Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

251658240

PROJECTE FINAL DE CARRERA

Assignació de serveis amb estalvi d'energia i emissions de CO2 per a Data Centers basada en el Dynamic Bin Packing Problem

(Energy and Carbon emissions aware service allocation for Data Centers based on the Dynamic Bin Packing Problem)

Estudis: Enginyeria de Telecomunicació

Autor: Bernat Guillén Pegueroles

Directors: Xavier Hesselbach-Serra, Xavier Muñoz López

Supervisora: Sonja Klingert (Universität Mannheim)

Any:

2013

ABSTRACT

This project endeavours to develop strategies to allocate services in Data Centers in a collaborative relationship between the Data Centers and its users and Energy Providers. We study the implications of allowing service delays and short-term prevision of the energy mix when allocating services in a Data Center.

The project aims to propose heuristics and analyze them with the background framework of All4Green, a european FP7 project which aims to foster the relationship between all the partners of the ecosystem Energy Providers - Data Centers - End Users by the use of special contracts between each part that include flexibility and collaboration clauses.

To that end, the theoretical framework of problem tractability, problem complexity, online decision problems theory and metaheuristics will be studied and fully explained in order to find the best way to model and develop approximation algorithms that allocate services in a Data Center taking into account the Carbon Emissions Factor. Finding the model is an arduous task and it will be through various modifications, simplifications and attuning of parameters that we will find the most suitable one. We will see that the final model of the system is a modified version of a very famous decision problem called the Bin Packing Problem.

Later on, a family of heuristics will be proposed, studying thoroughly a couple of them and confirming the expectations: If we allow collaboration via the possibility of delaying and anticipating services we can obtain a huge benefit both economical and environmental when allocating services. Moreover we found a relationship between the level of collaboration a Data Center reaches (reflected in the advice and the possibility to delay) and the benefit.

We can conclude that we have succeeded in reaching the original objectives of the project and we provide useful strategies, guidelines and recommendations to be used in the frame of the All4Green project or other similar projects.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

DC	Data Center
EP	Energy Provider
ITC	Information Technology Consumer
A4G	All4Green
GreenSLA	Green Service Level Agreement
GreenSDA	Green Supply Demand Agreement
BPP	Bin Packing Problem
FF	First Fit
FFD	First Fit Decreasing

INTRODUCTION

The problem of energy efficiency and optimization, and CO_2 emissions in Data Centres (DC) has been worrying the scientific and technical community for some time, inspiring many solutions and approaches to it. Until now, the approaches try to minimize the energy consumption by either improving the hardware efficiency or the software efficiency, but none of them aims to optimize the overall efficiency in a hollistic way, taking into account the relations between the DC and the End Users (ITC) or DC and the Energy Provider (EP). All4Green's approach considers the whole environment as the key to reduce the energy consumption and the CO_2 emission.

One of the key aspects of All4Green is enforcing a collaboration between EP, DC and ITC. This collaboration consists in agreeing to delaying or pausing some services (DC-ITC) in order to fulfill some requests by the EP (EP-DC) and not abuse of expensive and polluting energy sources.

The goal of this project is to develop a model that is suitable for the study of the viability of different heuristics and strategies related to this collaboration that is pursued by the project All4Green, as well as proposing strategies and studying them to give a qualitative idea of which might be the best approach. The rationale for this thesis is to give strength to the belief that a world where all the members of a society collaborate is a better world. And that this world is possible even with the current social point of view, i. e.collaboration is profitable for everybody. We firmly believe that economical benefit is not at odds with being concerned in the environment, and by pursuing this thesis we want to prove that it is possible. We want to help other researchers and workers that share this belief with us to give one step further in the race for environmental sustainability. Not only this, but we want to do our bit in showing that mathematics can, and should be in service of improving all the society and that they can be used for more things than only generating money.

To that end, a rigorous mathematical analysis of the problem is needed, which will lead to the conclusion that it is indeed possible to ensure a profitable collaboration not only for the environment but for all the parties. A lot of theoretical background will be required to further study the problem. That theoretical background will be presented as well in this memoire. Many problems will arise when trying to produce a model of the whole EP-DC-ITC relationship and they will be solved more or less succeedingly. But in the end, it will be shown that there are ways to be environmentally and economically

efficient, and these ways will be presented in form of a family of heuristics.

STRUCTURE OF THE PROJECT

In order to achieve the goals posed before, the project will first consist of having a clear idea of how the All4Green project is working and what their objectives are. The first chapter will treat the ideas exposed by the All4Green project and try to find a way to synthetize them in order to use them as mathematical objects.

In the second chapter, a brief introduction to the theory of problem tractability and online decision problems, which will be key to the development and further study of the mathematical model of the problem, will be presented. Several works regarding the theoretical study of tractability, $NP - completeness$, online decision problems and Genetic Algorithms will be studied and selected summaries of these works will be shown in this chapter, hence allowing the reader (and the writer) to become familiar with the terminology of these fields.

In the third chapter, the practical work will begin by using whatever mathematical objects can be obtained from the first chapter and developing several mathematical models. These models will follow the chain of thoughts of the writer during the development of the final version, in order to clarify the reasons for each choice.

Finally, in the fourth chapter, a family of solutions to the model will be explained and two heuristics will be posed and thoroughly studied, reaching the conclusion that it is indeed possible to achieve both environmental and economical efficiency through collaboration between these three parts of the ecosystem. Some improvements, guidelines and recommendations will be given for future work related inside the project All4Green or other projects of similar expenditure.

The thesis is structured and written this way in the hope of convincing the reader that these strategies are not only useful, but necessary, if we want to live in an environmental-friendly world.

Part I

THEORETICAL INTRODUCTION

The first part of the project consists in an introduction to the theoretical framework in which the thesis will develop. On one hand the All4Green project is used as the technological background and some assumptions that will be made later come from this project. On the other hand, the study of the tractability of problems, online decision problems and a brief introduction to metaheuristics will be the mathematical basis needed to proceed later with the model. Furthermore a specifical problem called Bin Packing Problem will be explained in detail.

1

THE TECHNOLOGICAL FRAMEWORK: ALL4GREEN

INTRODUCTION

To be able to develop the necessary tools and solve the problem posed in this thesis, we will need a technological framework inside which we will work. As it would have no sense to solve the motion of a fluid without establishing the laws that rule it, we must be aware of the restrictions of this particular problem before trying to solve it.
The project All4Green (A4G) will take into account the relationships between the Energy Provider (EP), the Data Center (DC) as the customer of the EP, and an IT Customer (ITC) as the end user of the services offered by a Data Center. Let us read a short description of the goals of each partner:

- **IT Customer**/ITC. Each ITC represents demand for a DC. A4G focuses in those ITC that would appreciate savings in energy and energy cost. Its objective is to minimize the price paid to the DC for the computing services, and the total time in which its generated workload is executed by the DC.

- **Energy Provider**/EP. It represents the energy supply side for the DC. It needs to homogenize the energy load delivered to the DC, and to keep the amount of emissions controlled ("Green Quota").

- **Data Center**/DC. It is at the same time the provider of computing services for one or various ITC and the demand side of the EP. Its goal is to be able to perform all the computing requests within some given guarantees of performance, while keeping inside the restrictions in Energy Consumption given by the EP. In this report in particular, the objective is to minimize the Energy/Emission consumption caused by the computational workload. Thus, in this report in particular, we are not focusing in the economical aspects of the relationship between EP/DC and DC/ITC.

Each of the partners has different motivations and goals and they do not always overlap. The project A4G establishes a common goal for all of them – Energy and Carbon Emission efficiency – and aims to create new approaches to the whole ecosystem EP-DC-ITC.

1.1 GOALS OF THE PROJECT

Electricity consumed in Data Centers (DC), air cooling devices (AC) and uninterruptable power supply systems (UPS), is foreseen as a major contributor to the electricity consumed in the commercial sector in the near future, especially with the cloud computing trend still on the rise .

The project All4Green analyses the relationship between the IT Customer (ITC) and DC or DC federation in conjunction with the relationship between the different parts of the DC seeking a hollistic way of improving energetical efficiency and CO_2 emissions.

Instead of focusing on the energy optimization of single ICT elements, or subsets of the ICT elements making up a data centre, All4Green broadens the scope of energy savings to the full ecosystem in which data centres operate, fostering collaboration between all entities in this ecosystem with the common goal of saving energy and emissions through special contracts between them, following the work in [1, 2, 3].

With the proposed technology, energy savings generated in the data centre through the new relation with ICT users are magnified at the very source of the electricity transformation process through the coordinated collaboration of all the actors inside the ecosystem.

This collaboration is not only beneficial for the environment, but also economically sustainable, and therefore not limited to customers with a strong green/ecological conscience.

The main benefits of the envisioned ecosystems results from the interplay between data centres and energy providers. On the one hand, data centres, as important customers of energy providers, can have a great impact on the emergence and avoidance of energy usage peaks, and on the other hand, energy providers can reduce the impact of such peaks by using the optimal balance of energy sources based on their flexibility and CO2 emissions, including renewable energy sources like solar or wind, which have traditionally been difficult to be fully integrated into the electricity grid due to their long-term unpredictability.

The european FP7 All4Green project addresses this problem by coordinating energy supply and demand by encouraging extensive collaboration between energy providers and data centres as major energy consumers.

1.2 STRUCTURE OF THE PROJECT

The project is divided in 8 Work Packages that are run by different people and are coordinated by one of them (WP1). WP2 is the one in charge of making the baseline and target scenarios for the technical aspects of the project and receive a feedback from the final simula-

tion and implementation. WP3 is the technical WP in charge of the relationship between DC and EU, WP4 focuses on the EP-DC relation and WP5 includes the DC federation into the ecosystem (DCs can federate and reallocate workload between them). WP6 provides a simulation environment and is where the first tests will be made, and WP7 is the final real-world implementation. Finally, WP8 dedicates to dissemination of the work done in All4Green and tries to reach to the maximum number of potential users.

1.3 DC MODEL

A model of the DC behaviour and topology is found in the first deliverable of WP3. However, I will introduce shortly the data model and focus in the physical model which will be the object of our main attention during the course of this report.

- *Multi-tier model*: Servers are deployed in a hierarchical manner. The frontend consists in the web servers, that are connected to the application servers (that are used in case a specialized service is needed). The application servers are connected to storage, database and backup servers.
- *Server Cluster Model*: Often used in grid environments. There is no (or little) hierarchy: The requests are concurrently served.

The interconnection network consists (not necessarily) in three tiers: The core layer, the aggregation layer and the access layer. UPS and cooling systems are also part of the physical model of a DC.
Servers, Network Elements, Storage (divided in Direct-attached Storage, Network-attached Storage and Storage Area Network), UPS and the AC are the sources of power consumption in the DC.
The data model of the DC can also be found in WP3. Fig. 1 illustrates the relations between DC, EP, ITC, and the Federation. It also shows the relation between the physical devices and the software and IT Services (VM, software, server entities). Finding a proper model is very important and it will be necessary to do accurate predictions and efficient algorithms to be energy-aware efficient.

1.4 AGENTS, CONNECTORS, GREENSLA, GREENSDA

The whole All4Green project is sustained by three pillars: Energy Saving, Flexibility and Collaboration. Amongst other tools, the most important ones are Agents, Connectors, and the contracts between EP-DC and DC-ITC (GreenSLAs and GreenSDAs).

