

$$\text{r} = (id, \ type, \ T_{avail})$$

The *id* permits to have more than one resource with the same type and availability in a set. The *type* represents the category of the resource (i.e. doctor, nurse, ...). The T_{avail} is a set of couples of start and end, that represents the periods in which the resource is available. For a hospital, it exists a set R that contains all resources.

$$T_{avail} = \{(t_{si}, t_{ei}) \mid t_{si}, t_{ei} \in Day, \ (t_{sj}, t_{ej}) \cap (t_{si}, t_{ei}) = \emptyset, \ \forall j \neq i\}$$

Task A Task corresponds to an action that covers one or more resources. As for the resources, the tasks are indexed by an unambiguous identifier to permit to have more than one task with same characteristics in the set. The mathematical representation is the following.

$$t = (id, \ R', \ Q_r, \ d)$$

$R' \subseteq R$ contains all the resources that are able to execute the task. Q_r links a requested quantity to every type of resources needed by the task.

$$Q_r = \{(t^y,\ q) \mid t^y \in R^{type},\ q \in \mathbb{N}\} \quad , \quad R^{type} = \{type(r) \mid r \in R'\}$$

d is a time interval that represent the duration of the task. In the hospital structure, all the tasks are contained in a tasks set T.

Treatment A Treatment is a sequence of possible tasks, with possible precedence constraints. This is not strictly necessary to manage the inpatient flow but it can be useful if appointments are inserted in the model. When an emergency patient arrives in the Emergency Department, it is impossible to know a priori what kind of operation would be done to him.

Patient It corresponds to a patient that arrives in the facility at any time. When a patient arrives, it is presumed that the Triage task is made automatically, so when the patient's presence is notified to the scheduler, there would be already a code and an assigned task that expresses the patient state.

$$p = (id,\ t,\ t_{arr}, t_{ddl},\ c,\ i_{prio}) \qquad \begin{aligned} t &\in T \\ t_{arr} &\in H_{arr} \\ t_{ddl} &\in Day \\ c &\in \{\text{Red, Yellow, Green, White}\} \\ i_{prior} &\in \mathbb{N} \end{aligned}$$

A patient arrives at a time t_{arr} that is the starting time for the horizon H_{arr} and he has to be treated with a task t. At his arrival, a code c with an internal priority i_{prior} is assigned to him. Depending on the code,it is assigned to him also a maximum time t_{ddl} before which the patient should start his care process. For a Red or Yellow patient this is an hard deadline, while for Green or White this is a tunable parameter, which can be adapted to the specific needs of a particular Hospital.

4.2 Search space and objective function

In what follows, we will show that from these entities it is possible to model the entire hospital's structure. Firstly, we will identify how to model the variables and the constraints which characterize the problem, then we will write the objective function.

4.2.1 The variables

In this section we will explain what are the variable. At time t_{arr} a patient p is added to the queue of waiting patients. He has just received a code c that means his urgency level and a task t that has to be accomplished as soon as possible (immediately if it is a red or yellow patient).

The solution for the patient p is represented by the assignment of a $t_{start} < t_{ddl}$ to his task while the complete solution is reached when the queue of waiting patients becomes empty and the largest number of constraints are satisfied.

This highlights that the plugs to be rightly scheduled are the couples $(p,\ t)$. In this way, given all assignments, the total time duration and the requested resources could be computed.

4.2.2 The constraints

There are several constraints that have to be considered. It is easy to give some example: a doctor can not execute a task if it is at home, a machine (like a MRI) can not work with two patients at the same time, the same thing occurs for any Operating Room, a patient can not be taken into account if it is not in the structure, and so on.

It is useful to make a distinction between the patients, the resources and the tasks constraints.

Patient constraints

The patient has to be in the structure and, specially for those that are classified Red or Yellow, they have to be treated immediately or, at least, as soon as possible. If the patient has to do a treatment (in other words a list of tasks) it is necessary to keep an order that is fixed and well-known, but also a no-overlap rule. In summary, the patients' constraints are the following:

- presence in the structure
- start immediately or at least as soon as possible
- order between consequently tasks
- no-overlap between the executions of two different tasks for the same patient.

Resource constraints

For the resources there are less constraints than for the patients. Because of the model's structure, resources can consider only a patient at a time, and they have to be present in the structure. So there are only a presence constraint and a no-overlap one. In summary, the resources' constraints are the following:

- presence in the structure
- no-overlap between patients in the same time.

Task constraints

Compared with the model proposed in [20], this is an innovation. Every task depends on the resources that are able to execute it. If the minimal quantity of resources is unavailable at a given time, the task can not start in that time. Obviously, if there is a surplus of resources, more than one patient can be served.

4.2.3 The Objective Function

An hospital has a clear objective: maximize the number of treated patients in a certain period of time. Obviously there are patients that require an immediate treatment while other can be delayed without any consequence. This is the case of a red patient that arrives later that another one: the red patient have to be treated before the other, regardless of arrival time.

The best choice in this cases is to work on the range between the arrival and the start of the treatment: if it is different from zero, a cost has to be paid depending on the dimension of the range. Obviously the cost for a delay is different according to the emergency level of the patient involved: the red patients have the higher cost while the white patients have the lower one.

Unfortunately this condition is not sufficient: every patient should have the start of his cure before a defined time. Using the only retard from the release it is not possible to consider this constraints. As for the other release condition, we consider the range between the deadline and the start of the treatment giving a cost to it. The cost is different for the urgency categories because, for example, the missing of a white patient deadline is less important then the missing of any other.

If it is possible, a solution should maintain precedence constraints: in the same category, if a patient arrives before than another one he should start before the other one (overall other constraints).

Observing the resource constraints, there are other important conditions: two patient can not be assigned to the same resource at the same time and every resource can not be used if it is not present in the structure. Both these cases give in output a scheduling that is impossible so their costs have to be large.

Summarizing, five conditions should be considered in the objective function. Considering the most important as the most expensive, we can write this list, where the first element is the most expensive while the last one the least expensive:

- the scheduling is feasible if all resources involved are present in the structure
- the scheduling is feasible if there are not conflicts between resources that are assigned to different patients contemporaneously
- the scheduling should have all deadline respected
- the scheduling should have all starts as soon as possible
- the scheduling should consider the precedence of patients that belong to the same emergency category

Each of these functions has the purpose of improving one or more aspects. Now it is necessary to find a way to balance these function weights to obtain a solid result. In our opinion, the trade-off is to maintain the high cost for the resource constraints while to give, to the patient constraints, some low values. This is a theoretical hypothesis that has to be verified by some tests (see chapter 7).

Giving a mathematical representation to the objective function, we can write the following statements. Starting from some constant values:

$$\Omega_{res} = \text{Resource conflict weight} \tag{4.1}$$
$$\omega_r = \text{Red patient weight} \tag{4.2}$$
$$\omega_y = \text{Yellow patient weight} \tag{4.3}$$
$$\omega_g = \text{Green patient weight} \tag{4.4}$$
$$\omega_w = \text{White patient weight} \tag{4.5}$$

where equation 4.7 is the weight for the use of a resource that is not present and from 4.8 to 4.11 are the weights for the missing of the deadline for every emergency category. The relationship between these values is expressed by equation 4.6.

$$\Omega_{res} << \omega_r < \omega_y < \omega_g < \omega_w \tag{4.6}$$

Starting from this preamble, it is possible to define all other parameters that are necessary to write the objective function.

$$\Omega'_{res} = \frac{\Omega_{res}}{x} \quad \text{is the weight for double assignment} \tag{4.7}$$
$$\omega'_r = \frac{\omega_r}{y} \quad \text{is the weight of the red patient retard on release} \tag{4.8}$$
$$\omega'_y = \frac{\omega_y}{y} \quad \text{is the weight of the yellow patient retard on release} \tag{4.9}$$
$$\omega'_g = \frac{\omega_g}{y} \quad \text{is the weight of the green patient retard on release} \tag{4.10}$$
$$\omega'_w = \frac{\omega_w}{y} \quad \text{is the weight of the white patient retard on release} \tag{4.11}$$

It is now possible to define the following objective functions for the various aspects.

$$\begin{aligned} OF_{res} = &\sum_{\forall p} \#\text{ of resources not present} \cdot \Omega_{res} \\ &+ \sum_{\forall p} \#\text{ of resource doubly assigned} \cdot \Omega'_{res} \end{aligned} \tag{4.12}$$

$$
\begin{aligned}
OF_{deadl} = & \sum_{\forall p \in \text{Red}} \max(0, start(p) - deadline(p)) \cdot \omega_r \\
& + \sum_{\forall p \in \text{Yellow}} \max(0, start(p) - deadline(p)) \cdot \omega_y \\
& + \sum_{\forall p \in \text{Green}} \max(0, start(p) - deadline(p)) \cdot \omega_g \\
& + \sum_{\forall p \in \text{White}} \max(0, start(p) - deadline(p)) \cdot \omega_w
\end{aligned} \tag{4.13}
$$

$$
\begin{aligned}
OF_{reles} = & \sum_{\forall p \in \text{Red}} \max(0, start(p) - deadline(p)) \cdot \omega'_r \\
& + \sum_{\forall p \in \text{Yellow}} \max(0, start(p) - deadline(p)) \cdot \omega'_y \\
& + \sum_{\forall p \in \text{Green}} \max(0, start(p) - deadline(p)) \cdot \omega'_g \\
& + \sum_{\forall p \in \text{White}} \max(0, start(p) - deadline(p)) \cdot \omega'_w
\end{aligned} \tag{4.14}
$$

Once found the trade-off changing the x and y values of the formulas 4.7, 4.8, 4.9, 4.10, 4.11 it is possible to define the final objective function.

$$
OF = \min \left(OF_{res} + OF_{deadl} + OF_{reles} \right) \tag{4.15}
$$

Chapter 5

The genetic algorithm

In this chapter the scheduling algorithm is presented. Before explaining our implementation, we must make a brief introduction to the genetic algorithms.

5.1 A brief introduction to the genetic algorithms

Genetic algorithms belong to the larger class of the evolutionary algorithms. They are search heuristic inspired by natural evolution. The solutions to optimization problems are generated using techniques such as inheritance, mutation, selection, and crossover.

5.1.1 Methodology

Initialization A genetic algorithm starts his evolutionary process from a population of randomly generates solutions. The population size depends on the nature of the problem and, traditionally, covers the entire range of possible solutions. The population is made by chromosomes (also called genotype of the genome) which encode candidate solutions (called individuals) to the optimization problem.

Selection Across generations, the solution is refined: the goodness of every individual in the population is evaluated calculating his fitness value. Individuals are stochastically selected through a fitness-based process, where the fitter solutions are more likely to be selected.

Reproduction The next step is to generate a second generation population of solutions from those selected through genetic operators. These are crossover and mutation.

- The **Crossover** combines two chromosomes, called parents, to create other two, called children. The process takes randomly some parts from a parent and some from the other to create a child and, with the remaining parts, it creates the other child.

- The **Mutation** changes some values of a child with some others that are randomly generated. This gives a major variability in the next generation than using the only crossover.

The selection of the parents and the mutation of the children are repeated until the new population of solutions has got an appropriate size. Generally the average fitness of the new generation will have increased by this process because only the best chromosomes are selected for generating the new population.

Termination The entire process is repeated until some terminating condition are satisfied. Common terminating condition are:

- a solution is found that satisfies minimum criteria
- fixed number of generations reached
- allocated budget reached
- the fitness remains constant for an high number of iteration
- manual inspection
- combinations of the above.

5.1.2 Criticism

As for other heuristic algorithms, there are some limitations. This scheduling problem, like others, is classified NP-Hard so the time needed to reach an optimal solution is exponential on the input size. In addition, evolutionary algorithms are stochastic processes so they give the certainty to obtain the optimal solution only if the calculus time goes to infinite ($t \to \infty$).

5.2 Genetic algorithms for dynamic scheduling

The starting point of our implementation is explained in [13] where, among other things, a genetic algorithms is implemented to solve a timetabling problem. This kind of problems is described by:

- a list of teachers
- a list of classes
- a list of weekly teaching hours for each class
- some internal conditions (like the curriculum of each class) and external conditions (like other bonds of teachers).

The construction of a timetable in which

- every teacher and every class must be present in a predefined number of hours

```
for(/*ALL NEW ARRIVALS*/)
  Patient p = nextPatient()
  Task t = p.getTask()
  for(i = 0; i < /*POPULATION SIZE*/; i++)
    Chromosome c = population[i]
    r = getRandomResources(p,t)
    if(/*PATIENT IS RED*/)
      c.add(p, t, r, now)
    else if(/*PATIENT IS YELLOW*/)
      c.add(p, t, r, /*NOW*/ : /*NOW + 1*/)
    else
      c.add(p, t, r, /*NOW*/ : /*NOW + HDIM*/)
    end if
  end for
end for

evalFitness(/*POPULATION*/, now)

while(/*NOT VERIFIED END TEST*/)
  for(i = 0; i < population.size() / 2; i++)
    /*TOURNAMENT SELECTION*/
    A = Random
    A1 = Random
    while(i < GROUP DIMENSION)
      B = Random
      B1 = Random
      if(population[A].getFitness() < population[B].getFitness() ||
          Random < (1 / GROUP DIMENSION))
        A = B;
      end if
      if(population[A1].getFitness() < population[B1].getFitness() ||
           Random < (1 / GROUP DIMENSION))
        A1 = B1;
      end if
    end while
    /*CROSSOVER*/
    crossover(population[A], population[A1], children1, children2)
    /*MUTATION*/
    children1.mutation(now)
    children2.mutation(now)
    evalFitness(children1, now)
    evalFitness(children2, now)
  end for
  /*ELITISM*/
  /*SORT POPULATION AND CHILDREN FOR FITNESS*/
  i = 0
  k = population.size() - 1
  while(/*POPULATION FITNESS*/ > /*CHILDREN FITNESS*/) {
    population[k--] = children[i++]
  }
end while

for(/*ALL ELEMENTS IN BEST CHROMOSOME*/)
  if(patient.start() = /*NOW*/)
    /*ADD PATIENT TO LOCKED LIST AND DELETE FROM ALL CHROMOSOME*/
  end if
end for
```

Figure 5.1: The pseudo-code for the genetic algorithm.

- there are not more than one teacher in the same class in the same hour
- no teacher is in two classes in the same hour
- no hours are uncovered (no teacher has been assigned to a class)

represents a feasible solution to the problem.

The greater distinction between a timetabling problem and our problem is the dynamism of the environment. As for the class, the hospital resources are known, but while the number of teachers is defined, it is not possible to know a priori the presence or absence of patients. This increases the uncertainty and the error rate. It is also necessary to lock the patients that have already started their tasks: for these ones the position can not be changed.

When the resources are sufficient in number to start all the pending patients, there are not problems but, when some patients are delayed, they will be considered in the following cycles. Our choice is to consider the previous work, so we try to insert the new patients without changing the already done. Nevertheless, if there are some constraint violations, the old solutions may vary consistently. Considering the mathematical model showed in section 4.1, a scheduling problem is described by:

- a time-variant list of patients with priorities and tasks
- a time-variant list of resources (we know how the resource presence varies)
- every couple patient-task requests the utilization of some resources.

The solution is feasible if some conditions are satisfied:

- the same resources can not be used at the same time in two different tasks
- to be used, the resource must be present in the structure.
- if the resources are free, a red patient must be scheduled before all others
- if the resources are free, a yellow patient must be scheduled before the green and white ones
- the deadline for a red patient prevails on all the other (it can not be missed)
- the deadline for a yellow patient prevails on the green and white ones
- all the deadline should be respected.

5.2.1 Chromosome

As shown in figure 5.2, every chromosome has a number of cells equal to the horizon size. These are the time instants in which all the patient-task couples could be placed. In each of the time instants can be placed more than one couple, but there should not be resources conflicts. This is an important point, since the fitness value increments a lot with a resource conflict.

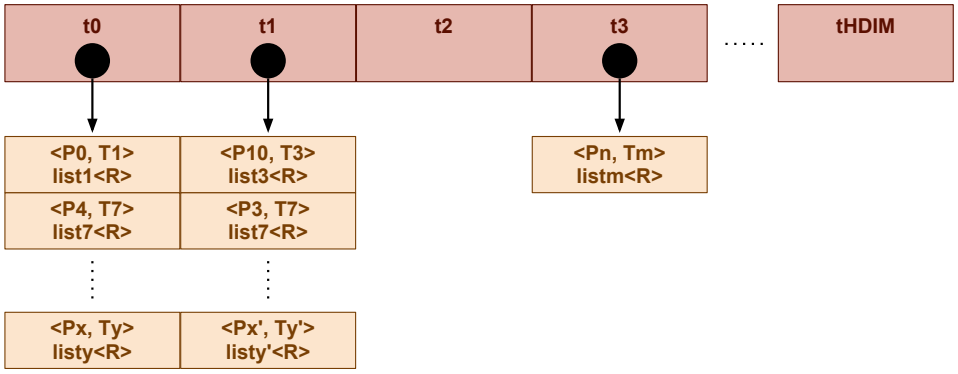


Figure 5.2: A representation of the structure of a chromosome

5.2.2 Population initialization

For every iteration, some new patients are added to the scheduling. These patients are inserted in all the chromosomes with random assignment of resources. The time instant in which they are inserted may vary from a fixed to a completely random choice. As shown in the below pseudo-code, a red patient is inserted at his arrival, a yellow may vary for one time instant while for green and white patients there are no limitation: they could be inserted in all the horizon.

```
for(/*ALL NEW ARRIVALS*/)
  Patient p = nextPatient()
  Task t = p.getTask()
  for(i = 0; i < /*POPULATION SIZE*/; i++)
    Chromosome c = population[i]
    r = getRandomResources(p,t)
    if(/*PATIENT IS RED*/)
      c.add(p, t, r, now)
    else if(/*PATIENT IS YELLOW*/)
      c.add(p, t, r, /*NOW*/ : /*NOW + 1*/)
    else
      c.add(p, t, r, /*NOW*/ : /*NOW + HDIM*/)
    end if
  end for
end for
```

5.2.3 The selection

We use a *tournament selection* method: from clusters of random selected chromosomes, the algorithm selects, with probability of $1/cl.dim$, a chromosome that has got a lower fitness than the best one in the cluster. This process is shown in the pseudo-code below.

```
    /*TOURNAMENT SELECTION*/
    A = Random
    A1 = Random
    while(i < GROUP DIMENSION)
      B = Random
      B1 = Random
      if(population[A].getFitness() < population[B].getFitness() ||
          Random < (1 / GROUP DIMENSION))
        A = B;
      end if
      if(population[A1].getFitness() < population[B1].getFitness() ||
           Random < (1 / GROUP DIMENSION))
        A1 = B1;
      end if
    end while
```

In addition to this selection, at the end of the creation of the children, there is a process called *elitism* that groups the best children with the best parents: this mix is the new generation. The combination of this strategies gives a good variability to chromosomes and permits a rapid convergence for a good solution.

5.2.4 The crossover

The crossover process mixes patients from two parents to create two children. The most important thing is that a patient can not be twice in a children (and not present in the other). To solve this problem, we order all the patient-task couples by patients id. Then, after having decide what number of elements of the first parent to swap with the second one, we combine randomly the elements. The process is shown in the following pseudo-code.

```
/*SORT ALL SCHEDULED PATIENT FOR PARENT1 BY ID*/
/*SORT ALL SCHEDULED PATIENT FOR PARENT2 BY ID*/

vector r = /*INIT TO RANDOM POSITION*/
cut = number of elements from parent1

while(i < cut)
  child1.add(parent1[r[i]])
  child2.add(parent2[r[i]])
  i = i + 1;
end while

i = cut
while(i < parent1.size())
  child1.add(parent2[r[i]])
  child2.add(parent1[r[i]])
  i = i + 1
end while
```

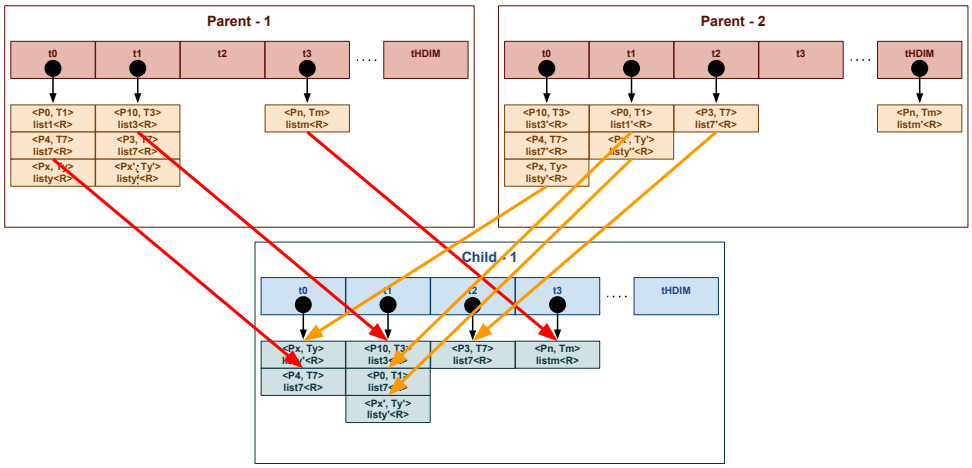


Figure 5.3: A representation of the crossover mechanism.

5.2.5 The mutation

There are two types of mutations: the first one for the time whereas the second one for the resource assignment. The time mutation chooses randomly a patient and assigns a new starting time which is also random. This process is shown in figure 5.4(a). Obviously a patient can not be placed before the time instant *now*.

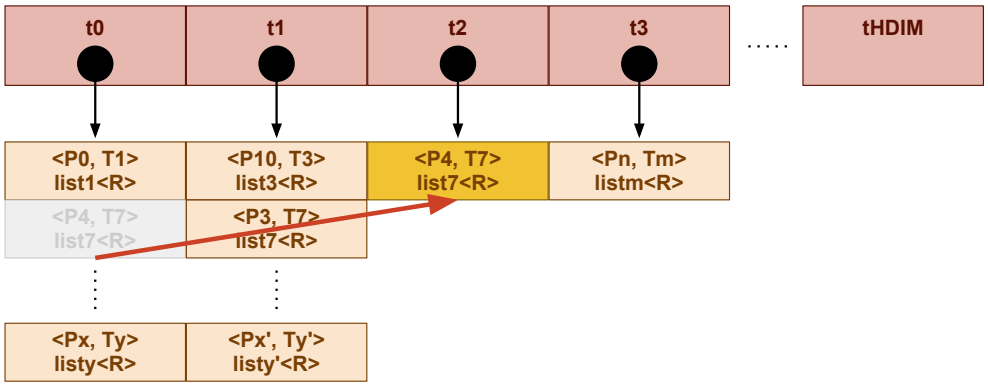
The mutation of the resource assignment is not completely random because it checks if the new resources are available. This permits to reach a better solution more rapidly than using a completely random process.

5.2.6 The locking of the started patients

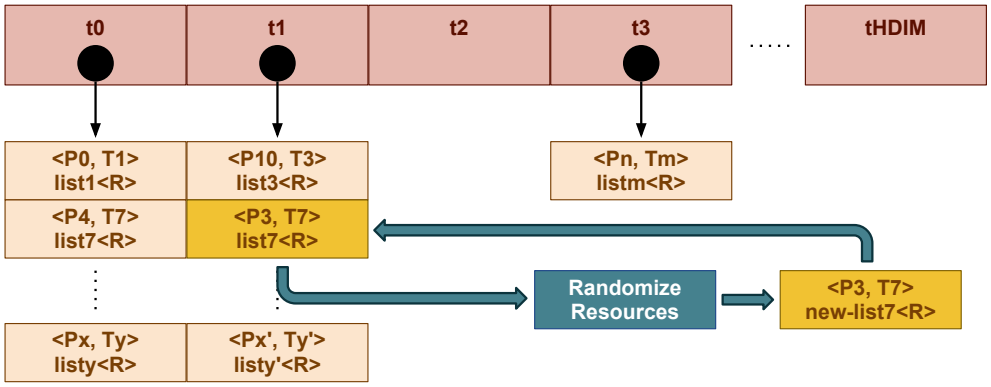
As mentioned above, when a patient starts his task, in the next cycles it is not possible to change his position. For this reason, after having reached the stopping condition, all the patients with the starting time at the now instant are deleted from all the chromosomes and inserted in a list of already scheduled patients. This last passage is necessary to keep track of the resource utilization in the next steps.

5.3 The fitness function

The fitness function is the implementation of the objective function showed in section 4.2.3. The following pseudo-code explains how it is obtained.