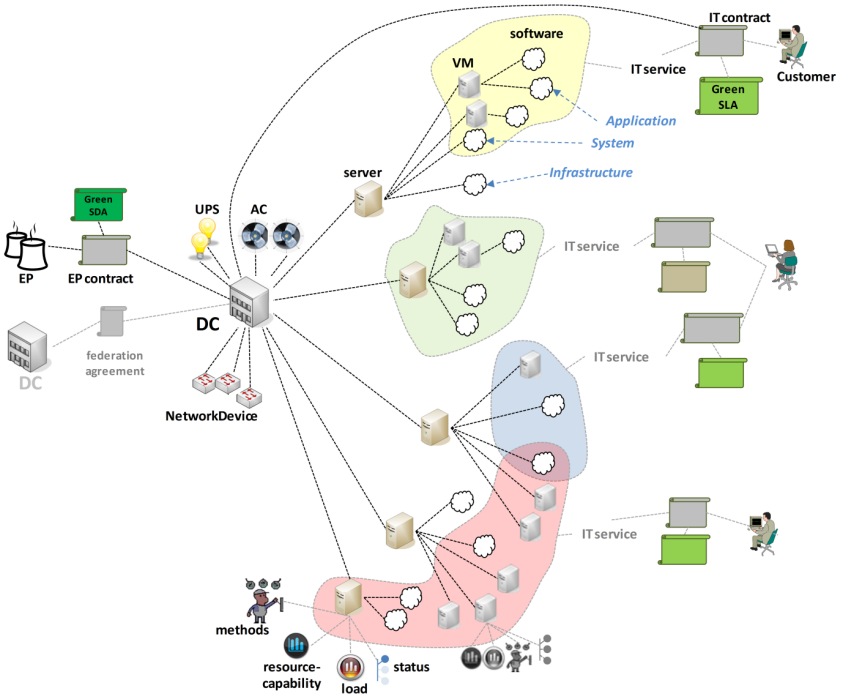

Figure 1: DC Data model seen in WP3

1.4.1 *Agents and Connectors*

Agents are autonomous software that monitor the state of each of the members of the ecosystem (DCs, EPs, ICTs) and they also are in charge of the communication between each of them (requests, accepting or rejecting offers, etc.). The connectors (DC or EP) are in charge of connecting the high-level decisions made by the agents to a low-level technology-dependent decision, made in two steps, the first "'translating"' the high-level decision to a low-level technology-non-dependent decision, and then again to a decision specific to the DC/EP control system.
Communicating and monitoring in an efficient way, and finding an optimal way of connecting high-level decisions to low-level decisions might probably be the key to an efficient way of reducing the energy consumption and the CO_2 emission.

1.4.2 *GreenSLA,GreenSDA*

The contracts made between DC-ITC are different from the traditional SLAs (Service Level Agreement). A traditional SLA states a QoS for some defined services. The QoS includes performance parameters, availability parameters, other parameters and, of course, the pricing. All of these are static and don't depend on anything, and work as can be seen in Fig. 2.

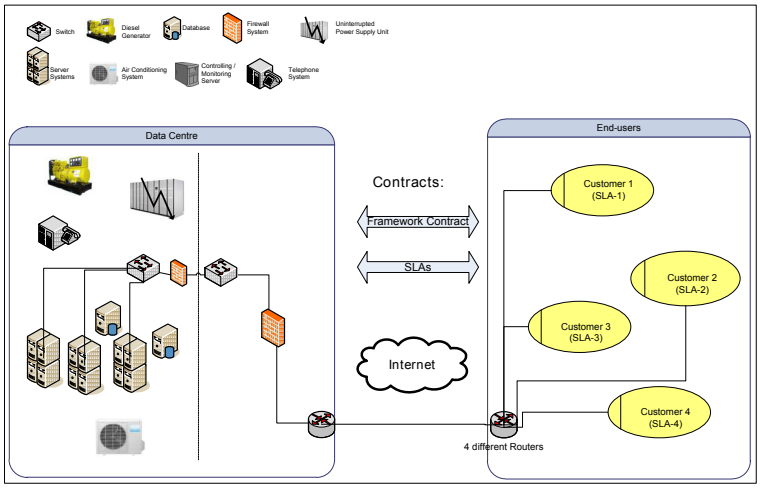

Figure 2: A standard DC model

A GreenSLA includes flexibility into this equation; the parameters and the pricing will depend on the context and will include some reduction of the general performance, in exchange for a reward in the pricing. The GreenSLA inside the All4Green project also fosters collaboration, and includes special rewards for the ITC that collaborates with the DC. The GreenSLA guarantees certain QoS but in a context-depending way.
Similarly, there is a GreenSDA between the DC and the EP, which also fosters collaboration towards a green ecosystem. Both GreenSLA and GreenSDA also define some GreenKPIs that will be monitored by the agents and will be indicators of the ecological efficiency of the situation. This KPIs will be used by the system to decide the actions to be taken by the DC.
The GreenSLA and GreenSDA have to be designed to ensure fairness to all the partners and should help to make All4Green attractive to as much possible stakeholders as possible, while providing a way to reduce CO_2 emissions and energy consumption. Therefore, they are one of the key elements of All4Green. It is very important to design them well, but from a technical point of view, their most important aspects are the parameters established by them (GreenKPIs and the Flexibility terms) because they will be used by the Agents and the Connectors. A summarizing image of the whole All4Green environment can be seen in Fig. 3. The part of the image in grey includes the federation of DCs which will not affect this thesis.

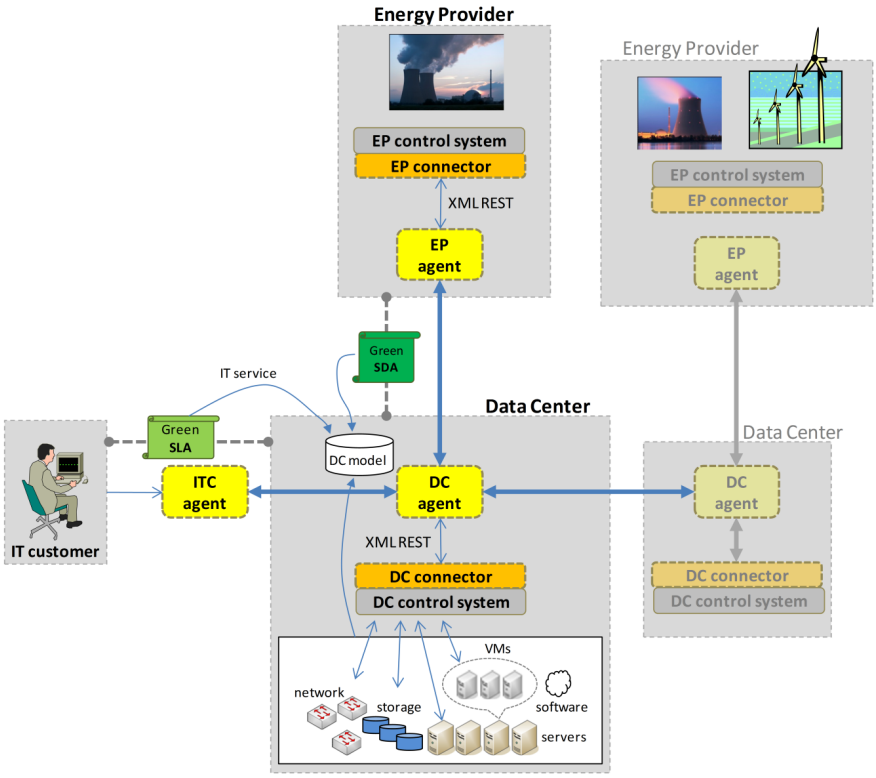

Figure 3: Overview of the All4Green system

1.4.2.1 *States of the DC*

The approach of the project to the problem involves categorizing the possible situations a DC can be in. This situations, or *states* of the DC, will change according to EP's demands and the availability of the ITC end users. In one of the deliverables of the project All4Green (WP3) we find a classification of the different states of the DC (Fig. 4).

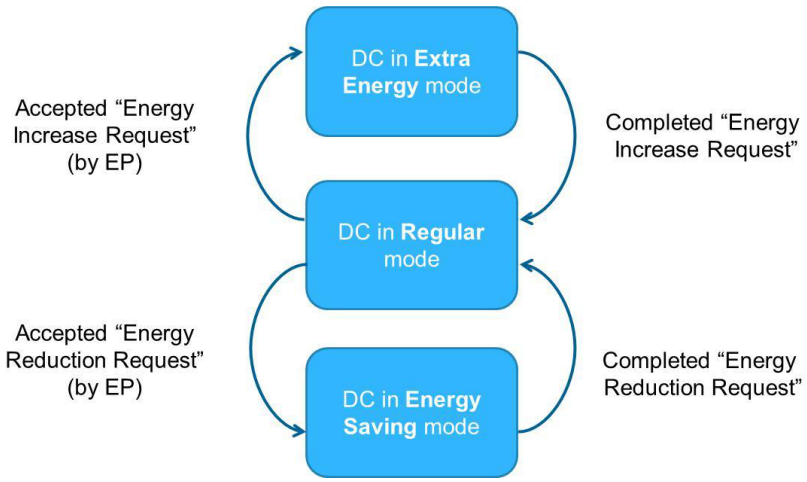

Figure 4: Different states of the DC

Each DC state comes with suggestions on policies and actions, as can be seen in Table 1, which has some minor changes to the one that appears in the project adapting to the necessities of this work.

Table 1: Policies and actions taken in each state of the DC

DC(Agent) Actions	Regular Mode	Energy Saving Mode	Extra Energy Mode
Allocation of new IT Service Requests	Local DC	Local DC	Local DC
Flexibility GreenSLAs: generic context dependent SLA value settings	YES	YES	YES
Energy context dependent SLAs		YES	YES
Collaboration GreenSLAs		YES	YES
user/system job shifting	regular	postpone	anticipate
pause, resume, or start extra services	resume	pause	extra services
AC tuning	Regular	Increase temperature	Reduce temperature

1.5 THE IMPLEMENTATION

WP6 is developing a simulator, and it will be a bench for testing models and strategies proposed in WP3, WP4 and WP5. The way it is done isn't a major concern in this work (understanding it as the "'PFC"'), but it will be web-based and will use a Tomcat server + Java + WS-agreements.

2

COMBINATORIAL OPTIMIZATION: BIN PACKING PROBLEM

INTRODUCTION

Most of the real world problems can be modeled as a mathematical problem that will be more or less accurate depending of the amount of information provided. Mathematical modelling is a very extensive field and arguably one of the oldest fields at least in applied Mathematics. As a field, its importance has grown exponentially with the arrival of the computers that allow massive calculations in few time. Therefore many different approaches have been derived, between three distinct branches: Continuous modelling, discrete modelling and statistical modelling (specially important nowadays with Big Data). Most problems are better suited for only one kind of modelling but some allow different approaches and even mixtures.

In this chapter we will discuss the nature of optimization and decision problems and the diverse ways to approach a solution. First we will define what an optimization problem is in rigurous terms and we will describe a (very famous) way to achieve an idea of its difficulty. Two main references will be used during this part of the chapter: [4] and [5]. In the second part of the chapter a problem very related to this project and its variations will be presented, together with the different algorithms that try to solve it and a small analysis.

The algorithms in pseudocode will be presented in the course of the chapter, however those that can be found in the references will not be proven and only the new results will.

2.1 TRACTABILITY OF OPTIMIZATION PROBLEMS

The first step is to understand what is an optimization problem. In plain words, in an optimization problem we receive some data, that we would divide in cost coefficients and restriction terms. Each possible input data is associated with a set of possible solutions and each pair *input data, solution* has a certan value defined by a function. The final objective is to maximize or minimize this value. Formally, a definition (seen in [4]) is as follows.

Definition 2.1.1. An optimization problem Q
is a 4-tuple $\langle I_Q, S_Q, f_Q, \mathrm{opt}_Q \rangle$ where I_Q is the set of input instances, S_Q is a function such that for each input $x \in I_Q$, $S_Q(x)$ is a set of solutions to x, f_Q is an integer-valued function that evaluates each

par $x \in I_Q$ and $y \in S_Q(x)$, and $opt_Q \in \{max,min\}$ specifies the problem to be a maximum problem or a minimum problem.

A very famous example of an optimization problem is, for example, the MINIMUM SPANNING TREE (MSP), where we must find the cheapest subnetwork in a network that connects all nodes. I_Q would be here the set of all weighted graphs G, thus $G \in IQ$ would be a weighted graph, $S_Q(G)$ is the set of all spanning trees of the graph G, $f_Q(G,T)$ is the weight of the spanning tree T of G, and the objective opt_Q is to minimize.

A special sort of optimization problems are called *decision problems*. In this kind of problems each input instance only admits one of the two possible answers –yes or no. An input taking the 'yes' answer will be a *yes-instance* for the problem and an input taking the 'no' answer will be a *no-instance* for the problem. The most famous decision problem is known as SAT (Satisfiability). A formulation is: 'Given a boolean formula F in the conjunctive normal form, is there an assignment to the variables in F so that the formula F has value TRUE?'.

A decision problem can be formed from an optimization problem by adding a parameter C. Then the question changes to: 'Given an instance $x \in I_Q$ is there a solution $y \in S_Q(x)$ such that $f_Q(x,y) - C$ is positive/negative (according to opt_Q)?'