(a) The time mutation



(b) The resource mutation

Figure 5.4: A representation of the two possible mutation mechanisms

```
/*CONTAINS:
 - ALL RESOURCES ALREADY IN USE
 - ALL RESOURCES THAT ARE ASSIGNED TO PATIENTS */
utilizedResources[HDIM][]

utilizedResource.add(/*ALL RESOURCES ALREADY IN USE*/)

for(/*ALL TIME INSTANT IN CHROMOSOME*/)
  for(/*ALL PATIENTS NOT LOCKED IN THE TIME INSTANT*/)
    coeff = /*NUMBER OF PATIENTS OF LESS IMPORTANT CATEGORIES*/
    urgency = /*WEIGHT OF THE URGENCY*/

    /*DELAY ON DEADLINE*/
    fitness += coeff * max(0, start - deadline)^2 * urgency * 10;

    /*DELAY ON RELEASE*/
    fitness += (start - release) * urgency;

    utilizedResource.add(/*ALL ASSIGNED RESOURCES*/)
    for(/*ALL ASSIGNED RESOURCES*/)
      if(/*IS NOT PRESENT*/)
        fitness += conflict * 10;
      end if
    end for
  end for
end for

count = 0
for(/*ALL MOMENTS IN WHICH NO RESOURCES ARE USED*/)
  count++
end for
fitness += count * 100

for(/*ALL HORIZON*/)
  for(/*ALL UTILIZED RESOURCES*/)
    if(utilizedResource[time][i] /*IS ALREADY UTILIZED*/)
      fitness += conflict
    end if
  end for
end for
```

As it is shown, there are several parts combined and weighted: delay on deadline, delay on release, presence of the resources and conflicts between the assigned resources. In addition, there is a little weight if, at a given time, resources are totally unused.

To prevent priorities inversion between different urgencies level, a coefficient multiply the delay on the deadline. For example, let be:

$$\omega_r = 10 \quad \text{is the red weight} \tag{5.1}$$

$$\omega_y = 5 \quad \text{is the yellow weight} \tag{5.2}$$

$$\Delta t = 1 \quad \text{is the duration of all the pending tasks} \tag{5.3}$$

If at time t there are two yellow patients (that have not resource conflicts between them) and a red one (that conflicts with both yellow patients), the possible cases are the following:

- scheduler delays both yellow patients of Δt $\rightarrow$ the overall cost is $2 \cdot \Delta t \cdot \omega_y = 10$
- scheduler delays the red patients of Δt $\rightarrow$ the overall cost is $\Delta t \cdot \omega_r = 10$.

In this condition, a stochastic process has got the same probability to chose one or the other solution. Situations like this one are not possible in the reality so let be:

$$\#_r \quad \text{is the number of red patients} \tag{5.4}$$
$$\#_y \quad \text{is the number of yellow patients} \tag{5.5}$$
$$\#_g \quad \text{is the number of green patients} \tag{5.6}$$
$$\#_w \quad \text{is the number of white patients} \tag{5.7}$$

For each category the coefficient is obtained as follows:

$$c_r = \#_r + \#_y + \#_g + \#_w \tag{5.8}$$
$$c_y = \#_g + \#_w \tag{5.9}$$
$$c_g = \#_w \tag{5.10}$$
$$c_w = 1 \tag{5.11}$$

This system prevents cases like the one described above. If the weights of urgency categories are similar, a precondition is that the duration of more urgent tasks are at least as longer as the others. So those weights, for example, can be scaled by 10. Anyway those values are inserted by the user.

To keep a `FCFS` behaviour, we consider the square of the delay on the deadline. Giving another example, let be p_1 and p_2 two patients of the same urgency level and let be:

$$t_{arr,p_1} \quad \text{the release time for } p_1 \tag{5.12}$$
$$t_{arr,p_2} \quad \text{the release time for } p_2 \tag{5.13}$$

with $t_{arr,p_2} = t_{arr,p_1} + 3$. In a real case, p_1 will be treated earlier than p_2. But supposing that the possible solution are the following:

- p_1 starts at $t_{arr,p_1} + 4$ while p_2 at $t_{arr,p_2} + 1$: in this case they start contemporaneously
- p_1 starts at $t_{arr,p_1} + 5$ while p_2 at t_{arr,p_2}: in this case p_2 starts before p_1.

If we adopt a linear method to calculate the weights of these solutions, the result is the following:

$$w_1 = (4 + 1) \cdot \omega = 5\omega = (5 + 0) \cdot \omega = w_2 \tag{5.14}$$

that evinces the equality of the solution weights. Instead, if we use the squares, the solution are weighted differently:

$$w_1 = (4^2 + 1^2) \cdot \omega = 17\omega \neq 25\omega = (5^2 + 0^2) \cdot \omega \tag{5.15}$$

In order to simplify the input, some parameters are fixed: for example the coefficient that distinguishes between the `no-presence` and the `conflict` of a resource is setted to 10. At now, all these values are obtained experimentally, but in future developments they can be inserted by users.

Chapter 6

The architecture

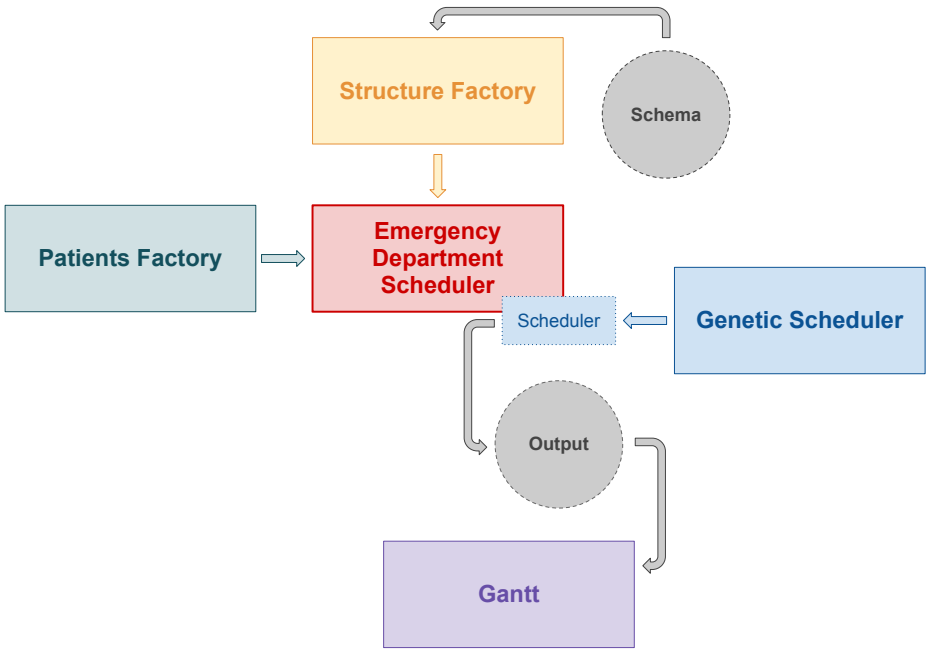


Figure 6.1: A simplified representation of the architecture of the software implemented to test the model.

In order to start from a good level of performances, we chose to use `C++` to write the code. As shown in figure 6.1 , the architecture is rather complicated. There is the node that contains the `main(int argc, char* argv[])` function in which all principal structures (like the patient queue or the list of all resources) are declared: it is called the *Emergency Department Scheduler*.

The purpose of the *Patients Factory* and the *Structure Factory* nodes are to generate, respectively, randomly patients of different codes accordingly to a Poisson distribution

and all possible resources and tasks defined in a XML file (*Schema*).

The *Scheduler* node is an interface that permit to use different types of scheduling algorithm: the one implemented is based on a Genetic Algorithm and is contained in *Genetic Scheduler* node.

This scheduler writes an *Output* file that is read from a Java program and plotted on a Gantt graph. This Java program is findable in the *Gantt* node.

6.1 The XML Schema

The XML schema represents, in a mnemonic way, the structure of the Emergency Department. It is part of the input that a user can insert in the algorithm. This file gives the possibility to define every resource and every task to model the structure.

6.1.1 TinyXML

To parse XML code, we use `TinyXML` [1]. It is a simple, small, minimal, C++ XML parser that can be easily integrated into every program.

It reads XML and creates C++ objects representing the XML document. The objects can be manipulated, changed, and saved again as XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<definition>
  ...
</definition>
```

In the fragment of code displayed above, it is possible to see the *header tag*. `TinyXML` requires it to generate the tree structure. As a matter of fact, `TinyXML` works recursively for the exploration of the file.

6.1.2 The Resources syntax

To define any resource it is sufficient to add a tag with the type and some other parameters, as displayed in the code fragment in figure 6.2 .

Theoretically, it is possible to add any type of resource but, until now, the tag definition is hard-coded so the possible resource types are four.

Doctor

As it is shown below, it is possible to define a doctor by specifying an `id` and an `availability`. Optionally, but in our opinion recommended, it is possible to define also a `name` and a `surname`. Every resource is indexed by its id, that is interpreted as a string, so it can contain every legal character. This is in order to give the maximum flexibility for the management of the structure in a real

[1] `http://www.grinninglizard.com/tinyxml/`

```
<!-- DEFINITION OF THE RESOURCES -->

<doctor id="A123-BF" avail="06.00-14.00;20.00-24.00">
  <name> DR.A </name>
  <surname> SRN.A </surname>
</doctor>
<doctor id="001023004" avail="06.00-14.00">
  <name> DR.B </name>
</doctor>

...

<nurse id="16" avail="06.00-14.00">
  <name> NU.A </name>
</nurse>

...

<machine id="30" avail="06.00-20.00">
  <name> TAC.A </name>
</machine>

...

<surgery id="46" avail="0.00-24.00">
  <name> OR.0 </name>
</surgery>

<!-- DEFINITION OF THE RESOURCES -->
```

Figure 6.2: An example of XML syntax for the declaration of the resources.

situation. Another important point is the availability: a doctor can not be present "24-7", so it is important to define the shift in which he will work. At the moment it exists a daily definition, that is made by a set of intervals: "`-`" divides the start from the end of an interval while the "`;`" divides different intervals.

```
<doctor id="A123-BF" avail="06.00-14.00;20.00-24.00">
  <name> DR.A </name>
  <surname> SRN.A </surname>
</doctor>
<doctor id="001023004" avail="06.00-14.00">
  <name> DR.B </name>
</doctor>
```

Nurse

The syntax is not different from the one listed for the doctor. There are the `id`, the `availability`, the `name` and the `surname` and each of them works as explained above.

```
<nurse id="16" avail="06.00-14.00">
  <name> NU.A </name>
</nurse>
```

Machine

In order not to specialize a resource, the same rules and the same syntax are used also for non-human resources. Obviously, the name can be thought, for example, as a brief description and the surname as a complete one.

```
<machine id="30" avail="06.00-20.00">
  <name> TAC.A </name>
</machine>
```

Surgery

In this case as well we can notice the same syntax also for the `availability` even though a room can not leave the structure. Reasonably, the availability for a room could be fixed to 24 hours, but this conflicts with the purpose of generality, so we leave to the user the decision.

```
<surgery id="46" avail="0.00-24.00">
  <name> OR.0 </name>
</surgery>
```

The standard way to declare resource types permits to instantiate whichever resource type present in an hospital structure. However, at the moment, it is necessary to put the new type name into the file *TagM.cpp*. This file contains the declaration of the class `TagM` that provides some structures and methods that permit to manage the XML syntax in a robust way. In the file it is possible to see the following initialization.

```
const vector<string> TagM::resourceType = boost::assign::list_of
    ("Doctor")
    ("Nurse")
    ("Machine")
    ("Surgery")
;
```

After the addition of a new row with the desired type name (without spaces), a new tag with that name will be correctly parsed (in other cases it will be ignored).

6.1.3 The Task syntax

In the matter of the declaration of the tasks, an example of syntax is shown in figure 6.3 . Compared to the declaration of a resource, it is slightly more difficult. But also the

```
<!-- DEFINITION OF THE POSSIBLE TASKS -->

<task id="01" duration="5" preemptible="no">
  <description> TASK.0 </description>
  <multi> 16,17,18,19,28,29,30,31,40,41 </multi>
  <minimal>
    <type t="nurse"> 1 </type>
  </minimal>
</task>

...

<task id="11" duration="20" preemptible="no">
  <description> TASK.11 </description>
  <multi> 01,03,05,... </multi>
  <minimal>
    <type t="doctor"> 1 </type>
    <type t="nurse"> 2 </type>
    <type t="surgery"> 1 </type>
  </minimal>
</task>

<!-- DEFINITION OF THE POSSIBLE TASKS -->
```

Figure 6.3: An example of XML syntax for the declaration of tasks.

task structure is more complicated: it contains all the resources that are able to execute the operation and a minimal number, for every typology, that is requested to complete the task.