2.1.1 *Algorithms, encoding and asymptotic notation*

The big issue with optimization problems and what makes them so difficult is that the objective is not to solve a given problem, i. e.solve the problem for a *given* instance $x \in I_Q$, but we want a way of finding the solution for *any* $x \in I_Q$. We want an *algorithm*.

An algorithm is a step-by-step specification of a procedure for solving a given problem. Each step of an algorithm consists of a finite number of operations. The algorithms in this thesis will be written in pseudocode with certain flexibilities. We say that an algorithm $\mathcal{A}$ solves the problem Q if, for any $x \in I_Q$ the algorithm produces a solution $\mathcal{A}(x) = y \in S_Q$ such that $f_Q(x, \mathcal{A}(x)) = opt_Q\{f_Q(x,y)|z \in S_Q(x)\}$. I. e., it produces an optimal solution for each input instance.

In a computer, everything is encoded, i. e.given as a sequence of symbols of a finite alphabet Σ. For example, an input instance of the problem MSP would be given by the adjacency matrix of the weighted graph (organized by rows, for example). The alphabet of a computer is $\{0,1\}$ and everything is encoded in a finite sequence of 0 and 1. Therefore, if the alphabet Σ has n elements, then each element can be encoded into a distinct binary sequence of $\lceil log_2(n) \rceil$ bits. If a sequence of elements of the alphabet has length m, then it can be encoded with $m\lceil log_2(n) \rceil$. Usually n is not a big number and therefore the binary representation is not much larger than the

original sequence. From now on, the length of an object w is $|w|$ and it refers to the length of its binary encoding.

To know the complexity of an algorithm $\mathcal{A}$ we must study how many operations does it take for it to solve a problem depending on the 'size' of the problem. In general it is difficult to find this complexity, but several asymptotic bounds to it can be found more easily. The notation is as follows, given $t(n) : \mathbb{N} \to \mathbb{R}$:

- We define $O(t(n))$ such that $f \in O(t(n))$ if, and only if, there exists a constant c_f such that $\forall n > n_0\ c_f f(n) \leqslant t(n)$.
- We define $o(t(n))$ such that $f \in o(t(n))$ if, and only if,
$$\lim_{n\to\infty} \frac{f(n)}{t(n)} = 0$$
- We define $\Omega(t(n))$ such that $f \in \Omega(t(n))$ if, and only if, there exists a constant c_f such that $\forall n > n_0\ c_f f(n) \geqslant t(n)$.
- We define $\omega(t(n))$ such that $f \in \omega(t(n))$ if, and only if,
$$\lim_{n\to\infty} \frac{t(n)}{f(n)} = 0$$
- We define $\Theta(t(n))$ such that $f \in \Theta(t(n))$ if, and only if, $f \in O(t(n)) \cap \Omega(t(n))$.

With these notations, we can define the time complexity of an algorithm $\mathcal{A}$. If we define the running time of an algorithm as the number of basic operations during the execution of the algorithm, then:

Definition 2.1.2. Let $\mathcal{A}$ be an algorithm solving an optimization problem Q and $f(n)$ a function. The time complexity of $\mathcal{A}$ is $O(f(n))$ if there is a function $f'(n) \in O(f(n))$ such that for every integer $n \geqslant 0$, the running time of $\mathcal{A}$ is bounded by $f'(n)$ for all input instances of size n.

Definition 2.1.3. An algorithm $\mathcal{A}$ is said to be a polynomial time algorithm if there is a fixed constant $c \geqslant 0$ such that the time complexity of $\mathcal{A}$ is $O(n^c)$. An optimization problem can be solved in polynomial time if it can be solved by a polynomial time algorithm.

Due to the fact that the binary encoding only multiplies by a constant factor, the time complexity of an algorithm does not depend on how its solutions are encoded.

It is important to remark that this analysis of the time complexity of an algorithm is always based on a worst-case situation. I. e., the last part of Def. 2.1.2 indicates that for all input instances of size n, the running time is bounded by $f'(n)$. Usually in most of the problems the instances will have a probability distribution and will not reach the boundary. In this case, average analysis or smoothed analysis of algorithms arises as the best tool. However it requires knowledge of the statistics of the problem.

2.1.2 *NP-completeness*

The NP-completeness theory studies decision problems and their tractability. Although the final objective is to study optimization problems as well, decision problems are easier and therefore, if the decision problem version is very hard, it is to be expected that the optimization problem will be very hard.

An algorithm $\mathcal{A}$ is said to *accept* a decision problem Q if on every yes-instance $x \in I_Q$ the algorithm stops at a 'yes' state, while on all other instances (including those that do not encode a correct input $x \in I_Q$) the algorithm stops at a 'no' state (rejects x).

Definition 2.1.4. A decision problem is said to be in the class P if it can be accepted by a polynomial time algorithm.

The same definition is correct for optimization problems as well.

It is, in general, 'easy' to see if a problem is in P once an algorithm has been found. However, not finding a polynomial time algorithm does not prove that the problem is not in P. It is very difficult to prove that there is no polynomial time algorithm that accepts a problem. In fact, a very hard branch of computer science and mathematical logic is 'decidability' or 'computability', where the question is: 'Is there an algorithm at all that will accept the problem?'. This subsection will cover the discussion about seeing if a problem is or not in P.

For some problems it has not been possible to prove that they don't belong to P and yet a polynomial time algorithm that accepts them has not been found. There is a definition for this class of problems (or at least a part of it).

Definition 2.1.5. A decision problem is said to be in the class NP if there is a polynomial time algorithm $\mathcal{A}$ that accepts it in the following manner. There is a fixed polynomial $p(n)$ such that

1. If x is a yes-instance for the problem Q, then there is a binary string y of length bounded by $p(|x|)$ such that on input (x, y) the algorithm $\mathcal{A}$ stops at a yes state.

2. If x is a no-instance for the problem Q, then for any binary string y of length bounded by $p(|x|)$, on input (x, y) the algorithm $\mathcal{A}$ stops at a no state.

It is easy to see that if a decision problem is in P, then it is in NP if we choose $p(n) = 0$. Two important remarks must be done about this definition. First, a problem in NP can be easily *checked* by an algorithm $\mathcal{A}$. This means that the algorithm $\mathcal{A}$ can solve the decision problem if a hint is given. Proof checking is a suitable analogy for this: The algorithm accepts a theorem if the 'short' proof given is valid (the algorithm checks the proof) but it has no way of proving itself the theorem. This is the reason why this class of problems is

said to be able to be solved by non-deterministic machines that can 'guess' the aforementioned y.

Another important remark is that this algorithm has no way to determine for sure if the instance is a no-instance, because no matter how many 'hints' we give it, we can not say for sure that there is no hint that would result in a 'yes' instance.

This definition is not as easy to extend to optimization problems as P.

Definition 2.1.6. An optimization problem $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ is an NP optimization (NPO) problem if there is a polynomial $p(n)$ such that for any instance $x \in I_Q$, there is an optimal solution $y \in S_Q$ whose length $|y|$ is bounded by $p(|x|)$.

Most optimization problems are in NPO. For example the famous Travel Salesman Problem (TSP) or the aforementioned MSP are NPO. It is easy to see that a NPO problem has a decision version which is in NP (just give as a hint $y \in S_Q$ and all yes-instances will return yes and all no-instances will return no).

One more definition is needed to 'order' problems in terms of difficulty.

Definition 2.1.7. Let Q_1 and Q_2 be two decision problems. Problem Q_1 is polynomial time (many-one) reducible to problem Q_2 (written as $Q_1 \leqslant_m^p Q_2$) if there is a function r computable in polynomial time such that for any x, x is a yes-instance for Q_1 if and only if $r(x)$ is a yes-instance for Q_2.

Some natural results follow from this definition, for example if a problem Q_2 is in class P and there is a problem Q_1 such that $Q_1 \leqslant_m^p Q_2$, then Q_1 is in P as well (this will not be proved as it is not relevant to the development of this thesis). This gives us an idea that this 'order' is a way of determining which problems are harder. There is also a theorem whose prove will not be essential to the development of this thesis but the theorem itself will be essential to understand the theory of NP-completeness.

Cook's Theorem. *Every decision problem in the class* NP *is polynomial time many-one reducible to the SATISFIABILITY problem.*

This theorem shows us that no decision problem in class NP is 'harder' than the SAT problem. It leads us to the following two definitions about problem complexity.

Definition 2.1.8. A decision problem Q is $NP-hard$ if every problem in the class NP is polynomial time many-one reducible to Q.

Definition 2.1.9. A decision problem Q is $NP-complete$ if it is $NP-hard$ and it is in NP.

In particular, SAT is $NP-complete$ (it is easy to show that it is in NP). According to what has been shown, if a $NP-hard$ problem can be solved in polynomial time, i. e.it is in P, then all problems in NP are as well in P. However there are very hard problems and during the last decades many mathematicians and computer scientists have tried to show that they are in P without any success. This leads to the generalized opinion that not all problems that are in NP (in particular, the $NP-complete$ problems) are in P as well.

Two important results for the development of the thesis are needed.

Lemma 2.1.1. *Let* Q_1, Q_2 *and* Q_3 *be three decision problems. If* $Q_1 \leqslant_m^p Q_2$ *and* $Q_2 \leqslant_m^p Q_3$ *then* $Q_1 \leqslant_m^p Q_3$

Proof. Let r_1 be the polynomial-time ($O(n^{q_1})$) computable function that reduces instances from Q_1 to instances for Q_2. Let r_2 be the ($O(n^{q_2})$) function that reduces instances from Q_2 to instances for Q_3. Let $r = r_2 \circ r_1$. It is also polynomial time computable: It needs $O(n^{q_1} + n^{q_2}) = O(n^{max(q_1,q_2)})$ operations. It reduces instances from Q_1 to Q_3: x is a yes-instance for Q_1 if and only if $x' = r_1(x)$ is a yes-instance for Q_2, if and only if $r_2(x') = r_2(r_1(x)) = r(x)$ is a yes-instance for Q_3. This shows that $Q_1 \leqslant_m^p Q_3$. □

Corollary. *Let* Q_1 *and* Q_2 *be two decision problems. If* Q_1 *is* $NP-hard$ *and* $Q_1 \leqslant_m^p Q_2$ *then* Q_2 *is also* $NP-hard$.

Proof. Let Q be any decision problem in NP. Then by the definition of $NP-hard$ problems, $Q \leqslant_m^p Q_1$. Using this and the previous lemma, we have that $Q \leqslant_m^p Q_2$ and thus Q_2 is $NP-hard$. □

This leads to a way of determining the hardness of a problem. If we suspect that a problem is very difficult, we might want to try to find a $NP-hard$ problem and see if that problem is polynomial time many-one reducible to our problem. If it is, then our problem is as well $NP-hard$ and thus it will be wise to stop looking for polynomial time algorithms that will solve it. Of course this is all based in the following conjecture that has already been mentioned previously.

Conjecture. $P \neq NP$, i. e.there are problems in NP that are not in P.

So far it has been impossible to prove this, however it is strongly believed by most of the computer scientists and it has a lot of evidencial support.

The next step is to extend these definitions and results to optimization problems.

Definition 2.1.10. A decision problem D is polynomial time reducible to an optimization problem $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ if there are two polynomial time computable functions h and g such that:

1. Given an input instance x for the decision problem D, $h(x)$ is an input instance for the optimization problem Q.

2. For any solution $y \in S_Q(h(x))$, $g(x, h(x), y) = 1$ if and only if y is an optimal solution to $h(x)$ and x is a yes-instance for D.

If a decision problem can be polynomial time reduced to an optimization problem, then the decision problem can not be much harder than the optimization problem.

Lemma 2.1.2. *Suppose that a decision problem* D *is polynomial time reducible to an optimization problem* Q*. If* Q *is solvable in polynomial time, then so is* D*.*

Proof. Let h and g be two polynomial time computable functions as seen in def 2.1.10 for the reduction from D to Q. Let $\mathcal{A}$ be a polynomial time algorithm that solves the optimization problem Q. Now a polynomial time algorithm for the decision problem D can be derived as follows: given an instance x for D, we construct $h(x)$ for Q and apply $y = \mathcal{A}(h(x))$ to find the optimal solution for $h(x)$. x is a yes-instance for D if and only if $g(x, h(x), y) = 1$. By definition, all the calculations are polynomial time computable. Thus, this algorithm runs in polynomial time and correctly decides if x is a yes-instance for D. □

Definition 2.1.10 leads to a definition of $NP - hardness$ in terms of optimization problems.