```
<task id="11" duration="20" preemptible="no">
  <description> TASK.11 </description>
  <multi> 01,03,05,... </multi>
  <minimal>
    <type t="doctor"> 1 </type>
    <type t="nurse"> 2 </type>
    <type t="surgery"> 1 </type>
  </minimal>
</task>
```

The task header has got the initialization of the `id` and the `duration` and provide the possibility to specify if the task could be stopped during its execution. At the moment, the latter option is not considered and all the tasks can not be interrupted during the execution. This is a limitation, but in order to make a solid model this aspect is in the background and it affects principally the scheduling algorithm.

As for the resources, every task is indexed by its id, that is an alpha-numeric code. The duration is not referred to the time instant of the algorithm, but is expressed in

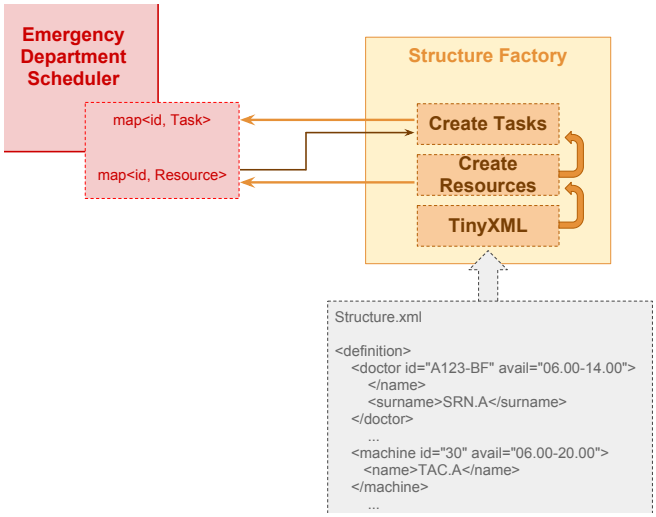


Figure 6.4: A simplified representation of the Structure Factory architecture

minutes. Obviously, if the duration is not a multiple of the time quantum dimension, it will be rounded up at the next time quantum.

It is possible to provide a description for the task but it is not necessary. Is obligatory, instead, the specification of the "multi set of resources" and the "minimal quantity". As it is visible, the `<multi></multi>` tag contains a list of ids belonging to some resources, while the `<minimal></minimal>` ones contains the typology and the quantity needed.

In order to make the algorithm work correctly, it is necessary to place that the declaration of the resources above the definition of the tasks. Several reasons justify this choice. First, the types of resources are not known a priori but only when all the resources are created. A second reason regards the algorithmic complexity: every task does not contain a copy of the requested resources, it contains the pointers to them and this permits to have a global entity and to use less memory.

6.2 The Structure Factory

The Structure Factory is the node that permits to read the XML accordingly to the rules showed in the section 6.1. To parse the file in input it uses TinyXML. As it is shown in the pseudo code 6.5, there is a cycle for all the nodes of the tree representation of the document. The declaration of the resources has thus to be placed over the tasks declaration.

The Structure Factory fills the structures containing resources and tasks that are instantiated in the Emergency Department Scheduler node and it is the only to create instances of this two object. This means that every other step that implies the use of this objects, as a matter of fact use pointers to them. This choice, as explained above, allows not to waste memory space and to have a "global resource" or a "global task" in which set parameters like, for example, the utilization.

```
readStructure()
  document = TinyXML.loadFile("path")
  resources = pointer(EDScheduler.resources)
  tasks = pointer(EDScheduler.tasks)


  for all nodes in document
    if current node is a Task
      property = readPropertyFromXML(current node)
      attribute = readAttributeFromXML(current node)
      tasks.add << makeTask(property, attribute)
    else if current node is a Resource
      property = readPropertyFromXML(current node)
      attribute = readAttributeFromXML(current node)
      resource.add << makeResource(property, attribute)
    end if
  end for
end readStructure
```

Figure 6.5: The pseudo code of the StructureFactory class

6.2.1 The Resource object

The resource object is quite simple: the mathematical model of the resource is equal to its implementation. So, starting from

$$\mathrm{r} = (id,\ type,\ T_{avail})$$

where

$$T_{avail} = \{(t_{si}, t_{ei}) \mid t_{si}, t_{ei} \in Day,\ (t_{sj}, t_{ej}) \cap (t_{si}, t_{ei}) = \emptyset,\ \forall j \neq i\}$$

we obtain an object that contains:

- `string id` that is an unambiguous identifier
- `string type` that represents the type of the resource object
- `vector<pair<int, int> > timeAvail` that is a list of couple of starts and ends of the periods in which the resource is available
- `string name` that is a mnemonic name (i.e. Mario for an human resource or CAT-A for a machine...)
- `string surname` that is the surname for a human resource or a description for other types of resources.

The only difference is the use of a vector instead of a set for the `timeAvail` alias T_{avail}: this, potentially, could break the rule of uniqueness, therefore it is granted by some

check in the code. Obviously there are some methods to set and get this parameters and, more interesting, there is another one to see if the resource is free. The `isAvail` function verifies if the given quantum is contained in one of the available intervals. To improve the performance, we will check only if the resource is available at the possible start time for a task: this simplification is reasonable because, for example, if a doctor starts to perform an operation, until the operation will be finished he will remain in the structure.

6.2.2 The Task object

As for the resource, the task is very similar to its model:

$$t = (id,\ R',\ Q_r,\ d)$$

Instead of sets, in the implementation we used maps. Conceptually, it is the same thing because the STL implementation of the map does not permit to have more than one copy of the same object. So the Task object contains:

- `string id` that is the unambiguous identifier
- `map<string, vector<Resource*> > m` that contains, for all the types, the resources
- `map<string, int> minimalQuantity` that contains, for all the types, the minimal quantity requested
- `int duration` that is the duration of the task
- `string description` that is a description of the task.

6.3 Patients Factory

The Patients Factory is a simple node that, accordingly to a Poisson distribution, generates a random number of patients in every time instant. As shown in figure 6.6 there are four instances of the class, one for each color. In this way the structure is very simple as it is shown in the listing 6.7. As for the Structure Factory, the Patient Factory fills a list of patients that is instantiated in the EDScheduler class. Now is the time to see the Patient object.

6.3.1 The patient object

As for the other objects, the patient object is very similar to its model. So, starting from its model

$$p = (id,\ t,\ t_{arr}, t_{ddl},\ c,\ i_{prio})$$

we make an object that contains:

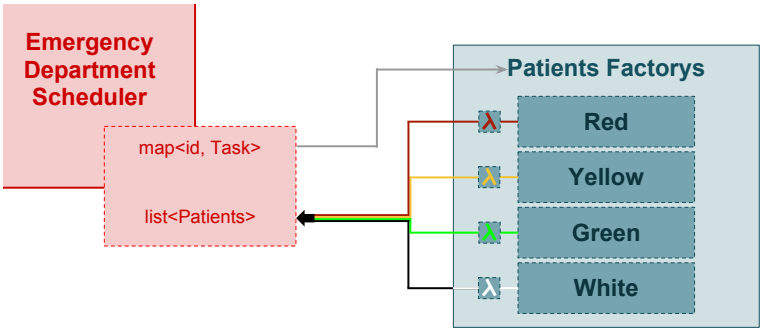


Figure 6.6: A simplified representation of the Patients Factory architecture. There are four instances of the class, one for each color.

```
generatePatients(now, code, lambda)
  patientsQueue = pointer(EDScheduler.queue)
  tasks = pointer(EDScheduler.tasks)

  random = PoissonDistribution
  num = random(lambda)

  for num times
    priority = uniformDistribution(0, 100)
    task = getRandomTask(tasks)
    patientsQueue << makePatient(name, surname, code, priority);
  end for
end generatePatients
```

Figure 6.7: The pseudo code of the PatientsFactory class

```
patientQueue
resources
tasks

main()
  StructureFactory.readStructure()
  scheduler = new Scheduler()
  now = Day[Time.now()]
  while(TRUE)
    PatientsFactory.generatePatients(now, RED, lambdaRED)
    PatientsFactory.generatePatients(now, YELLOW, lambdaYELLOW)
    PatientsFactory.generatePatients(now, GREEN, lambdaGREEN)
    PatientsFactory.generatePatients(now, WHITE, lambdaWHITE)

    scheduler.run(now);

    wait(remining quantum time)

    now = now + 1
  end while
end main
```

Figure 6.8: The pseudo code of the EDScheduler class

- `int id` that is an unambiguous identifier assigned directly by the Patients Factory
- `Task* task` that is the pointer to the assigned task
- `int start` that correspond to the arrival time
- `int deadline` that is the time in which the patient should start his care process
- `Code code` that is the priority code
- `int priority` that is the internal priority
- `string name` that is the name of the patient
- `string surname` that is the surname of the patient.

There is another variable, `bool locked`, that is necessary in the scheduling process. It will be explained in chapter 5.

6.3.2 Emergency Department Scheduler

It is the central node of the architecture, the one from which every algorithm is run and in which every structure is declared and maintained. Substantially, it runs the Structure Factory and this fills the lists of the tasks and the resources. Then it instantiates the Patients Factories and the Scheduler and, in an infinite loop, generates and schedules the patients.

6.3.3 Genetic Scheduler

The Genetic Scheduler node does the scheduling of the inpatients flow. As suggested by its name, it uses a Genetic Algorithm to execute its task. It implements an interface called Scheduler that provide some methods. Like the plotting, they are useful for all the scheduling algorithms. In the chapter 5 there are all the details of this algorithm.

6.3.4 Output

The output file contains all the informations to make the graphical output. As it is shown in figure 6.10 the output is composed by different parts. In the first line, the interval is written in the form `HH MM` for the starting time and `HH MM` for the ending time.

The lines from the second to the end are the real output. In order, it is visible the code, the patient's name, some numbers and the brief descriptions of the resources used by the task. The numbers in every patient line are organized in this manner: `HH MM` for the arrival of the patient, `HH MM` for his deadline, `HH MM` for the start of the task and `HH MM` for its end.

6.3.5 Gantt

The Gantt node is a simple Java application that, from the textual output, plots a graph. It uses the JFreeChart library [2]. An example of output of this application is shown in figure 6.9 and it is based on the textual output of the figure 6.10.

JFreeChart JFreeChart is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications. JFreeChart's extensive feature set includes:

- a consistent and well-documented API, supporting a wide range of chart types;
- a flexible design that is easy to extend, and targets both server-side and client-side applications;
- support for many output types, including Swing components, image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS and SVG);

JFreeChart is "open source" or, more specifically, free software. It is distributed under the terms of the GNU Lesser General Public Licence (LGPL), which permits use in proprietary applications.

[2] `http://www.jfree.org/jfreechart/`

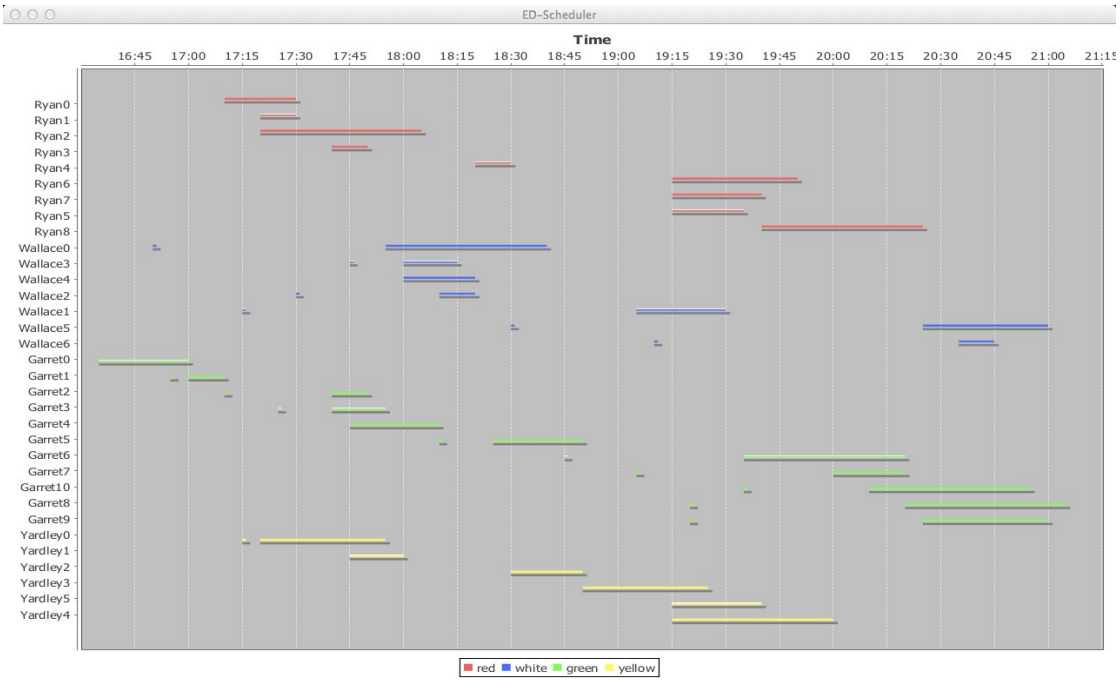


Figure 6.9: An example of the JFreeChart outcome using the textual output 6.10

```
16 35 21 40
green Garret0 16 35 17 40 16 35 17 0 RESOURCES: DR.I NU.V NU.X OR.1
green Garret1 16 55 18 0 17 0 17 10 RESOURCES: NU.T
red Ryan0 17 10 17 15 17 10 17 30 RESOURCES: DR.K NU.L OR.0
yellow Yardley0 17 15 17 25 17 20 17 55 RESOURCES: DR.N DR.B NU.O NU.A NU.I
    OR.6
red Ryan1 17 20 17 25 17 20 17 30 RESOURCES: NU.H
red Ryan2 17 20 17 25 17 20 18 5 RESOURCES: DR.F NU.S NU.1 OR.4
green Garret2 17 10 18 15 17 40 17 50 RESOURCES: NU.T
green Garret3 17 25 18 30 17 40 17 55 RESOURCES: NU.F NU.C
red Ryan3 17 40 17 45 17 40 17 50 RESOURCES: NU.2
green Garret4 17 45 18 50 17 45 18 10 RESOURCES: DR.I NU.X NU.H OR.3
yellow Yardley1 17 45 17 55 17 45 18 0 RESOURCES: NU.Q NU.R
white Wallace0 16 50 18 55 17 55 18 40 RESOURCES: DR.K NU.M NU.E OR.9
white Wallace3 17 45 19 50 18 0 18 15 RESOURCES: NU.C NU.D
white Wallace4 18 0 20 5 18 0 18 20 RESOURCES: DR.O NU.2 OR.1
white Wallace2 17 30 19 35 18 10 18 20 RESOURCES: NU.3
red Ryan4 18 20 18 25 18 20 18 30 RESOURCES: NU.3
green Garret5 18 10 19 15 18 25 18 50 RESOURCES: DR.M NU.J NU.R OR.2
yellow Yardley2 18 30 18 40 18 30 18 50 RESOURCES: DR.O NU.2 OR.1
yellow Yardley3 18 50 19 0 18 50 19 25 RESOURCES: DR.H DR.F NU.0 NU.A NU.G
    OR.6
white Wallace1 17 15 19 20 19 5 19 30 RESOURCES: DR.O NU.L NU.P OR.3
red Ryan6 19 15 19 20 19 15 19 50 RESOURCES: DR.J DR.B NU.E NU.O NU.Q OR.8
yellow Yardley5 19 15 19 25 19 15 19 40 RESOURCES: DR.E NU.D NU.N OR.1
red Ryan7 19 15 19 20 19 15 19 40 RESOURCES: DR.K NU.F NU.H OR.2
yellow Yardley4 19 15 19 25 19 15 20 0 RESOURCES: DR.G NU.S NU.T OR.7
red Ryan5 19 15 19 20 19 15 19 35 RESOURCES: DR.L NU.I OR.0
green Garret6 18 45 19 50 19 35 20 20 RESOURCES: DR.N NU.U NU.3 OR.9
green Garret7 19 5 20 10 20 0 20 20 RESOURCES: DR.K NU.V OR.2
green Garret10 19 35 20 40 20 10 20 55 RESOURCES: DR.C NU.L NU.X OR.8
green Garret8 19 20 20 25 20 20 21 5 RESOURCES: DR.G NU.V NU.S OR.7
green Garret9 19 20 20 25 20 25 21 0 RESOURCES: DR.H DR.F NU.A NU.G NU.M
    OR.4
white Wallace5 18 30 20 35 20 25 21 0 RESOURCES: DR.J DR.N NU.O NU.E NU.K
    OR.6
white Wallace6 19 10 21 15 20 35 20 45 RESOURCES: NU.C
red Ryan8 19 40 19 45 19 40 20 25 RESOURCES: DR.F NU.2 NU.A OR.6
```

Figure 6.10: An example of the output of the scheduler

Chapter 7

Experimental results

In this chapter, we will show the results of our implementation. Unfortunately, there are not standard benchmark to test the quality of the algorithm. This is not strange because in the literature there are not a standard approach to the solution of this problem. As already told, the various approaches strive to improve one or at most a pair of aspects, but it does not exist a solution to a more general problem.

Despite the absence of a benchmark, we try to realize some cases that could be comparable to a real situation. In the literature, if we look for a model of the emergency department, it is possible to find the trend of the patient flow or some data about the resource quantity (especially for the human factor). We are aware of the necessity to deep into a real emergency department to glean real data, but only with this purpose it is possible to write more than a thesis. Our decision is to bring some models, for example the ones illustrated in [15, 9, 2, 1], and to extract some data to confirm the model.

7.1 The construction of the model

Resources	**Time**					
	12 mid 7 a.m.	7 a.m. 11 a.m.	11 a.m. 2 p.m.	2 p.m. 3 p.m.	3 p.m. 11 p.m.	11 p.m. 12 mid
Nurses	4	4	6	6	7	4
Technicians	2	3	4	4	4	2
Doctors	1	1	1	2	2	2

Table 7.1: The model for the article [1].

Analysing some works we found some information about the human resources.

For example in [1] we found the table 7.1 that describe the distribution of the staff in a day. Another model divide by departments the staff and is visible in [2].

Unfortunately, no one of these models explain how to set the structural resources or the machines. But also for the task that are available we do not find any documentation.

In some works information are given about the typology of diagnosis and their probability but nothing was found about the needed exams.

	Morning Shift	Evening Shift	Night Shift
	Surgery		
Nurse	3	3	2
Doctor	1	1	1
	Neurology		
Nurse	1	2	1
Doctor	1	1	1
	Internal Medicine		
Nurse	2	2	2
Doctor	1	1	1
	Children's Medicine		
Nurse	0	1	0
Doctor	0	1	1
	Shared Resources		
Secretary	2	2	2
Lab Staff	1	1	1
X-Ray Staff	1	1	1

Table 7.2: An example of the model for the human resources that is cited in [2].

Another important point is the affluence of patients: we found in [15] some average values. From these values we depicted the figure 7.1 in which the blue line shows the number of patients per hour that come during week days while the red one shows the number for weekend days. With some simple calculus, it is possible to see that the average arrival for hour is less than 4 patients during week days and less than 5 during weekends.

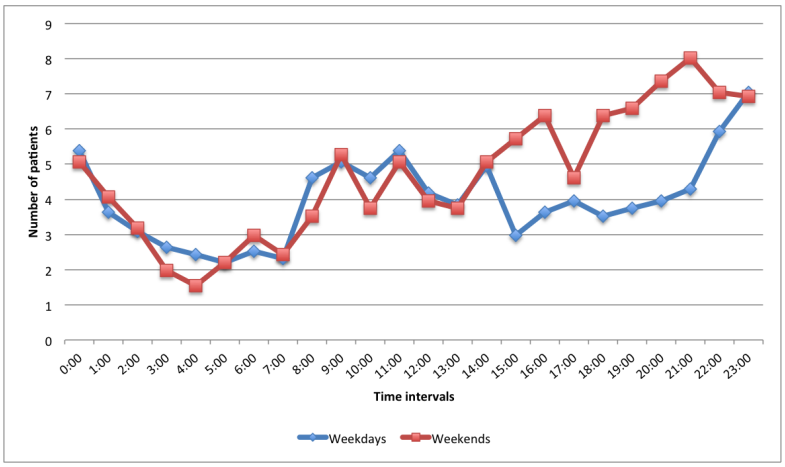


Figure 7.1: Some values about the arrival of patients during a day. The blue line shows values for week days while the red one for weekend days.

From this model, we design a table that provides the λ values to the "Poisson Randomiser" which generates the patient number. As shown in table 7.3, we started from the average number of patients, then we calculated the Λ for each time instant belonging to the time interval and finally we assigned a part of this value to each category. The rule we used is the following:

$$\Lambda = \frac{\lambda_{red}}{15} + \frac{\lambda_{yellow}}{10} + \frac{\lambda_{green}}{2} + \frac{\lambda_{white}}{3}$$

Time interval	**Average number**	Λ	λ_{red}	λ_{yellow}	λ_{green}	λ_{white}
00:00 02:00	9.02	0.375833333	0.025055556	0.037583333	0.187916667	0.125277778
02:00 04:00	5.72	0.238333333	0.015888889	0.023833333	0.119166667	0.079444444
04:00 06:00	4.62	0.1925	0.012833333	0.01925	0.09625	0.064166667
06:00 08:00	4.84	0.201666667	0.013444444	0.020166667	0.100833333	0.067222222
08:00 10:00	9.68	0.403333333	0.026888889	0.040333333	0.201666667	0.134444444
10:00 12:00	10.01	0.417083333	0.027805556	0.041708333	0.208541667	0.139027778
12:00 14:00	8.03	0.334583333	0.022305556	0.033458333	0.167291667	0.111527778
14:00 16:00	7.92	0.33	0.022	0.033	0.165	0.11
16:00 18:00	7.59	0.31625	0.021083333	0.031625	0.158125	0.105416667
18:00 20:00	7.26	0.3025	0.020166667	0.03025	0.15125	0.100833333
20:00 22:00	8.25	0.34375	0.022916667	0.034375	0.171875	0.114583333
22:00 24:00	12.98	0.540833333	0.036055556	0.054083333	0.270416667	0.180277778

Table 7.3: The table of the λs used to make our tests: it is made using the model in [2] for the average arrivals.

From the data showed above, we realised the model for our tests. This model have only "standard" resources (doctors, nurses, operating rooms, and ambulatories) to the purpose of diverting the least possible from the collected data: in table 7.4 are shown the quantities for each resource type.

While for the structure and the arrival of the patients we found some models, the task structure is all figment. For the major part of our tests the task structure is reported in table 7.5. As names suggest, there are one tasks for red and one for yellow patients while three for green or white ones. Tasks differ in duration and requirement of resources. For example, for the red task a doctor, two nurses and an operating room are necessary and the duration is of 30 minutes.

	Morning Shift	Evening Shift	Night Shift
Doctor	3	3	3
Nurce	6	6	6
Op. Room	3	3	3
Ambulatory	3	3	3

Table 7.4: The model of the resources.

Tasks	Doctor	Nurse	Op. Room	Ambulatory	Duration
Red	1	2	1		**30**
Yellow	1	1	1		**20**
Green/White	1			1	**15**
Green/White	1			1	**10**
Green/White	1	1		1	**10**