Definition 2.1.11. An optimization problem Q is $NP - hard$ if there is an $NP - hard$ decision problem D that is polynomial time reducible to Q.

If a decision problem derives from an optimization problem and it is $NP - hard$, then the optimization problem itself is $NP - hard$ as well (h only needs to be the identity except the parameter, and g a simple comparison between the optimal solution and the parameter). An example of an $NP - hard$ optimization problem is Integer Linear Programming (Integer LP). A similar definition to being polynomial time many-one reducible can be made for optimization problems as well.

Definition 2.1.12. An optimization problem Q_1 is polynomial time reducible (or p-reducible) to an optimization problem Q_2 if there are two polynomial time computable functions χ and ψ such that:

1. For any instance $x \in I_{Q_1}$, $\chi(x) \in I_{Q_2}$.

2. For any solution y_2 to the instance $\chi(x_1)$, $\psi(x_1, \chi(x_1), y)$ is a solution to x_1 such that y_2 is an optimal solution to $\chi(x_1)$ if and only if $\psi(x_1, \chi(x_1), y)$ is an optimal solution to x_1.

From this definition two lemmas come naturally. The proof is very similar to the proofs already written and will therefore be included only in the appendix.

Lemma 2.1.3. *Suppose that an optimization problem* Q_1 *is p-reducible to an optimization problem* Q_2*. If* Q_2 *is solvable in polynomial time then so is* Q_1

Lemma 2.1.4. *Suppose that an optimization problem* Q_1 *is p-reducible to an optimization problem* Q_2*. If* Q_1 *is* $NP - hard$*, then so is* Q_2*.*

Some optimization problems have subproblems that are easier to identify than by p-reducibility. This is what the next definition states.

Definition 2.1.13. Let $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ be an optimization problem. An optimization problem Q' is a subproblem of Q if $Q' = \langle I'_Q, S_Q, f_Q, opt_Q \rangle$ where $I'_Q \subset I_Q$.

Note that in a subproblem every input instance is also an input instance of the problem, and every input instance in the subproblem has the same set $S_Q(x)$. We could say that the only difference between a problem and its subproblem is the number of instances that we allow to be settled as input (for example, forbidding the more pathological input instances). The next theorem arises naturally from the definition.

Theorem 2.1.5. *Let* Q *be an optimization problem and* Q' *be a subproblem of* Q*. If the subproblem* Q' *is* $NP - hard$*, then* Q *is* $NP - hard$*.*

2.1.3 *Tractability Theory: Polynomial time approximation, asymptotic approximation*

Once we have convinced ourselves that some problems are probably too hard to find the optimal solution in a fast and efficient way, the next natural step is to try to relax the requirement that we always find the optimal solution, and find an 'almost optimal' solution instead. Sometimes we even would have enough finding *a* solution for the problem.

Definition 2.1.14. An algorithm $\mathcal{A}$ is an approximation algorithm for a problem $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ if on any input $x \in I_Q$, the algorithm produces an output $\mathcal{A}(x) \in S_Q(x)$.

This is of course a very lax definition and it does not help us to know how good is this algorithm, for that we will use the approximation ratio of the algorithm.

Definition 2.1.15. An approximation algorithm $\mathcal{A}$ for an optimization problem $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ has an approximation ratio $r(n)$ if on any input instance $x \in I_Q$, the solution $\mathcal{A}(x)$ satisfies

$$\left| \frac{Opt(x)}{f_Q(x, \mathcal{A}(x))} \right| \leqslant r(|x|) \text{ if } opt_Q = \max$$

$$\left| \frac{f_Q(x, \mathcal{A}(x))}{Opt(x)} \right| \leqslant r(|x|) \text{ if } opt_Q = \min$$

Where $Opt(x)$ is defined to be $\max\{f(x,y)|y \in S_Q(x)\}$ if $opt_Q = \max$ and to be $\min\{f(x,y)|y \in S_Q(x)\}$ if $opt_Q = \min$.

It is important to not confuse this definition with the competitive ratio definition that will be explained in detail later. Note that $r(n)$ is always at least as large as 1 (being 1 if $\mathcal{A}$ solves the problem Q).

Definition 2.1.16. An optimization problem Q can be polynomial time approximated to a ratio $r(n)$ if it has a polynomial time approximation algorithm whose approximation ratio is $r(n)$.

There is a class of $NP-hard$ optimization problems, most of them coming from scheduling problems, can be polynomial time approximated to a ratio $1+\epsilon$, for any ϵ. The running time of the algorithms that approximate these problems grow with $\frac{1}{\epsilon}$, but in a polynomial way, i. e.$O(\frac{1}{\epsilon^n})$. These families of algorithms are called fully polynomial time approximation schemes for the $NP-hard$ optimization problems. The first step is to study the pseudopolynomial running time algorithms and for this it is necessary to take into account not only the length of the input but also the maximum value of it.

Definition 2.1.17. Suppose $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ is an optimization problem. For each input instance $x \in I_Q$ we define:

- $length(x)$ = the length of a binary encoding of x.
- $max(x)$ = the largest number that appears in the input x. If no numbers appear in the input, then $max(x) = 1$.

Definition 2.1.18. Let Q be an optimization problem. An algorithm $\mathcal{A}$ solving Q runs in pseudopolynomial time if there is a two-variable polynomial p such that on any input instance x of Q, the running time of the algorithm $\mathcal{A}$ is bounded by $p(length(x), max(x))$. In this case, we also say that the problem Q is solvable in pseudopolynomial time.

The positive aspect of depending on $max(x)$ is that sometimes it makes sense and it is possible to rescale the problem to reduce $max(x)$ convert a problem to a problem solvable in pseudopolynomial time. For example, if $max(x)$ can be scaled to be bounded by a polynomial of $length(x)$ then the problem can be solved in polynomial time.

Definition 2.1.19. An optimization problem Q has a fully polynomial time approximation scheme (FPTAS) if it has an approximation algorithm $\mathcal{A}$ such that given (x, ϵ) where $x \in I_Q$ and $\epsilon > 0$, $\mathcal{A}$ finds a solution for x with approximation ratio bounded by $1 + \epsilon$ in time polynomial in both n and $\frac{1}{\epsilon}$.

There is an important theorem that shows that having a pseudopolynomial time algorithm for a problem is usually a necessary condition for the existence of a FPTAS.

Theorem 2.1.6. *Let* $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ *be an optimization problem such that for any* $x \in I_Q$ *we have* $Opt(x) \leqslant p(length(x), max(x))$ *for a given polynomial* p. *If* Q *has a FPTAS, then* Q *can be solved in pseudopolynomial time.*

There are ways to improve the time complexity of the algorithms that solve problems. Apart from the aforementioned rescaling of the problem, we can reduce the number of parameters, for example storing values that are to be reused. Or we can reduce the search space by running first a worse but faster algorithm that will give rough boundaries to the solution (a similar method called cutting planes is used in Mixed Integer Programming). Another popular technique is to separate items by size, establishing a threshold and dividing the items in two groups (the big and the small elements).

Unfortunately, there are problems that have no FPTAS, and alas the problem that will be the cornerstone of this thesis is one of those. There is a result that allows us to easily determine so and we need a few more definitions to that end.

Definition 2.1.20. Let $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ be an optimization problem. For each input instance $x \in I_Q$ define

$$Opt_Q(x) = opt_Q\{f_Q(x, y) | y \in S_Q(x)\}$$

The following theorem will erase all hope for our problem to have a FPTAS.

Theorem 2.1.7. *Let* $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ *be an optimization problem. If there is a fixed polynomial* p *such that for all* $x \in I_Q$, $Opt_Q(x)$ *is bounded by* $p(|x|)$, *then* Q *does not have a FPTAS unless* Q *is in* P.

Theorem 2.1.7 shows that it will be common that a problem does not have a FPTAS. Some problems don't even have a polynomial time approximation algorithm for any $\epsilon < c$ for some given c. However, some problems can have an approximation ratio as close to 1 as we want when the solution is large enough, i. e.asymptotically. The next definitions and theorems will lead us to the asymptotic approximation schemes and algorithms. These will be based on minimization problems although it is easy to extend them to maximization problems as well.

Definition 2.1.21. Let $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ be a minimization problem and let $\mathcal{A}$ be an approximation algorithm for Q. The asymptotic approximation ratio of A is bounded by r_0 if for any $r > r_0$ there is an integer N such that for any input instance $x \in I_Q$ with $Opt(x) \leqslant N$, the algorithm $\mathcal{A}$ constructs a solution to x satisfying $\frac{A(x)}{Opt(x)} \leqslant r$.

Definition 2.1.22. An optimization problem Q has an asymptotic fully polynomial time approximation scheme (AFPTAS) if it has a family of approximation algorithms $\{A_\epsilon | \epsilon > 0\}$ such that for any $\epsilon > 0$, A_ϵ is an approximation algorithm for Q of asymptotic approximation ratio bounded by $1 + \epsilon$ and of running time bounded by a polynomial of the input length and $\frac{1}{\epsilon}$.

Finally, two more simple definitions that will appear later in the thesis are presented.

Definition 2.1.23. An optimization problem Q admits a polynomial time approximation scheme (PTAS), if for every $\epsilon > 0$, there is an algorithm $\mathcal{A}$ of running time bounded by a polynomial of the input lengthe such that the approximation ratio for $\mathcal{A}$ is $1 + \epsilon$.

Note that this definition does not mention ϵ, so usual running times of such algorithms could be, for example, $O(n^{(1/\epsilon)!})$.

Definition 2.1.24. An optimization problem Q is in APX if it allows polynomial time approximation algorithms with approximation ratio bounded by a constant α.

2.2 ONLINE DECISION PROBLEMS

This section is mostly based on [5], where more thorough definitions can be found. An online decision problem is a decision problem in which we do not know the whole I_Q but it is updated in real time.

Definition 2.2.1. In an online decision/optimization problem, the input data that arrives between t and $t + \Delta t$ for a given Δt is $x(t, \Delta t)$. The possible input instances that will have arrived until the instant t' knowing that in instant $t < t'$ the data received is $x(t)$ is $I_{Q(t')|x(t)}$ and all its elements are of the form $x(t, t' - t) \in I_{Q(t')|x(t)}$.

Remark. An online optimization problem $Q(t) = \langle I_{Q(t)}, S_{Q(t)}, f_{Q(t)}, opt_{Q(t)} \rangle$ is an optimization problem with the following restrictions:

1. For all t, t' with $t < t'$, every element $x(t') \in I_{Q(t')}$ is of the form $x(t)||x(t, t' - t)$ for some $x(t) \in I_{Q(t)}$, $x(t, t' - t) \in I_{Q(t')|x(t)}$ where $||$ is the concatenation of symbols of the alphabet. This means that we are allowed to store information of all events in the past (it can be modified if we don't want to store all events in the past).

2. If $x(t) = x(t')$ for $t \neq t'$ then $S_{Q(t)}(x(t)) = S_{Q(t')}(x(t'))$, i. e.the solutions don't change if the input instance doesn't change, independently of the moment. One could argue that the solution in a future is depending on the solution in the past but that is not always like that. However the solution in t usually imposes restrictions on the solution in t'.

3. The same can be said for $f_{Q(t)}$, we only can assure that if the pair 'input, solution' is the same, $f_{Q(t)}$ will be the same no matter t.

4. $opt_{Q(t)}$ is constant.

An algorithm that solves an online decision problem must make a decision, i. e.find a
solution $y \in S_{Q(t)}(x)$If $x \in I_Q, x = \{constraints, data\}$, we could say that in the online version of the problem the constraints change in time and data is added and removed systematically. Therefore, an algorithm that solves the online version of a problem will make the decision based only in the data received until that moment and the current constraints. There is a very important concept related to online decision problems: The competitive ratio. For $x \in I_Q$ let $OPT(x)$ be the offline optimal solution of a problem, that is if all the data was known at the beginning. Let $A(x)$ be the output of the algorithm that tries to solve the online problem for that data. Then (Eq. 1), the competitive ratio is the worst-case relation between the offline optimal solution and the solution given by the online algorithm.