Table 7.5: This table show the task structure: the duration is in minutes while, over every typology of resource, are reported the needed quantities.

7.2 The comparison between a static and a dynamic approach

We tested the algorithm in periods of 24 hours and, to verify the goodness, we varied the average number of patient accordingly to the table 7.6: we started from the half arriving to the double of the average number.

Name	Meaning
(a)	All the simulations use a half of the average number of patients
(b)	All the simulations use a three-quarter of the average number of patients
(c)	All the simulations use the average number of patients
(d)	All the simulations use one and a half of the average number of patients
(e)	All the simulations use twice the average number of patients

Table 7.6: The various types of tests made to verify the model.

All tests are made with two versions of the algorithm, the first has a static approach while the second a dynamic approach. The following lines explain the difference between the two variants.

Static algorithm This solution, on each time instant, cleans all chromosomes: each pending patient receive a new assignation both of resources and starting time. This strategy increases the variability of the population, that is remade on every patient insertion. Nevertheless, all preceding work is not used to produce the new solution.

Dynamic Algorithm This version starts from the solution obtained in the past itera-

tions: if a patient comes, it tries to insert him in an already good list otherwise it tries to improve the existing list.

In the following sections, we will show the results of these tests and finally a comparison between the two methods. We chose to use the same configuration (depicted in 7.1) for the genetic algorithm, in order to evaluate the differences between the outputs.

$$\begin{aligned} \#_{chr} &= 1000 \\ \omega_{red} &= 1000 \\ \omega_{yellow} &= 100 \\ \omega_{green} &= 10 \\ \omega_{white} &= 1 \\ \omega_{resource} &= 1000000 \\ \text{Stop Condition} &= \begin{cases} 0 & \text{for the cost value} \\ 1000 & \#\text{ of iterations without fitness improvements} \\ 4'50'' & \text{maximum time} \end{cases} \end{aligned} \tag{7.1}$$

7.2.1 Static algorithm

All results wrote above are the average behaviour calculated on multiple executions of each configuration. As it is visible in table 7.7 the number of patients, obtained using the table 7.3, varies consistently from **(a)** to **(e)**.

Type	Red	Yellow	Green	White	Total
(a)	1.8	5	21.8	15	**43.6**
(b)	4.6	8.2	36.4	26	**75.2**
(c)	8.4	7.8	44	35	**95.2**
(d)	8	17.6	70.2	52.2	**148**
(e)	7.5	17	89.25	66.25	**180**

Table 7.7: Average number of patients generated for every type of test.

In table 7.8, the chosen parameters are visible: average processor time and cost and also the total cost calculated on the final list of scheduled patient. We chose to consider both the average and the total costs because only the second one is not sufficient with a restricted number of tests while the first gives an average behaviour that not fully characterize the final result.

The average processor time is kept low by the stopping condition that fix the number of iterations without fitness improvements to 1000: this is not a problem, in a real application the only two stopping conditions can be the maximum time (setted a bit smaller than the time quantum dimension) and the 0 value for the cost function. We used this condition to accelerate the output that, notwithstanding this simplification, takes in average 9 to 11, with some picks of 18, hours to be produced.

As it is visible in 7.8 for small numbers of patients the algorithm in this configuration reaches a solution. When the number is greater or equal to the average, the fitness values

Type	Average		Total
	Processor Time	Cost	Cost
(a)	0.055633892	0.02291674	4.4
(b)	0.317276162	0.2375	19.2
(c)	1.89819918	110.25828	7335.2
(d)	11.007026	13137.83863	1402834
(e)	21.6055	17509.72	22008777.75

Table 7.8: The results of the tests: the first two values are computed as an average of each time instant value, while the Total Cost is computed at the end of the execution on the list of scheduled patients.

increase noticeably and this means that a lot of constraints are violated.

7.2.2 Dynamic algorithm

Also in this case the results are calculated on an average behaviour. As in preceding tests, we used the table 7.3 to calculate the patient number: the outcome (visible in table 7.9) is a noticeable similarity both in number and distribution between categories.

Type	Red	Yellow	Green	White	Total
(a)	2	4.4	23.2	16.2	**45.8**
(b)	4.8	7.4	32.6	21.4	**66.2**
(c)	7	9.4	43	32.2	**91.6**
(d)	9.8	14	84.2	50.6	**158.6**
(e)	12.25	18.25	95.5	56	**182**

Table 7.9: Average number of patients generated for every type of test.

As visible in 7.10, the average processor time is very low. Also the total cost reaches values meaning the violation of some “soft constraints”, those are some retard for green or white patients or at most a little retard for a yellow patient.

Type	Average		Total
	Processor Time	Fitness	Fitness
(a)	0.024821954	0.01388888	2.2
(b)	0.18033223	0.20000002	28.6
(c)	0.150752814	0.249728888	46.6
(d)	1.5204164	0.7965278	96
(e)	2.83402	1.44704575	171

Table 7.10: The results of the tests: the average values are computed as an average of each time instant, while the Total Cost is computed at the end of the execution on the list of scheduled patients.

7.2.3 Comparison

As it is visible in the preceding sections, there are some differences between the solutions. The first noticeable difference is that, for small numbers of patients, both algorithms reach good solution while for great numbers only the dynamic one can produce an acceptable output. This is clearly visible in figure 7.4 where only the cost of the dynamic algorithm is smaller than a thousand. In fact, an average cost under a thousand implies that all emergencies has their constraints satisfied. Obviously, when resources can not satisfy all patients immediately some green or white patients are delayed and this is the reason of the cost greater than zero.

Another evidence is the start value for the cost: in the dynamic cases, the algorithm maintains an already good solution inserting new patients while in the static cases the solution is remade from scratch with the new arrivals. In the first case, there is a good starting fitness so the insertion of some patients can make worse this value for the violation of a restricted number of constraints and the value remains, in average, lower than starting from scratch.

Nevertheless for small number of patients, the great variety gave by the re-initialization of the population every time improves the performances. This is visible in 7.3 (a), in which the starting value is comparable, but the solution is improved more quickly than the dynamic case.

However, this behaviour of the static algorithm is findable only on very small patient numbers and this confirm the goodness of the dynamic solution that, in addition, requests very low processor time to reach good solutions. And this is another important thing: the genetic algorithm improves the solution more and more if it has time. In view of the situation, it is possible to decrease the quantum dimension (for example from 5 to 3 minutes) making better the similarity with the reality.

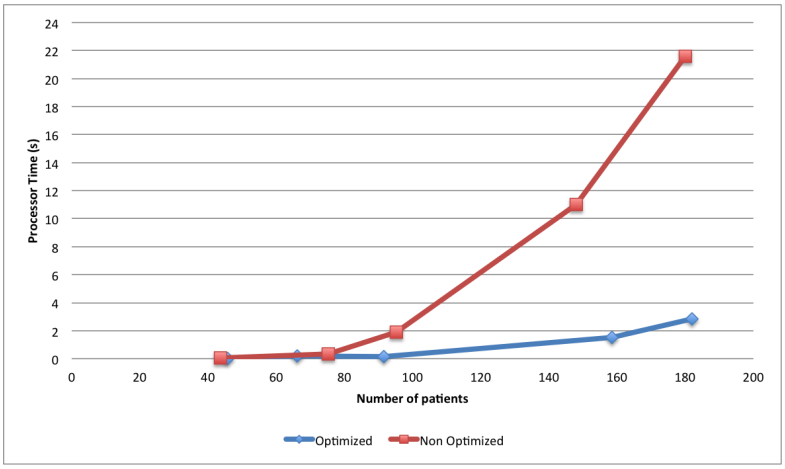
Summarizing, for very small number of patients, the behaviours of the dynamic and the static solutions are similar: as it is visible in figure 7.3 (a) and (b) they start from similar cost values and they reach good solutions. As the patient number increases, the static solution is not able to give outcomes within the same time of the dynamic one and this is visible in figure 7.3 (c) and more clearly in figure 7.4.

7.3 Tests on a particular pattern

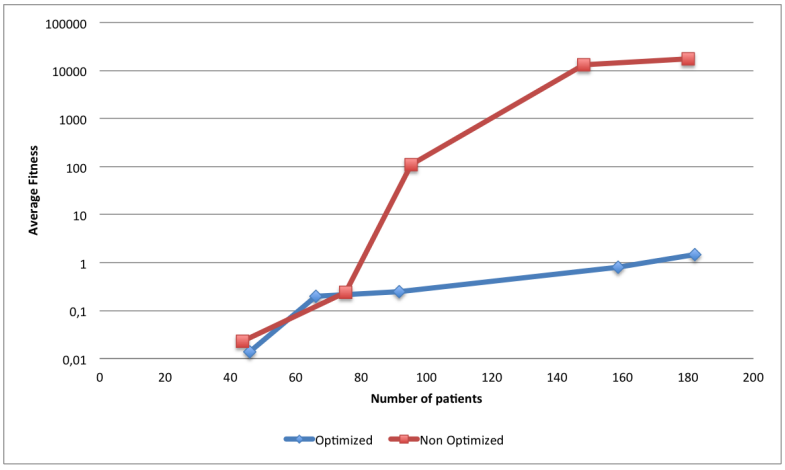
We simulate also a particular pattern changing the available resources: this pattern is named “car accident” because it tries to simulate an instantaneous pick of emergencies. This particular pattern is the following:

- a red and a yellow patients come simultaneously
- after five minutes, another yellow patient comes
- ten minutes later another red and yellow patients come.

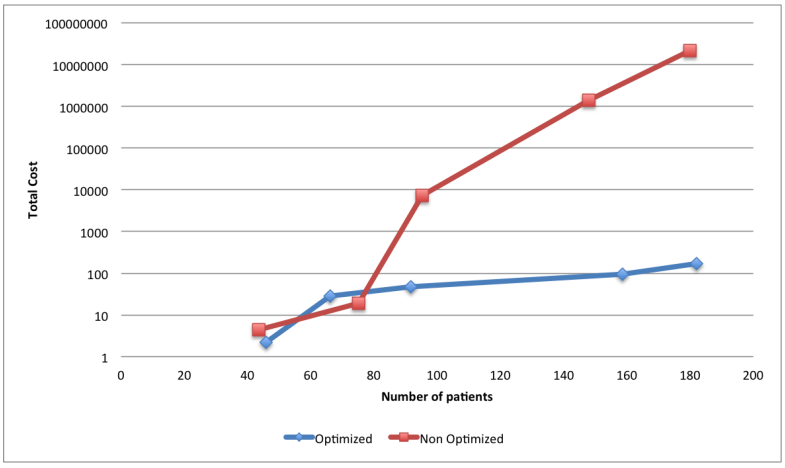
There is also a flow of green and white patients to complicate the problem. In these tests the average arrival is about 260% with a high number of emergencies.



(a) The average processor time for every iteration

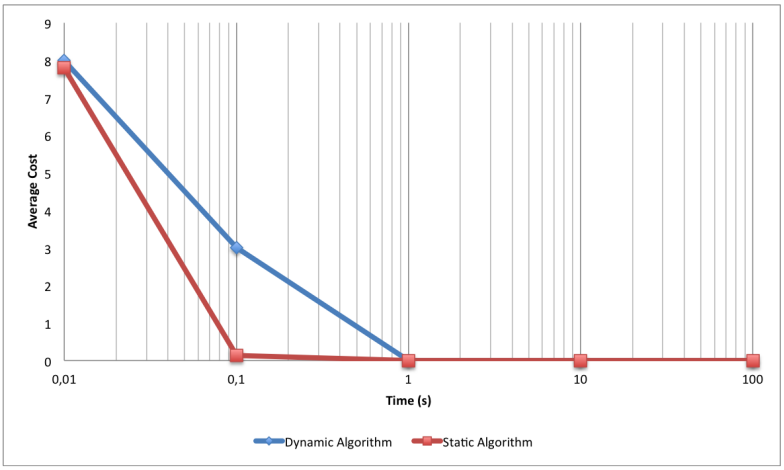


(b) The average cost for every iteration

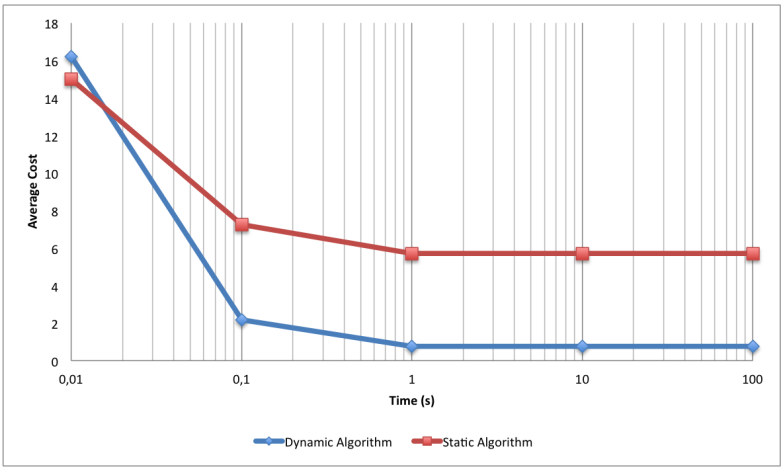


(c) The average of the total cost of the final solutions

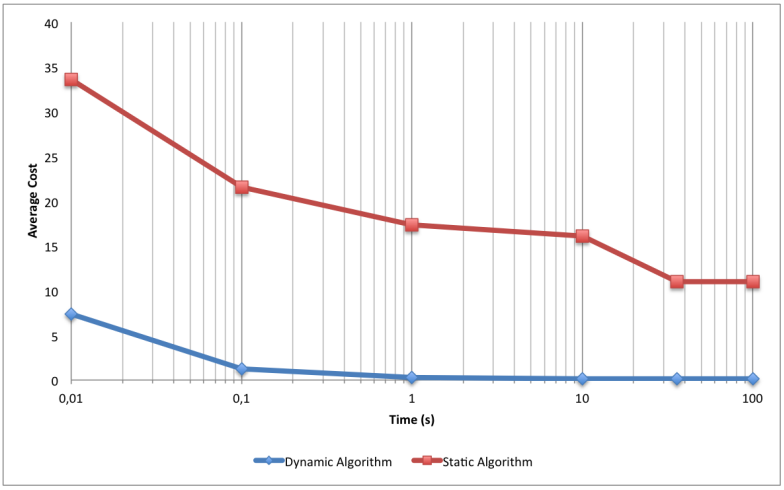
Figure 7.2: The comparison of some average behaviours of the two algorithms: these parameters consider all iterations, also the ones in which there are not patients and there are not computation.



(a) 50% of the average patient flow

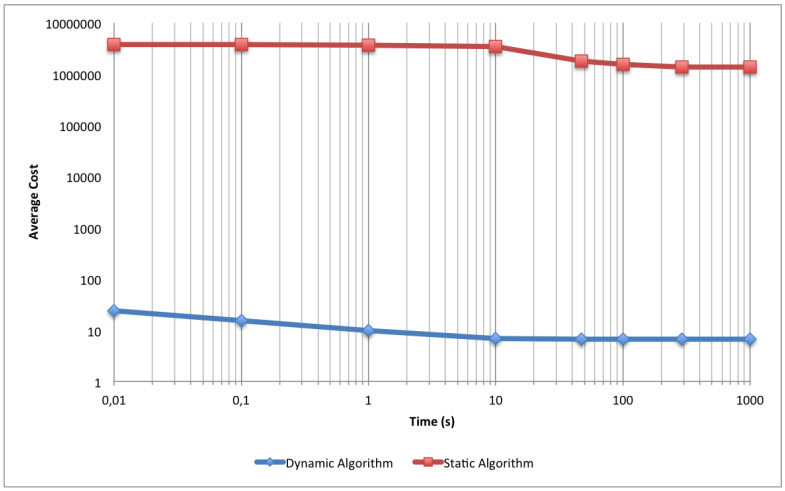


(b) 75% of the average patient flow

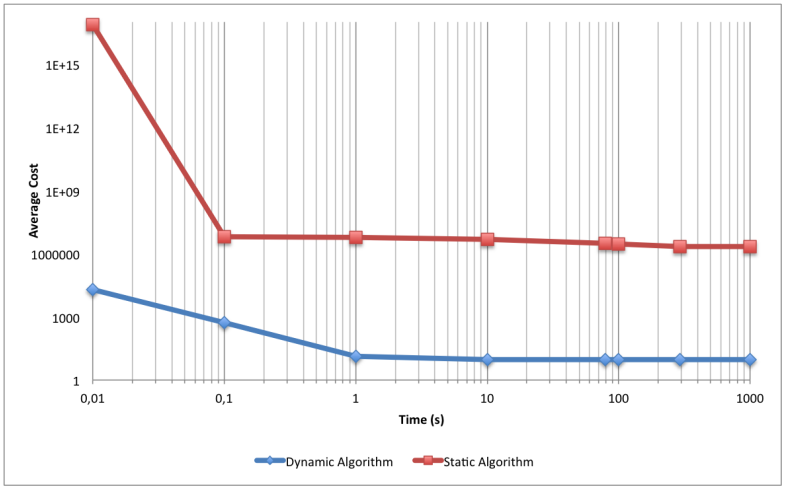


(c) 100% of the average patient flow

Figure 7.3: The difference between the costs of solution obtained with the static algorithm and the dynamic one. In these cases both algorithm reach acceptable solutions.



(a) 150% of the average patient flow



(b) 200% of the average patient flow

Figure 7.4: The difference between the costs of solution obtained with the static algorithm and the dynamic one. In these cases only the dynamic algorithm reaches solutions.

	Morning Shift	Evening Shift	Night Shift
Doctor	2	2	2
Nurce	4	4	4
Op. Room	2	2	2
Ambulatory	2	2	2

Table 7.11: Another model for the resource structure.

We use two variants of the resource structure: the first is visible in table 7.11 while the second is the one used for all other test and visible in table 7.4. The first model is characterized by a low number of resources and this fact improves the difficulty to obtain a good solution: surely some emergencies have to be delayed because the absence of available resources.

Tests on this particular pattern are not made to consider the parameters used in other tests. These are made in a limited time interval so the average processor time is not important and, in addition, the structure can not manage correctly all patient, so the fitness value increase consistently. Nevertheless, the outputs (that are visible in figure 7.5, 7.6) show how the resources are assigned to patients and, also, it shows that the results are quite similar to the best solution achievable. An important evidence is the utilization of the resources `Ambulatory A` and `Ambulatory B`: when the emergencies come, they are unused in spite of the necessity to allocate white and green patients. This fact highlight the correctness of the solution that prefers to care emergencies instead of normal patients.

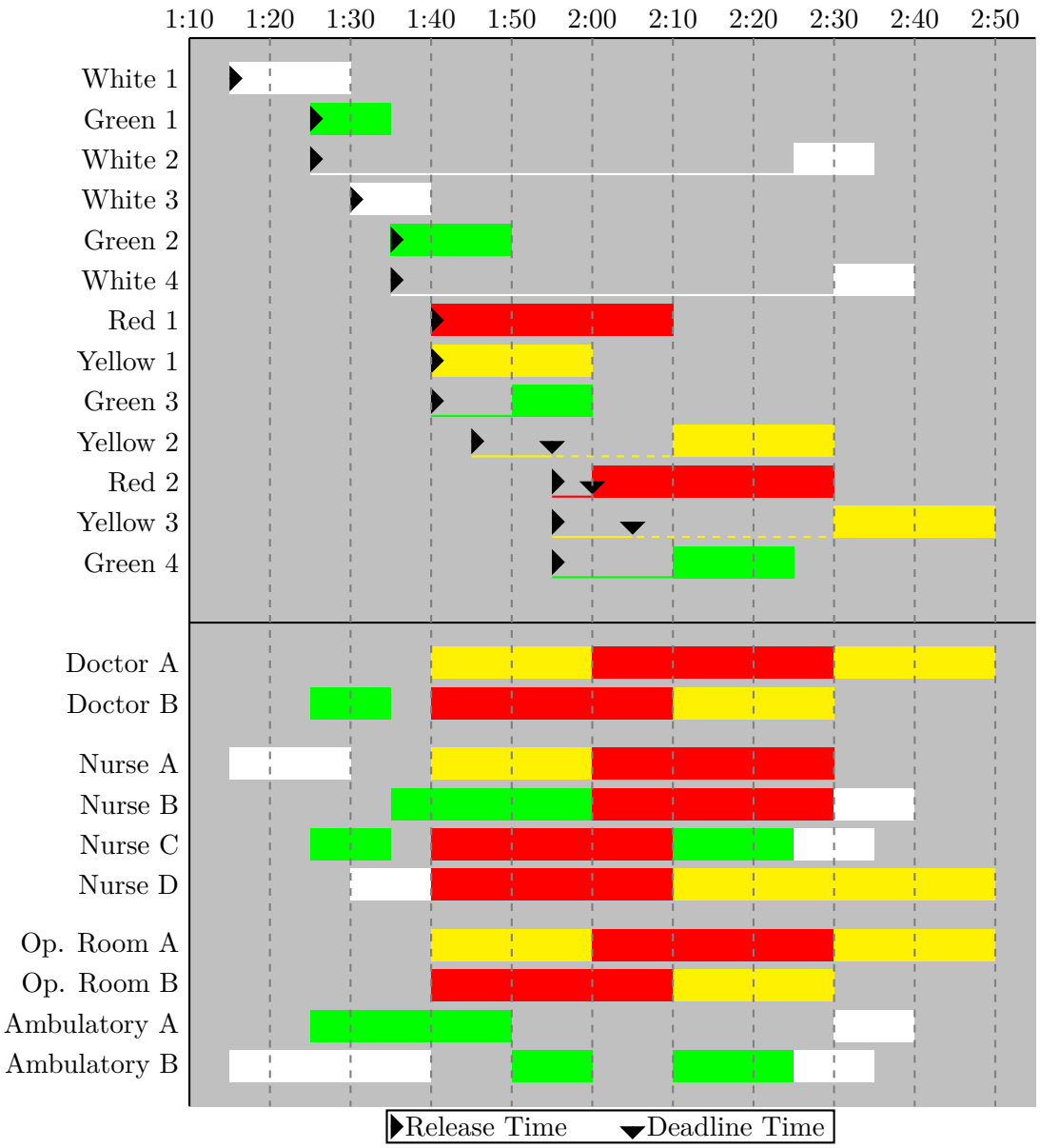


Figure 7.5: This figure shows the graphical output and the utilization of the resources in the “accident test” with few resources.

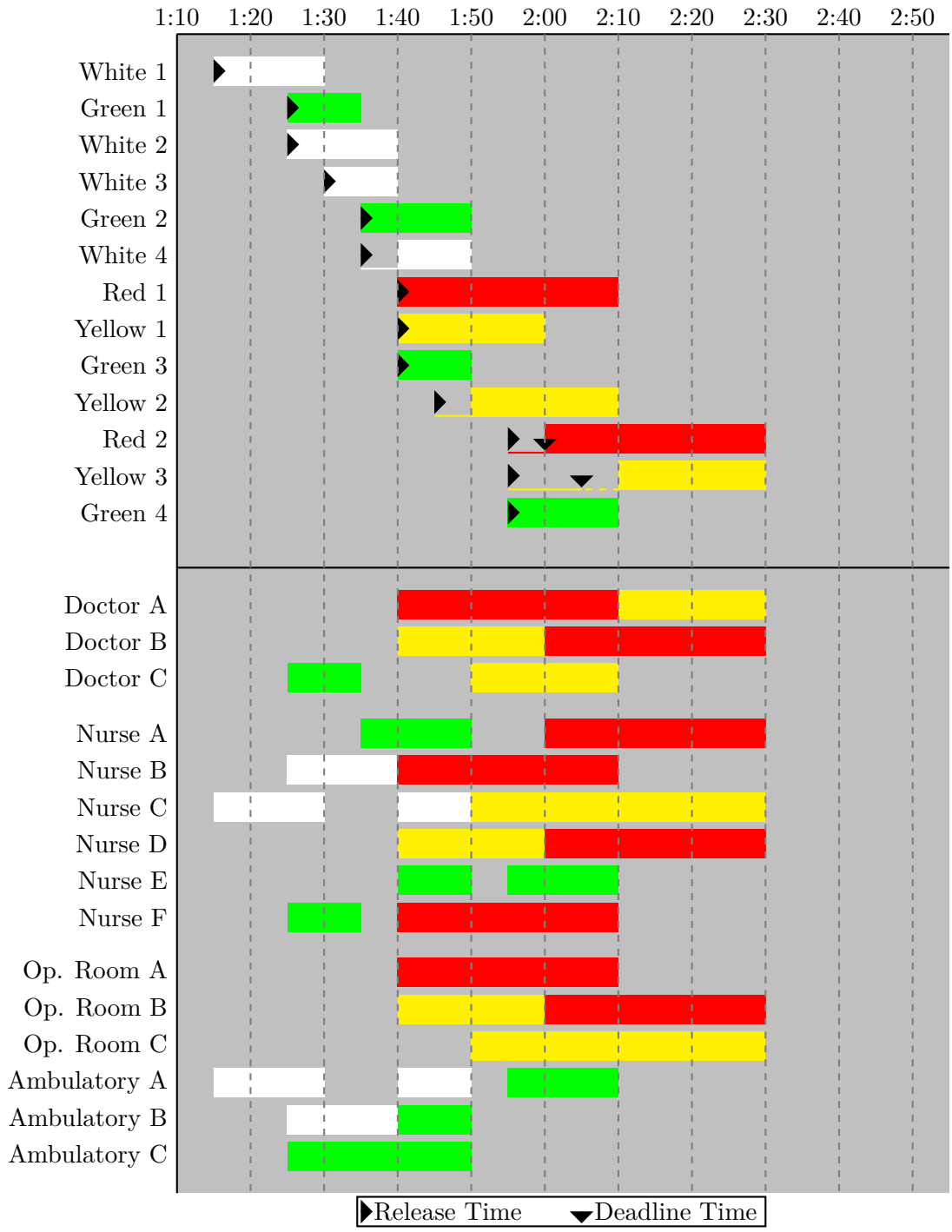


Figure 7.6: This figure shows the graphical output and the utilization of the resources in the "accident test" with an average number of resources.

Chapter 8

Conclusions and future directions

In this thesis we have presented a dynamic approach to patient scheduling at the Emergency Department, that differs from the classical scheduling problem because of the unpredictability of the flow of patients. This approach is based on a new model that is not present in the literature and that has the characteristic to be more similar to the reality than the others.

This needing of similarity to the reality is justified by the needing of design all minimal details of the hospital environment. Undervalued aspects of other models are key aspects for this work: one of this aspects is the lack of resources differentiation.

Further more, as it has been designed, the model can be largely customized: the possibility to design the hospital structure, with a lot of details, by a quite simple XML file, gives great advantages compared to other approach.

The tests highlighted that the use of the dynamic approach to the scheduling is largely better than the use of a static approach. In fact, the outcome of the dynamic algorithm is good both for small and high numbers of patients. Under the same conditions and with an high number of patients, the static algorithm requires more than ten times the processor time of the dynamic solution.

The dynamic algorithm can manage the patient flow, also in presence of picks: the algorithm respects as much as possible the urgencies requirement and the majority of the precedence constraints. In these cases the error ratio increases lightly due to the major number of reschedules. We have never found the violation of resources constraints that, conceptually, are the most problematic: a physician can not perform a treatment while he is at home or while he is doing another performance.

8.1 Problems

During the developing process we encountered numerous problems due both to implementation and to utilized components. Some of these problems gave us some random crashes or strange results.

Boost [1] **libraries** Despite the excellent performance of these libraries, we encountered

some difficulties due to some bugs in the algorithms of different versions. The standard version installed in *MacPorts* [2] repositories is the 1.47.0 while for Ubuntu is 1.46.1. When the code was compiled under Mac OS X, the algorithm finished its work without any problem, while the same configuration compiled under Ubuntu went in infinite loops. After several checks on our code we noticed that the problem involved the `boost::PoissonDistribution`: for small lambdas, there was a bug that caused infinite loops. This bug has been fixed from version 1.47.0 so, it is important to upgrade these libraries in Debian based operating system.

TinyXML problem We noticed a strange behaviour of this component: randomly, the tree of the XML structure is parsed in different ways. Before to close this cause off, several crashes happened during the execution. At now, this problem seems solved, however, in future developments on the XML syntax, it is suggested to check this factor.

The locking procedure When the stopping condition is satisfied, all patients belonging to best chromosome that have the starting time at the "now" instant are locked. At the beginning of the following time instant, this condition is spread to all chromosomes. Nevertheless, this procedure was made only from the new "now" instant to the dimension of the horizon, neglecting the lock of all patients belonging to those chromosomes that, as the best one, set their start time at the "now - 1" instant. The result of this error was the random rescheduling of patient that was already scheduled and locked. Once we found the cause, we easily fix this problem: for all assignment in which the start time is less that "now" instant, the start time is updated to "now".

The dynamic environment The great dynamism of the environment does not help the process of the refinement of the solutions made by the genetic algorithm. Fortunately, the number of patients added for each iteration is not so high to attack the evolutionary process. Nevertheless, we had to increase the number of mutations to permit a higher variability of solutions. But, in spite of this changing, in some cases, five minutes are not sufficient to reach a good solution: it happens that, in particular when the fitness reach high values, the algorithm does not move itself away from some local optimums.