Definition 2.2.2. Let $Q(t)$ be an online decision problem and $\mathcal{A}$ an approximation algorithm for the online decision problem. Then the competitive ratio $c(A)$ is:

$$c(A) = sup_{x \in I_Q} \frac{A(x)}{OPT(x)} \tag{1}$$

Note that $x \in I_Q$ represents all the data received in a 'significant interval' of time. For example in the paging problem it would represent all the data until the cache memory is restarted.

Here, worst-case is the keyword. There is another way of analyzing the outcome of the algorithms (offline or online) in an average situation [6], but it requires further probabilistic and statistic study (compared to the combinatoric approach of a worst-case study). However, some problems that appear to be very hard and have very bad boundaries on the worst-case scenario, might admit a polynomial time algorithm that will give the optimal solution in an average scenario.

2.3 BIN PACKING PROBLEM AND VARIATIONS

A very famous NP-hard [4, p. 210] problem is the so-called Bin Packing Problem (BPP). Using the formal definition, the problem is formulated as:

- I_Q: the set of tuples $\alpha =< s_1, \ldots, s_n; T >\in I_Q$, with $s_i \leqslant T \forall i$, $s_i \in \mathbb{Z}$ and $T \in \mathbb{Z}$. Sometimes it can be seen as $s_i \in (0, 1]$ and $T = 1$.
- $S_Q(\alpha)$: the set of partitions ("packings") $Y = (B_1, \ldots, B_r)$ of $\{s_1, \ldots, s_n\}$ such that $\sum_{s_i \in B_j} s_i \leqslant T$ for all j
- $f_Q(\alpha, Y)$: the number of subsets ("packs") in the partition Y of α.
- $opt_Q = min$

The items s_i are the items that we pack in bins of size B. Our objective is to minimize the number of bins used (or the total waste in the used bins). This problem has been used for stock cutting (using the least possible bars of longitude 1 and cut them into different sizes), transport scheduling (packing trucks with boxes), and job scheduling, which will be the main intention in this thesis.

The decision version problem of BPP is $NP - complete$, more specifically:

Theorem 2.3.1. *It is* $NP - complete$ *to decide if an instance of BPP admits a solution with two bins.*

BPP is also a NPO problem.

Lemma 2.3.2. *BPP is* NPO, *furthermore BPP does not have a* FPTAS.

Proof. If α consists in n elements, then the optimal solution will have at most n subsets. This is a polynomial that bounds $Opt(\alpha) < |\alpha|$. □

In fact, there is no approximation algorithm for the BPP that has a better approximation ratio than 1.5.

Theorem 2.3.3. *BPP is in* APX, *being* 1.5 *the boundary for its approximation ratio, unless* $P = NP$.

There are many algorithms that approach the BPP and we will not attempt to give them all, only a short description and their approximation ratios. More information about them can be found in [7]. For commodity we will use the following notation:

Definition 2.3.1. We will write the approximation ratio, and asymptotic approximation ratio of the BPP for a given algorithm $\mathcal{A}$ as R_A and R_A^∞. If the size of the elements in the problem is bounded by α, then we will write $R_A(\alpha)$ and $R_A^\infty(\alpha)$ the approximation ratio and asymptotic approximation ratio, respectively, of the algorithm working on lists with size bounded by α.

2.3.0.1 *Online algorithms*

The first algorithm that comes to mind is Next-Fit. It always has only one bin open, (this is called bounded space algorithm), and it works as can be seen in Listing 1

Listing 1: Next Fit Algorithm

```
Input: α : ⟨s_1,...,s_n;T⟩
Output: A packing of the items s_1,...,s_n into bins of size T
0. Suppose that all bins B_1,B_2,... are empty, start by B_1
1. for i = 1 to n do
2.    if B_j free space is greater than s_i
3.        put the item s_i in B_j
4.    else open B_{j+1}, put the item s_i in B_{j+1}, close B_j
5. end for
```

This algorithm runs in $O(n)$ (linear time), it is online (we don't need to know the future comings) it has an approximation ratio $R_{NF} = 2$ which is a tight bound because the list $L = \langle \frac{1}{2}, \frac{1}{2N}, \ldots, \frac{1}{2}, \frac{1}{2N} \rangle$ with this algorithm is put in $NF(L) = 2 * OPT(L) - 1$. Therefore $R^{\infty}_{NF} = 2$. Further results shown in [7] prove that $R^{\infty}_{NF}(\alpha) = 2$ for all $\alpha > \frac{T}{2}$ and for alpha $0 \leqslant \alpha \leqslant T/2$ $R^{\infty}_{NF}(\alpha) = \frac{1}{1-\alpha}$.

The second algorithm that tried to solve the problem is called First-Fit: for each s_i from $i = 1$ to n, put the item s_i in the first bin it fits.

Listing 2: First Fit Algorithm

```
Input: α : ⟨s_1,...,s_n;T⟩
Output: A packing of the items s_1,...,s_n into bins of size T
0. Suppose that all bins B_1,B_2,... are empty, start by B_1
1. for i = 1 to n do
2.    j=1
3.    while B_j free space is lesser than s_i, increase j
4.    Put the item s_i in the first B_j it fits.
5. end for
```

This algorithm is also online, it runs in $O(n^2)$ ($O(n \log n)$ if a proper data structure is used), and it has an approximation ratio $R_{NF} = 1.7$. In fact $FF(L) \leqslant \lceil 1.7 OPT(L) \rceil$. Furthermore, it can be shown that:

Theorem 2.3.4 ([7]). *Let* $m \in \mathbb{Z}$ *such that* $\frac{1}{m+1} < \alpha \leqslant \frac{1}{m}$.

1. *For* $m = 1$, $R^{\infty}_{FF}(\alpha) = \frac{17}{10}$.
2. *For* $m > 1$, $R^{\infty}_{FF}(\alpha) = 1 + \frac{1}{m}$.

There are many algorithms that work similarly and it can be proven that they behave similarly to NF and FF. We will call Any Fit (AF) those algorithms that will not pack an item in a new bin unless all the partially filled bins do not have enough space for the item to fit,

and Almost Any Fit (AAF) those algorithms that will not pack an item into a partially filled bin with the lowest level (meaning: being the least filled) unless there is more than such bin – or that bin is the only one to have enough room. Then we have the following result:

Theorem 2.3.5. *For all* $\alpha \in (0, 1]$

1. *If* $\mathcal{A}$ *is an AF algorithm, then* $R^{\infty}_{FF}(\alpha) \leqslant R^{\infty}_{A}(\alpha) \leqslant R^{\infty}_{NF}(\alpha)$.
2. *If* $\mathcal{A}$ *is an AAF algorithm, then* $R^{\infty}_{A}(\alpha) = R^{\infty}_{FF}(\alpha)$.

There are another type of algorithms (also on-line) called Harmonic algorithms H_k, where the interval of possible data is divided into k subintervals and each item is classified. These algorithms can break the 1.7 barrier when $k > 6,7$ (depending on which version is used), but they can not go further than $1.69103\ldots$, and they add complexity to the calculations. Furthermore, no algorithm that has a bounded-space restriction, i. e.having at most N open bins each time, can do better than $1.69103\ldots$. However there is an algorithm (and a subsequent chain of slight improvements) that is online and breaks the 1.7 barrier. It is called Refined First Fit (RFF) and it also divides the items' sizes into various possibilities, and then packs them according to a special strategy. It has a ratio of $R^{\infty}_{RFF} = 1.6666\ldots$. Currently the best known online algorithm for the BPP has a ratio $R^{\infty}_{A} = 1.588$ and it has been proved that no online algorithm can have a ratio $R^{\infty}_{A} = 1.540$. Even in randomized online algorithms it has been proved that there are lists that yield ratios of $E(A(L))/OPT(L)$ approaching 1.536....

One particular interesting variation of the online BPP is the so-called Dynamic Bin Packing Problem (DBPP), where the items depart at some time (leaving empty space) and the algorithm is not allowed to repack items. In [8] a lower bound for this kind of online algorithms is found at 2.5, although if the input data is simple enough (can be written as fractions of the form $\frac{1}{k}$) then there is an algorithm that solves it in 2.4985.

2.3.0.2 *Semi-online algorithms*

Semi-online algorithms for the BPP are those that admit repacking of the items when a new item arrives. Usually they put a limit on the repacking (otherwise it would simply be offline), for example, a linear delay. They admit several algorithms, but it is more interesting to focus on the variation Fully Dynamic Bin Packing Problem (FDBPP) where items also depart and they can be repacked every time an item arrives or departs. [9] shows an algorithm that runs in $\Theta(\log n)$ for each item arrival and has an asymptotic competitive ratio of $\frac{5}{4}$ (asymptotic in the usual sense, i. e.it tends to this ratio when the optimal solution is large enough).

2.3.0.3 *Offline algorithms*

Offline algorithms can also be applied to Online situations when a full repacking of the items can be applied each time there is a new item arrival/departure. The most famous and simple one is the First Fit Decreasing algorithm. The First-Fit-Decreasing (FFD) algorithm is the same as the First Fit (FF) algorithm but it sorts the items first from biggest to smallest has a worst-case performance of $1.22 \cdot OPT + 4$ and runs in $O(n\log(n))$ (if a proper data structure is used). Its approximation (not asymptotic) ratio is 1.5, so it is the best approximation algorithm possible for the BPP. However it is not the best asymptotically approximation algorithm possible for the BPP. Indeed, the BPP admits an AFPTAS [4]. Moreover, if we restrict the BPP to having at most π different sizes (bounded by δ) then there is a polynomial algorithm that solves the problem and finds the optimal solution. It consists in finding all the possible combinations of elements that fit into a single bin, and then find the best combination of such bins. Given π and δ, the number of combinations is a polynomial in n. However both the AFPTAS and this exact algorithm are not practical due to the degree of the polynomial.

Fortunately, an average-case analysis of the FFD algorithm shows that it behaves in average as well as an optimal algorithm [10]. In fact, when comparing the results of the FFD with a metaheuristic, it is true that in 66% of the cases, they give the same result, and the other 34% situations only differ by 1 bin. Of course this does not prove anything, but it is a way to show and give confidence in the FFD algorithm.

2.4 METAHEURISTICS: GENETIC ALGORITHMS

A metaheuristic is [11] a solution method that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space.

In this thesis we will focus on Genetic Algorithms (GA) and will follow the work in [12]. Genetic algorithms try to work as natural selection and genetics. It is one of the most robust metaheuristics, being its only handicap that a proof for its convergence to an optimal solution has not been made until now, which puts it under the Simulated Annealing metaheuristic in a comparison. However, GA are very robust and tend to find a good, if not the best, solution for a problem without as many tuning needed as in Simulated Annealing. Genetic Algorithms try to find the best solution in $S_Q(x)$ by trying many different solutions and choosing in a smart way between them. A thorough explanation can be found in [12], but we will give here a summary and an example for the BPP.

In order for a Genetic Algorithm to work, we first must find a good way of *encoding* the parameters. Based upon that, we create a number

of different guesses (possible/feasible solutions). These guesses are random and from them we start creating more solutions. In each step we first do a *crossover* between the solutions from the last step. This crossover operator must be chosen accordingly to the encoding and will create new solutions. After this crossover we perform a *mutation* to the solutions created by the crossover. This mutation is thought to be necessary to guarantee that the GA will find an optimal solution, because it 'scrambles' the solutions and allows them to flee from a local optimum. When we have crossed and mutated the solutions, we study the *fitness* function (i. e. f_Q). and apply a *selection* operator that will select some of the solutions. From these solutions we create new ones by crossing, mutating and selecting again. We repeat this process until a number of iterations has been performed or until the fitness function does not improve during a certain number of iterations.

The problem with a Genetic Algorithm is that sometimes the crossing or mutation of a solution could give as a result a non-feasible solution. This could be solved with a good notation (not likely) or making the fitness function ∞ if a restriction is broken. Fortunately, for the BPP a smart way of avoiding unfeasible solutions has been found in this thesis, using a similar strategy to that of the Travelling Salesman Problem explained in [12, p.63].

Next, we present a Genetic Algorithm for solving the BPP in Listings 3, the G-First Fit.