The fitness function Obtaining a good trade-off between the weights of the various components in the fitness function was a long process. During the tests, several times we obtained wrong outputs due to unbalanced values. If the deadline missing weight was to high compared to the resource conflicts one, it could happen that the scheduler chose to assign a resource that was not present. Or, if the weight for the retard on release was to low, surely there was lot of precedence constraints violated.

[2] `http://www.macports.org/`

The testing phase During the testing phase, it could be important to have some reference models. Starting only from our knowledge, it was very difficult to design situations as close as possible to the reality in which making a series of tests. Improving this knowledge on the environment mechanisms and making standard parameters to permit standard comparison are two points necessary to be faced.

8.2 Future developments

In our opinion the model is very flexible. Nevertheless, during the testing phase some limits are came out. We have noticed that this limits are not due to errors of the conceptual model but, instead, to some possible improvements.

8.2.1 An advance notice of urgent patients

The advance notice of urgent patients is a point that was not known at the beginning of this work: just after talking to a hospital employee we have learned that the major part of urgencies are preceded by a call. This call is received by an health worker that is prepared to make some clear questions from which he can understand the urgency level. In this manner, if there are available resources these can be pre-emptively locked to treat the urgency while, if there are not, the algorithm can try to free as soon as possible the needed ones.

An example of the improvements that this solution can carry is visible in 8.1. As it is visible, the execution of the yellow patient delays the red one, but in a real case it may be better to take care of a red code with no delay than a yellow code.

8.2.2 The fitness function

Another possible development involves the fitness function: the current setting work properly but, sometimes, the precedence between patients belonging to the same urgency level are not maintained. This is probably due both to the stochastic process of evolution and to the weighting system. As explained in the previous chapters, the request of computational time is quite low, so increasing the complexity of the fitness calculus does not represent a problem.

8.2.3 A priority code internal to colors

Patients, in addition to the code, have got a priority that specify the urgency level inside the category. At now it is not considered by the algorithm but in future developments, this parameter may be useful to manage in a better way the priority inversions. In fact, the only four colors in some cases are not sufficient to give a clear order to the patients.

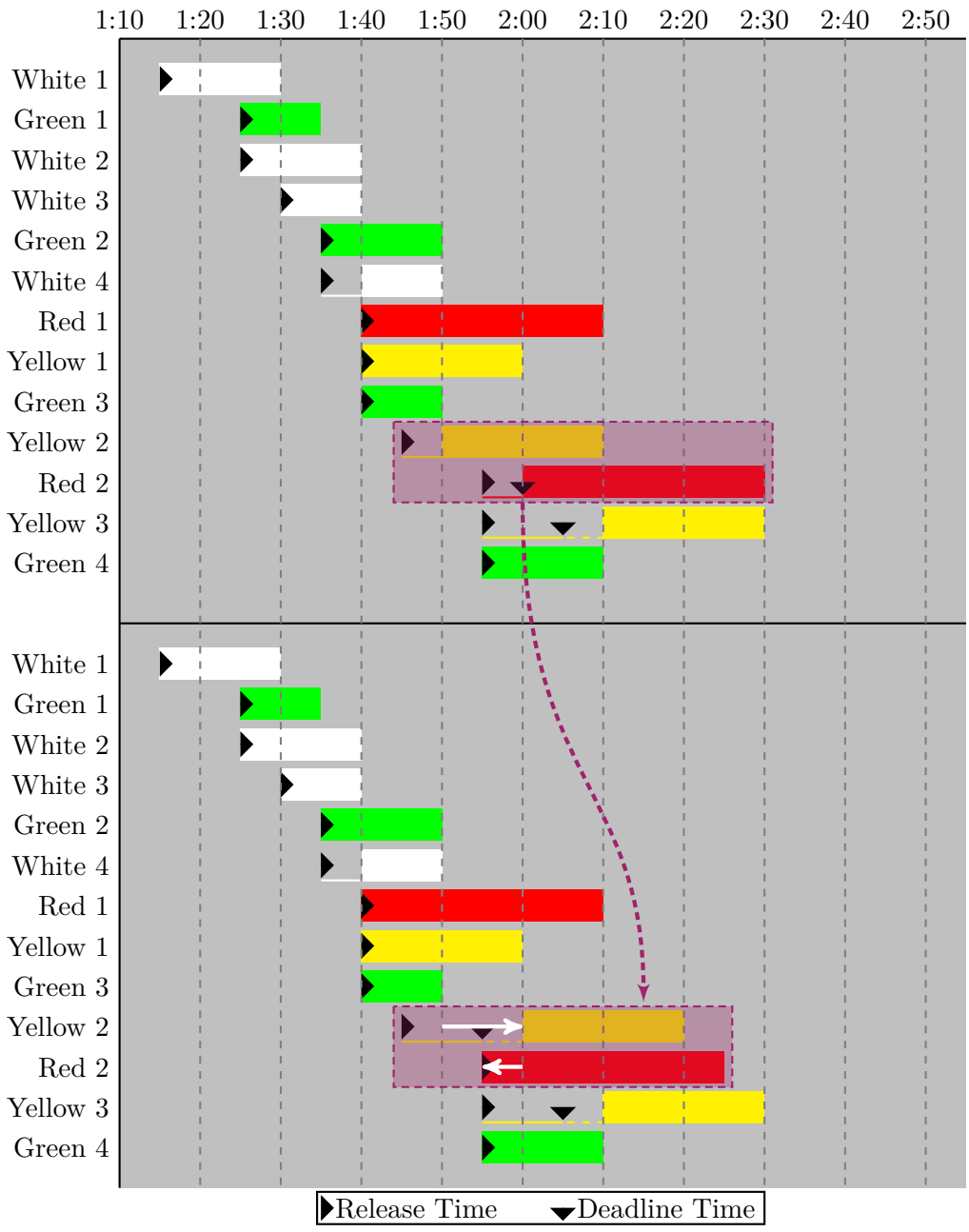


Figure 8.1: Having advance notices of emergencies coming, it is possible to prevent delay like the one showed.

8.2.4 Multi-threading

A possible future development is the transition form this mono task sequential algorithm to a multi-threading one. This is an interesting point and it can give some advantages: for example, if no one patient comes, the scheduler can continue doing its work without useless loss of time. But also the time management can be simplified inserting a thread that realizes the one-to-one correspondence between the quantum representation and the computer clock. This can simplify the management of the stopping condition. On the other hand, it is necessary to face the concurrency management but, fortunately, there are some libraries, like Boost [3], that helps on multi-thread synchronization. Boost Thread [4] library provides some methods, like locking procedure or semaphores, to manage shared resources. In our opinion, splitting the algorithm into the following thread could be very useful: PatientFactory, StructureFactory, Scheduler and TimeManager.

[3] http://www.boost.org/

[4] http://www.boost.org/doc/libs/1_47_0/doc/html/thread.html

Bibliography

[1] G. W.Evans, T. B. Gor, and E. Unger, "A simulation model for evaluating personnel schedules in a hospital emergency department," Winter Simulation Conference, 1996.

[2] T. Ruohonen, P. Neittaanmäki, and J. Teittinen, "Simulation model for improving the operation of the emergency department of special health care," Winter Simulation Conference, 2006.

[3] I. Berrada, J. A. Ferland, and P. Michelon, *Socio-Econ. Plann. Sci.s*, vol. 30. Pergamon, 1996. A Multi-objective Approach to Nurse Scheduling with both Hard and Soft Constraints.

[4] U. Aickelina and K. A. Dowslandbn, *Computers & Operations Research*, vol. 31. Elsevier, 2004. An indirect Genetic Algorithm for a nurse-scheduling problem.

[5] C.-C. Tsai and S. H. A. Li, "A two-stage modeling with genetic algorithms for the nurse scheduling problem," *Expert Syst. Appl.*, vol. 36, pp. 9506–9512, July 2009.

[6] H. Beaulieua, J. A. Ferlandb, B. Gendronb, and P. Michelonc, "A mathematical programming approach for scheduling physicians in the emergency room," *Health Care Management Science*, vol. 3, pp. 193–200, 2000.

[7] I. G. Stiell, G. A. Wells, B. J. Field, D. W. Spaite, V. J. De Maio, R. Ward, D. P. Munkley, M. B. Lyver, L. G. Luinstra, T. Campeau, J. Maloney, E. 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*, vol. 281, no. 13, pp. 1175–1181, 1999.

[8] G. Abraham, G. B. Byrnes, and C. A. Bain, "Short-term forecasting of emergency inpatient flow," *Ieee Transactions on Information Technology in Biomedicine*, vol. 13, 2009.

[9] P. Facchin, E. Rizzato, and G. Romanin-Jacur, "Emergency department generalized flexible simulation model," *IEEE*, 2010.

[10] M. Hannebauer and S. Muller, "Distributed constraint optimization for medical appointment scheduling," *AGENTS '01 Proceedings of the fifth international conference on Autonomous agents*, 2000.

[11] I. Vermeulen, S. Bohte, P. Bosman, S. Elkhuizen, P. Bakker, and J. L. Poutré, "Optimization of online patient scheduling with urgencies and preferences," *AIME 2009*, pp. 71–80, 2009.

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

[13] A. Colorni, M. Dorigo, and V. Maniezzo, "Metaheuristics for high school timetabling," *Computational Optimization and Applications*, vol. 9, pp. 275–298, 1998.

[14] "Pronto soccorso e sistema 118," 2007. http://www.mattoni.salute.gov.it/.

[15] J.-Y. Yeh and W.-S. Lin, "Using simulation technique and genetic algorithm to improve the quality care of a hospital emergency department," *Expert Systems with Applications*, vol. 32, pp. 1073–1083, 2007.

[16] A. Mazier, X. X. M. IEEE, and M. Sarazin, "Scheduling inpatient admission under high demand of emergency patients," *6th annual IEEE Conference on Automation Science and Engineering*, 2010.

[17] A. Daknou, H. Z. S. Hammadi, and H. Hubert, "A dynamic patient scheduling at the emergency department in hospitals," *Health Care Management (WHCM)*, 2010.

[18] Şafak Kırışa, N. Yüzügüllüb, N. Ergünc, and A. A. Çevik, *Knowledge-Based Systems*, vol. 23. Elsevier, 2010. A knowledge-based scheduling system for Emergency Departments.

[19] T. L. Saaty, *The Analytic Hierarchy Process*. New York: McGraw-Hill, 1980. A knowledge-based scheduling system for Emergency Departments.

[20] M. Ventilii, *Dynamic Constraint Satisfaction approach to Hospital Scheduling Optimization*. Padova: Università delgli Studi di Padova, 2010.