Listing 3: Genetic First Fit

```
Input: α : ⟨s_1,...,s_n;T⟩
Notation: s = (s_1,...,s_n)
Output: A packing of the items s_1,...,s_n into bins of size T
0.1. Encode the solution by giving a permutation of n elements
    σ ∈ S_n.
0.2. Create 2^k possible random permutations.
1. Do
2.    Select the 2^{k-1} best solutions from applying FF to σ_i(s).
3.    Apply the crossing operator to 2^{k-2} pairs of solutions to
    generate 2^{k-2} pairs more.
4.    Mutate them.
5. N times
```

Listing 4: Crossing Operator

```
Input: Two permutation vectors σ_1, σ_2 ∈ S_n
Output: Two permutation vectors σ_3, σ_4 ∈ S_n
0. Inicialize σ_3 = σ_1, σ_4 = σ_2
1. Select two random numbers i < j
2. for l from i to j do
3.     σ_3(l) = σ_2(l), σ_4(l) = σ_1(l)
4. end for
5. Exchange elements that appear twice in σ_3, σ_4.
```

Listing 5: Mutation Operator

```
Input: A permutation vector σ
Output: A permutation vector σ
1. Swap two random elements of σ
```

The crossing operator seems a bit strange, but it is the same as the one used in [12, p.63]. Let us see an example of how it works. Note that $\sigma = (1432)$ means that $\sigma(1) = 1$, $\sigma(4) = 2$, $\sigma(2) = 4$ and $\sigma(3) = 3$:

Listing 6: Example of the crossing operator

```
σ1 = (2 3 1 4 6 5) = σ3
σ2 = (3 5 4 6 2 1) = σ4
i = 3, j = 4
σ3 = (2 3 4 6 6 5)
σ4 = (3 5 1 4 2 1)
These are not valid permutations!
6 is repeated in σ3, 1 is repeated in σ4, we exchange them
σ3 = (2 3 4 1 6 5)
σ4 = (3 5 6 4 2 1)
```

It is easy to see that with this method we will always end up having two correct permutations.

Theorem 2.4.1. *The crossing operator always returns two correct permutations.*

Proof. Let

$$u = (\sigma_1(1), \ldots, \sigma_1(i-1), \sigma_2(i), \ldots, \sigma_2(j), \sigma_1(j+1), \ldots, \sigma_1(n))$$

and

$$v = (\sigma_2(1), \ldots, \sigma_2(i-1), \sigma_1(i), \ldots, \sigma_1(j), \sigma_2(j+1), \ldots, \sigma_2(n))$$

If there is any pair $l \in I_{ij} = \{i, i+1, \ldots, j\}$, $r \in \{1, \ldots, n\} - I_{ij}$ such that $u_r = u_l$ then there is no number $m \in \{1, \ldots, n\}$ such that $v_m = u_r$ (because that number appeared once in σ_1 and once in σ_2). But having v n elements between 1 and n, this means that v has at least a pair of elements repeated (by the pigeonhole principle). One of the elements of the pair must be in I_{ij} (because σ_2 was a correct permutation), and again this element does not appear in u. Therefore there as many pairs of repetitions in u as in v, and if we exchange them we do not incur in a new repetition. □

Once we have proven that the method is correct, we must prove that it can lead to an optimal solution.

Theorem 2.4.2. *Let $\langle s_1, \ldots, s_n; T\rangle$ be an input instance of the BPP. Then, there is a permutation σ of the elements $(s_1, \ldots, s_n)$ such that the FF algorithm applied to the instance $(s_{\sigma(1)}, s_{\sigma(2)}, \ldots, s_{\sigma(n)})$ gives an output that uses a minimum number of bins.*

Proof. The BPP has an optimal solution, i. e.it attains its minimum because there is a finite number of possible solutions given an input instance. Let $B_1, \ldots, B_m$ be such an optimal solution, with the items divided in the following way:

$$\begin{aligned}
&\{a_{i_{11}} \ldots a_{i_{1n_1}}\} && \in B_1\\
&\vdots\\
&\{a_{i_{m1}} \ldots a_{i_{mn_m}}\} && \in B_m
\end{aligned}$$

Clearly $n_j > 0$ and $\sum_{j=1}^{m} n_j = n$ for all j. We create now a permutation σ in the following way:

$$\begin{aligned}
&\sigma(1) && = i_{11}\\
&\vdots\\
&\sigma(n_1) && = i_{1n_1}\\
&\sigma(n_1+1) && = i_{21}\\
&\vdots\\
&\sigma(n_1+n_2) && = i_{2n_2}\\
&\vdots\\
&\sigma(1+\sum_{j=1}^{m-1} n_j) && = i_{m1}\\
&\vdots\\
&\sigma(n) && = i_{mn_m}
\end{aligned}$$

We will now prove that, using the FF algorithm, for every $l \in \{1 \cdots n\}$, $a_{\sigma(l)}$ fits in the same bin as in the optimal solution. I. e., if $l \in (\sum_{j=1}^{k} n_j, \sum_{j=1}^{k+1} n_j]$ then $a_{\sigma(l)}$ fits in the bin $k+1$. We will prove it by strong induction.

First, it is clear that $a_{\sigma(1)}$ fits in the bin B_1: That bin is empty because $a_{\sigma(1)}$ is the first item to place. It is also clear that $a_{\sigma(l)}$ fit in the bin B_1 for all $l \leqslant n_1$: The optimal solution from where we come has the items $a_{\sigma(1)}, \ldots, a_{\sigma(n_1)}$ in the bin B_1.

Let us assume that for every item lesser than $r = \sum_{j=1}^{k} n_j + 1$, that item was able to be placed in its 'corresponding' bin, can the item $a_{\sigma(r)} = a_{n_{k+1}1}$ be placed in the bin B_{k+1}? The answer is clearly yes, because all the previous items fitted in their bins (which were before than B_{k+1}) and the FF algorithm will have placed them in the first

bin they fitted. Therefore when the algorithm FF tries to place $a_{\sigma(r)}$ it is clear that B_{k+1} is empty at that moment.

Let us assume now that for every item lesser than $r = \sum_{j=1}^{k} n_j + l$, $1 < l \leqslant n_{k+1}$ its corresponding bin had enough space to hold it. Because of the FF algorithm, this means that those items could not have been put in a bin posterior to that that corresponded to them in the optimal assignment. But this means that only the $l-1$ previous items admit the possibility to have been put into the bin B_{k+1}. Those items are $a_{(k+1)1}, \ldots, a_{(k+1)(l-1)}$ and the optimal solution had those items plus $a_{(k+1)l}$ (plus more), so there is at least enough space to put $a_{(k+1)l}$ in bin B_l with the FF algorithm (it could be placed in a bin $j < l$ of course).

This means that every item will be placed at most in its corresponding bin, and therefore no item will be placed in any bin $B_{m'}$ with $m' > m$. □

Therefore the GFF can achieve the optimal solution for the BPP if it 'hits' one of the correct permutations. Note that this does not alter the complexity of the problem, notwhitstanding that we cannot be sure that we have found the optimal solution unless we try them all. There are $n!$ possible permutations (which is less than the original $O(2^{n^2})$ feasible solutions), and for each solution such as the one used in the proof of Theorem 2.4.2, there are $m!n_1! \ldots n_m!$ possible permutations that give an optimal solution. It seems reasonable that the larger the elements are, the more bins an optimal solution will use and therefore the more probable will be to find an optimal permutation of the elements, but then again the smaller they are, the more elements will fit each bin and each n_i will be bigger, thus increasing the probability (furthermore it will be more likely than some items could be moved from bin to bin therefore adding new optimal solutions). A possible way to see if the algorithm GFF has found an optimal solution is to compare its outcome with that of a regular FF or a FFD algorithm. However, the 'optimal' average performance of the FFD makes it unrealistic to follow this guideline. Finally, it is important to note that the GFF algorithm is an offline algorithm.

With the proof of this theorem we conclude this chapter. We have presented a means to analyse the different decision and optimization problems and a very special problem called the Bin Packing Problem. We have presented many approximation algorithms for it and a whole family of algorithms called metaheuristics, from which we have centered in the Genetic Algorithms and provided a Genetic Algorithm for solving the offline BPP. The next chapter will relate the real world problem with the BPP with a mathematical model.

Part II

PRACTICAL WORK

The second part of the project consists in the practical work done in order to give advice on the development of strategies for the assignation problem with delaying. First a mathematical model will be procured and afterwards two heuristics and some improvements will be proposed and thoroughly studied. Finally, some guidelines and recommendations for future use will be posed as well.

3

MODELLING A DATA CENTER

INTRODUCTION

Giving a whole mathematical model of a DC and the processes that take place in it is an enormous task and it can mislead to unprofitable results. Therefore, some hypothesis must be made when modelling the DC in order to simplify the problem, while not losing information about itself.

The EP has a contract with the DC called Green Supply Demand Agreement or GreenSDA. In this contract there are "green clauses" such as:

- Maximum amount of power consumption reduction per request $\beta_{max} < 1$, e.g. $\beta_{max} = 0.1$ implies that the EP can ask a reduction in the power consumed of maximum 10% (it must be specified if it is with respect to the hired power or to the previously used power).
- Maximum number of energy consumption reduction requests per month.
- Rate of CO_2 emissions prediction.

Plus the "regular clauses", e.g.:

- Maximum power hired P_h.

These are the clauses important to this paper. For example, the DC will always have a prediction in the CO_2 emission factor for the next hour given by the EP, namely $CEF(t)$ (Carbon Emissions Factor as a function for the next hour). The EP will also provide $\beta(t)$, i.e. the reduction in power consumption requested for the next hour.

The DC network forms a graph V, each of its elements being a server. The set of all the users is $\mathcal{U}$ and the set of all possible services is $\mathcal{S}$. The set of all possible pairs user-service (allowed by each user's contract) is $\mathcal{D} \subseteq \mathcal{U} \times \mathcal{S}$.

At the same time, the ITC has a contract with the DC called Green Service Level Agreement, or GreenSLA. In this contract there are regular clauses (QoS, price per service, etc.) as in a regular SLA and "green clauses":

- Agreeded possible reduction in the QoS time-depending (for example, reducing the availability of a web server in weekends) and reward.

- Agreeded possible reduction in the QoS state-depending (for example, reducing the availability of a web server if the DC is in "Energy Saving State") and reward.
- Agreeded possible reduction in the QoS after negotiations, meaning that the DC can negotiate with the ITC for a reduction in some special moments (for example, if the EP has asked a reduction in the energy consumption) and reward/penalty.
- Agreeded maximum amount of times per month a DC can negotiate with an ITC for a reduction in the QoS.
- Agreeded maximum amount of times per month a DC can delay or pause the execution of a service or VM for an ITC u: $d_{max}(u)$.
- Agreeded maximum amount of delays a user can suffer given a particular state of the DC or moment, according to the GreenSLA, e.g. "Every weekend or when the DC is in Energy Saving State the user can see at most two of his services' executions delayed": $d_{max}(u, t)$

Each service in $\mathcal{D}$ will consume an instantaneous amount of CPU workload, memory usage and hard drive space. However, it will be summarized as a percentage of "computer usage": $0 < a_{(u,s)}(t) \leqslant 1$ is this average of the relative workload (relative to using 100% of a server) in the instant t. Each service will also have assigned a duration $T_{(u,s)}$. This duration will be also known in the offline version of the problem. $W(t) \in \mathcal{D}$ is the set of services that could be running in the instant t, i. e.it includes the services that are delayed.

A server that is online but idle will consume power, namely P_{idle}, and a server at maximum utilization will consume P_{MAX}. Moreover, the air conditioning (AC) consumes $P_{AC}(t)$. A service consuming $a_{(u,s)}$ "workload" will give a power consumption given by Eq. 2.

$$P_{service}(u, s)(t) = (P_{MAX} - P_{idle})a_{(u,s)}(t) \tag{2}$$

The DC can choose between two actions regarding the execution of a service:

- Run the service instantaneously.
- Pause or delay the execution of the service.

The objective is to choose the one that will result in less energy consumed and CO_2 emitted overall, i.e. at the end of the month/year.

A few more assumptions must be made from services:

- A service will always use less or equal than a server (no redundancy or services that require more than one server to be run). Otherwise, the heuristics proposed would not change significantly but the analysis of the algorithm would increase in complexity.

Table 2: Parameters involved in the All4Green System

Parameter	Defined by	Constraints
P_{idle}	Hardware	
P_{MAX}	Hardware	
Δt	Design	
P_{AC}	Variable	$T(i)$ limit and P_{max}
G	Geometry	
$CEF(i)$	GreenSDA / EP	
P_h	GreenSDA	
β_{max}	GreenSDA	
$\beta(i)$	EP	
$d_{max}(u)$	GreenSLA	GreenSLA
$d_{max}(u,i)$	GreenSLA / DC state	
$a_{(u,s)}$	Hardware / Design	$\leqslant 1$
T_{min}/T_{max}	Hardware	
ΔT_{max}	Hardware	

- Even in the online version, it will be known (or at least estimated) whether the service execution will end in the near time or not.

Finally, it is assumed that the consolidation of the services does not consume any extra energy, and a simplified equation for calculating the Temperature in the room in the slot i can be seen in Eq. 3

$$T(t) = G\int_{t-\Delta t}^{t} T(\tau)d\tau + c_1(P_{servers}(t)) - c_2 P_{AC}(t) \tag{3}$$

The terms in Eq. 3 are defined as follows.

$T(t)$ is the average temperature of the room in the instant t. The integral goes from $t-\Delta(t)$ to t (it resembles an average). $P_{servers}(t)$ is the power consumption of the servers in the instant t, $P_{AC}(t)$ is the power consumption of the AC in the instant t. c_1, c_2 are constants related to the temperature in the past. G is a geometrical constant depending on the structure of the room. There are intrinsecal boundaries to $T'(t)$ and $P'_{AC}(t)$ but we will not go deeper into them because the final model will be discrete.

The model of the temperature might as well be changed, this is only a simple version for academic purpose only. The restriction imposed by the DC structure is that $T_{min} \leqslant T(t) \leqslant T_{max}$ at all times.

A summary of the parameters involved in this chapter can be seen in Table 2.

About the temperature

The temperature model is a very simple and generalized model. It is clear that the temperature at each moment will depend on the servers open, the power given to the AC and the geometry of the room, and that it is a smooth function.

3.1 MATHEMATICAL MODEL OF THE PROBLEM

The objective is to minimize the total energy consumption and CO_2 emissions in a period of time. It can be a day or a month, let it be called T. We will have a weighting factor $CEF(t) > 0$ according to the carbon emission produced by a kWh consumed in t. Let $\delta_v^{(u,s)}(t)$ be a binary variable that indicates whether the service $(u,s) \in \mathcal{D}$ is running in the server $v \in V$ in the instant t. Let $O_v(t)$ be a binary variable that indicates whether the server $v \in V$ is open or not in the instant t. The formula for calculating the power consumption of the server v in the instant t is the one seen in Equation 4. The function f_0 that we try to minimize is shown in Equation 5.

$$P_v(t) = P_{idle} O_v(t) + (P_{MAX} - P_{idle}) \sum_{(u,s)\in\mathcal{D}} \delta_v^{(u,s)}(t) a_{(u,s)}(t) \quad (4)$$

$$\int_{t_0}^{t_f} (\sum_{v\in V} P_v(\tau) + P_{AC}(\tau)) CEF(\tau) d\tau \quad (5)$$

Before going deeper into this equation, we should realize that it is very difficult and more when we do not know which will be the function $CEF(t)$, or which and how many services will be requested ($P_v(t)$ is a steplike function). It is unpractical, and some simplifications must be made.

3.1.1 *Static Model*

First we will assume that $a_{(u,s)}$ is a fixed value for each pair $(u,s) \in \mathcal{D}$. Furthermore, all the services will arrive in the beginning and will never depart. We will not be able to delay any of them. Therefore in the instant t_0 we will receive n services that will use at most a server ($a_{(u,s)}$) and have to be placed into servers, altogether with deciding the $P_{AC}(t)$ function. Now it is obvious that, $\forall t$,

$$\sum_{v\in V} \sum_{(u,s)\in\mathcal{D}} d_v^{(u,s)}(t) a_{(u,s)} = \sum_{(u,s)\in\mathcal{D}} a_{(u,s)}$$

This means that there is a constant factor in f_0 that can not be minimized. Equation 6 shows the function to minimize now:

$$\int_{t_0}^{t_f} CEF(\tau)(\sum_{v \in V} O_v(\tau) + P_{AC}(\tau))d\tau \tag{6}$$

Note that the factor $\sum_{v \in V} O_v(t)$ will also not change in time in an optimal solution: If it changed it would be to add 1 or to substract 1 (or more) for some time T. The first case would contribute in $\int_{t_1}^{t_1+T} CEF(\tau)d\tau$ which would result in a non-optimal situation, and the second one would imply that this substraction could have been done earlier thus resulting in a smaller solution (remember $CEF(t) > 0$). This means that at the beginning the number of open servers would be fixed, and therefore the way to minimize $P_{AC}(t)$ will be to keep $T(t) = T_{max}$ during the whole time, leaving $P_{AC} = \frac{T_{max}(G\Delta t - 1) + c_1 P_{servers}}{c_2}$ (of course, c_1, c_2, G must be consistent in a way that P_{AC} must only depend on T_{max} and $P_{servers}$). This leaves us with only having to minimize ($\int_{t_0}^{t_f} CEF(\tau)d\tau$ is a constant) what can be seen in Equation 7.

$$\sum_{v \in V} O_v(t_0) \tag{7}$$

The restrictions are that each service must be in one and only one server and that no server can hold services that sum up to more than a 100% of the server capacity. This is clearly the Bin Packing Problem explained in the previous chapter. We will not go into further detail, as any offline algorithm that solves the BPP will solve this one as well. This model is, however, very simplified and therefore not realistic at all. We need to add the time factor.

3.1.2 *Discrete Dynamic Model*

We will try to simplify the first model seen in Equation 5 by giving a discretization of the time. We will divide the time in slots of duration Δt (the same that we use for calculating the temperature). We will name slot $i = t_0 + i\Delta t$. Therefore $O_v(t_0 + i\Delta t + t)$ will be a constant for all $t \in (0, \Delta t)$ and we will write $O_v(i)$. The same with $\delta_v^{(u,s)}(i)$, $P_{AC}(i)$, $CEF(i)$ and $W(i)$. We will still assume that $a_{(u,s)}$ is a constant. Therefore, the model would try to minimize Equation 8

$$\Delta t \sum_{i=0}^{N} (\sum_{v \in V} P_v(i) + P_{AC}(i))CEF(i) \tag{8}$$

Where $P_v(i)$ is given by Equation 9

$$P_v(i) = O_v(i) + \sum_{(u,s) \in \mathcal{D}} \delta_v^{(u,s)}(i) a_{(u,s)} \tag{9}$$

This is a simpler model, however it still assumes too much knowledge. Knowing that the objective is to reduce the overall energy consumption and Carbon Emissions of the DC, it is reasonable to assume that $N\Delta t$ is going to be big (in the order of weeks or months). And it is reasonable as well to assume that neither the EP nor the DC will have a monthly predition on the $CEF(t)$ function or the whole service request schedule. Furthermore, we are only able to delay the executions of some services at most 2 hours (and usually not that long). In order for this model to work, Δt should be smaller than that. Otherwise we cannot delay a service and expect $\delta_v^{(u,s)}$ to be constant at the same time. We will delay the service a time multiple of Δt. We will also assume that $CEF(i+l) = CEF(i+1) \forall l > 1, \forall i$, i. e. $CEF(i+1)$ will give an approximate idea of the trend of the carbon emission factor in the near future. In this case the function to minimize can be seen in Equation 10

$$CEF(i)(\sum_{v \in V} P_v(i) + P_{AC}(i)) + CEF(i+1)(\sum_{j=i+1}^{N} (\sum_{v \in V} P_v(j) + P_{AC}(j))) \tag{10}$$

Now, given the fact that all the future request events are constant, and that we can not delay them, we can remove them from the formula. But this is not totally realistic, as even delaying a service in the slot i can result in a total different service assignation in the future, and therefore more servers used in the future. Furthermore, if a service that was going to start in the slot i and finish in the slot N was delayed, the model would mislead to think that it is better to delay. To fix this we will use a new variable: $d_{(u,s)}(i)$ that will indicate if the service $(u,s) \in W(i)$ has been delayed or not. Also, a variable $d_u(i)$ indicating how many times has the user $u \in U$ been delayed. Now, on the one hand, Equation 11 holds for all $(u,s) \in W(i)$ and for all i, implying that a service is either assigned in the slot i or delayed. This implies that the second summand of P_v will become a constant once it has been added for all $v \in V$ and for all $j > i$. This allows to reduce the problem to Equation 12 (we have normalized by $CEF(i)P_{idle}$ without loss of generality).

$$\sum_{v \in V} \delta_v^{(u,s)}(i) + d_{(u,s)}(i) = 1 \tag{11}$$

$$\begin{aligned} &\sum_{v \in V} O_v(i) + \\ &+(CEF(i+1)-1)(\frac{P_{MAX}}{P_{idle}} - 1) \sum_{(u,s) \in W(i)} d_{(u,s)}(i) a_{(u,s)} + \\ &+CEF(i+1) \sum_{l=1}^{N} \sum_{v \in V} O_v(i+l) \end{aligned} \tag{12}$$

There is a big problem here and it is basically that we have no way of knowing how the delaying of services will affect the open servers in the future. Even if we restrict the maximum duration of the execution of a service to $N_{(u,s)}$, we still would have to add all the terms up to $max_{(u,s)\in W(i)}$ and we have no way of knowing the service requests of the future. Thus, apparently we are in a blind alley. However – and we will come to it later – there are assumptions we can make and different heuristics will come from there.

WHY HEURISTICS? The problem BPP is polynomial-time reducible to this problem, we only have to think that if $CEF(i)$ was constant, and there was no restriction on temperature and it was impossible to delay any service we would be in front of a BPP. But a BPP is $NP-hard$ and therefore so it is our problem (even the off-line version). Otherwise, if we could find a solution for our problem in polynomial time for any instance, we could find a solution in polynomial time for the BPP.

The restrictions are written in the following equations. Eq. 14 refers to the maximum workload of a server (it can not exceed 100%). Eq. 15 refers to the natural impossibility of running the same service more than once. Eq. 16 and Eq. 17 refer to the restrictions in temperature (see Eq. 13). Note that, in Equation 13, G, c_1 and c_2 are constants that depend on the previous value of the temperature. This means that, for example, if the temperature in the previous slot was different than the temperature in the exterior (T_{env}) and everything was shut down, then the temperature would tend to T_{env}. This coefficients are adaptive and should be calculated in each DC, however as it has been mentioned before, it is likely that each DC has its own temperature model depending on the power spent on the AC and the servers, and small modifications of the heuristics here proposed should still prove useful.

$$T(i) = G(T(i-1) - T_{env})T(i-1) + \\ + c_1(T(i-1)) \sum_{v\in V} P_v(i) - c_2(T(i-1))P_{AC}(i) \tag{13}$$

$$\sum_{(u,s)\in W(i)} \delta_v^{(u,s)}(i)a_{(u,s)} \leqslant O_v(i)\ \forall v \in V \tag{14}$$

$$d^{(u,s)}(i) + \sum_{v\in V} \delta_v^{(u,s)}(i) = 1\ \forall (u,s) \in A(i) \tag{15}$$

$$T_{min} \leqslant T(i) \leqslant T_{max} \tag{16}$$

$$T(i) - T(i-1) \leqslant \Delta T_{max} \tag{17}$$

More restrictions involving delays and power limits follow in Eq. 18, Eq. 19 and Eq. 20, and Eq. 21 restricts the number of consecutive delays of a service (it is written as an example, the number of consecutive allowed delays will be left to designers).

$$\sum_{\{s:(u,s)\in A(i)\}} d^{(u,s)}(i) + d^{u}(i) \leqslant d_{max}(u) \tag{18}$$

$$\sum_{\{s:(u,s)\in A(i)\}} d^{(u,s)}(i) \leqslant d_{max}(u,i) \tag{19}$$

$$\frac{E(i)}{\Delta t} \leqslant \beta(i) P_h \tag{20}$$

$$d^{(u,s)}(i-1) + d^{(u,s)}(i) \leqslant 1 \tag{21}$$

In this sense, the approach is obviously online, the immediate future is relevant because the model needs to know whether $CEF(i+1) > CEF(i)$ or not, or if there is any service about to end in the slot $i+1$, but it should be blind to what services requests will come in next slots and whether they will be delayed or not.

3.1.3 *Dealing with* P_{AC}

Regarding P_{AC}, clearly only three restrictions affect its value. On the one hand, the power limit (Eq. 20) limits the maximum power that can be spent in $P_{AC}(i)$ given the power spent on the servers. On the other hand, the temperature restrictions (Eqs. 16 and 17) limit the minimum power that can be spent in $P_{AC}(i)$ in order to keep the temperature between some limits.

If the approach were to be hollistic the problem faced would be treated as a Mixed Integer Programming (MIP) problem, with some binary variables and some real valued variables. These problems tend to be very complex.

However, we can study further this problem to see to what extent it requires MIP techniques.

First of all, notice that objective is to minimize the total energy consumption and therefore, it is to be expected that Eq. 20 is accomplished always, and if it does not follow, the heuristical approach will try to solve it. The only way to not being able to accomplish Eq. 20 is either by not delaying any service and having a very small $\beta(i)$ in the slot i, or delaying too many services and having this request in the slot $i+1$. Nevertheless if $\beta(i)$ or $\beta(i+1)$ are very small, it is reasonable to expect low or high values of $CEF(i+1)$, respectively, thus being improbable that many services will be not delayed or will be

delayed, respectively. The point is that the problem formulation and heuristics themselves should avoid solutions in which Eq. 20 does not follow.

Moreover, it is clear that $T(i)$ depends on all the previous temperatures, server power usage and P_{AC}. I. e., if we wanted to solve the problem formulated as in Equation 12, the only way to decide $P_{AC}(n)$ for all $n \geqslant i$ would be knowing all the service requests and assignations, and as we have said, this is impossible in this online approach. What we will do in order to try to solve the online problem is to approximate the function to minimize by only counting until $N = i + 1$. That is, we will see the problem in a 'myopic' way only taking into account the present slot and the inmediate future one. In layman terms, we are going to try to minimize the function $P_{AC}(i) + CEF(i+1)P_{AC}(i+1)$ (normalizing by $CEF(i)$), given the restrictions on power and temperature.

One can distinguish two major situations, between many others:

- $1 \geqslant CEF(i+1)$: Apparently the best solution is to delay as many services as possible. However, sometimes not delaying can be better (in terms of P_{AC} if, for example, it implies using less servers). Moreover, using less servers might change the strategy of AC usage.
- $1 < CEF(i+1)$: The inverse situation takes place. Again, the best solution is not always that which does not delay any service.

The heuristics proposed should take into account these and more different conclusions that can be reached from simple manipulation of the equations. In the next chapter the necessary conditions will reveal and be used in one or several heuristics, in order to give boundaries or at least a flair of what the competitive ratio of the algorithms would be.

Why the CEF?

One question that might come to our minds after all these presentations is: Why are we adding this CEF and weighting the power consumption by it? Why don't we just limit ourselves to try to minimize the overall energy consumption?

On one hand, solving the problem for any CEF, we can solve it for a constant $CEF = 1$ which would minimize the overall energy consumption. On the other hand, the CEF accounts for the collaboration between the DC and the EP as much as the delays account for the collaboration between the DC and the ITC. If we only try to minimize energy consumption from the DC, we can fall into delaying services and using, for example, coal fuelled energy in the future thus sacrificing the collaboration.

4

PROPOSED HEURISTICS AND RESULTS

4.1 SHOULD WE ALLOW CONSOLIDATION?

In this section we are going to study the consequences of allowing consolidation vs. not allowing consolidation of services.

We will begin by the 'easiest' configuration possible: We are allowed to migrate as many services as we want each time a service has to be assigned or a service is retired from duty. Note that if no delaying was allowed, this would be the Fully Dynamic BPP adding the AC, therefore the Fully Dynamic BPP can be considered a subproblem of this one. This means that this problem is $NP-hard$ and therefore we should not expect a polynomial time algorithm that solves the problem, even in its offline version. The Fully Dynamic BPP has an asyntotic approximation ratio of $\frac{5}{4}$ while the offline BPP has an asyntotic approximation ratio of $\frac{11}{9}$ with FFD algorithm, which is a little better.

If the configuration is such that does not allow us to consolidate services, then we are, proceeding with a similar reason as before, in front of a variation of the Dynamic BPP, which has no better approximation ratio than 2.5, while an online algorithm without repacking but without departures will have a ratio of approximately 1.6.

These results suggest that in general it will be better to allow consolidation than not allowing it. Let us focus now on another keystone of Equation 12: Should we assume that delaying a service will result in using one server more 'later' or not?

Let us reason intuitively as follows: Given a Poisson distribution of service requests and an exponential discrete distribution of service durations (with memory, because we can know for how long a service has been running in the slot i), how can a service be actually using one more server in the future? An example would be (for both options) as follows:

A service that last two slots arrives in the slot i and fits into an already open server. However we decide to delay it. But all the services that were running in the slot i end in the slot $i+1$, so this service will be using a server for itself in the slot $i+2$, and we could have avoided that if we had placed the service in the slot i in the first place. Now, intuitively speaking this is very difficult to happen but it can happen nevertheless that we decide to delay a service s and place it in the slot $i+1$ in the server v, but all the services that are running in the server v will stop before our service. And if we can not repack and consolidate, this is quite likely to happen and therefore

we will have a situation in which we will use one more server more often. This gives us a new insight when solving the problem and adds empirical evidence (it is not a proof) of the benefit that comes from being able to consolidate and repack the services.

4.2 HEURISTICS PROPOSED

We need to define heuristics and analyze them. The heuristics here proposed have the following structure:

1. Decide whether to delay or not one/a group of/all services, estimating whether delaying or not will be better with the knowledge present at the moment.
2. Assign the services in such a way that optimality (or the nearest form) is achieved.
3. Calculate $P_{AC}(i), P_{AC}(i+1)$ minimizing the cost function explained at the end of the last chapter.

These steps can be clearly separated or interleaved between them, but in any case they allow us to separate the big problem into smaller, easier problems. The critical steps are the first and the third. The second step is always a BPP (one of its variants – online, offline, dynamic, fully dynamic).

The heuristics will come from different interpretations of the policies proposed by A4G, see Table 1. We will understand *anticipate* as the following: If we have a request in the slot $i+1$, we will have the possibility to put it now, thus considering it the same as having a service request in the slot i. On the other hand, we will add some special services (antivirus programs, updates) not as a request and neither being able to delay them, this is *extra services*. We can *pause* them if the conditions change. We will understand *postpone* as delaying or pausing some service executions.

The definition of three different states of the Table 1 seems to suggest that only three different values for $CEF(i+1)$ may be possible. However this is not completely true, because (citation may be required) the EP can be in more states, depending on the mixture of different energy sources it uses, and can send as information different $CEF(i+1)$ that will give us information about whether it is better to delay a service even if it requires using more servers later or not.

4.2.1 *Strict heuristic*

The first heuristic proposed, H_1, will follow strictly the policies mentioned in the Table 1. Thus, we will distinguish three cases:

1. $CEF(i+1) < 1$: This means that the near future is expected to be better in terms of CO_2 emissions and DC State than the

present. We will postpone as many and as big services as we can, i. e.if a user can have n services delayed in the slot i and has $m > n$ services susceptible to postponing in the slot i, we will postpone the services that consume more energy. In a similar fashion, we will pause the extra services that we might have started previously. We will place all the services that we must put in the slot i and the slot $i+1$ using a fully dynamic BPP algorithm or an offline BPP algorithm.

2. $CEF(i+1) = 1$: This means that we do not expect the near future emissions or state to change much from the present. We will not postpone any service nor anticipate any. We will not start any extra service nor pause any.

3. $CEF(i+1) > 1$: This means that the near future is expected to be worse than the present. We will try to anticipate as many services as we can and start extra services.

After delaying and assigning the services, as if no new petitions were to be made in the slot $i+1$ we will calculate the P_{AC} finding the optimal values of $P_{AC}(i, i+1)$ given that assignment. This can be done by brute force, considering that P_{AC} has some limits given by the fabricant. This heuristic is called H_1. It is the first heuristic that comes to mind. It is clearly not the best and we will enumerate possible problems that it may pose.

- Some times it is better to not delay or delay a service if it results in using one server less.

- Some times it is better to not delay or delay a service even if it does not result in using one server less because the P_{AC} combination can change and the overall function be smaller. This is because some times we can use more power in the AC in the slot i, thus reducing the temperature and being able to use less power in the slot $i+1$.

- The limit on delays can be tight, thus leaving very little options at the end of the month if all delays have been used already.

- If CEF is constant, this heuristic will not attempt to delay or anticipate any service at all, thus not solving the problem of minimizing total energy consumption.

For the aforementioned reasons, we can not expect a very good outcome from this heuristic.

More precisely, imagine the following chain of events:

We have N servers full up to $\frac{1}{2} - \epsilon$ of their total capacity. The servers are about to finish in a slot. N services that use $\frac{1}{2} + \epsilon$ arrive that will last only one slot, but $CEF(i+1) < 1$ and therefore using the

heuristic we will delay them. Assuming there is no other income in the servers, the result of the heuristic would be as seen in Equation 22, while the optimal result would be N. We are not calculating the AC here, although it would improve a little bit the result, in order to understand fully the problem of the heuristic. As can be seen, the result can be as bad as twice the optimal (if $CEF(i+1) \to 1$). From now on for conciseness we will have $\alpha = \frac{P_{max}}{P_{idle}} > 1$. However as $CEF(i+1)$ is smaller it is clear that the heuristic would give an optimal solution. The limit would be in $CEF(i+1) = \frac{(\alpha-1)\frac{1+2\epsilon}{2}}{1+(\alpha-1)\frac{1+2\epsilon}{2}}$. For example if we accept that $\alpha \approx 1.42$ ($P_{idle} \approx 0.7P_{max}$ then $CEF(i+1) > 0.3$ would guarantee that the solution given by the heuristic is worse than the optimal solution.

$$N(1+CEF(i+1)) - (\frac{P_{max}}{P_{idle}} - 1)(1-CEF(i+1))\frac{N(1+2\epsilon)}{2} \quad (22)$$

Can we find a situation in which H_1 is worse than the optimal no matter what $CEF(i+1)$? Not exactly, but we can find a situation in which $CEF(i+1) = 1$ and the solution is far from optimal. Imagine there are N servers full with two classes of services, one that fills $\frac{1}{2}+\epsilon$ and the other that fills $\frac{1}{2}-\epsilon$. The services that fill $\frac{1}{2}+\epsilon$ will be gone in the slot $i+1$ and the other services still need m slots to finish. In the slot i, N services arrive that use $\frac{1}{2}+\epsilon$ and need $m-1$ slots to finish. We don't delay any service and therefore we use 2N servers in the slot i and 2N servers in the slots $i+1,\ldots,i+m-1$ and N servers in the slot $i+m$ if we do not allow consolidation, or N servers from the slot $i+1$ until the slot $i+m$ if we do. If we delayed we would have used only N servers from the slot i until the slot $i+m$. This leads to an approximation ratio of $\frac{2m-1}{m+1}$ (it can be as big as we want) if we can not consolidate or $\frac{2+m}{1+m}$ (reaches a maximum of 1.5 when $m = 1$) if we can. Therefore H_1 has an unbounded approximation ratio if we are not allowed to consolidate, and an approximation ratio of at least 1.5 if we are allowed to consolidate.

4.2.2 *Dynamic Heuristic*

The previous reasoning leads us to a second heuristic H_2: For each service we compare the weighted power function $P_{serv}(i) + CEF(i+1)P_{serv}(i+1)$ delaying (if we can) and not delaying the service, and we choose the best option. This heuristic brings the problem of deciding whether delaying a particular service might affect negatively or positively, it really can not be known, so some empirical improvements may be devised:

- If we allow to consolidate, it is to be expected that delaying will not incur in using one more server later. If we don't consolidate, it is the opposite way. For example we could consider that the

probability of incurring in using one more server later is p_{cons} and p_{ncons} for each case, and calculate the mean value.

- If it is better to delay even if we incur in using one more server, then we will delay. If it is better not to delay even if we do not use one more server, then we will not delay. For the other situation, we can randomize the algorithm and decide to delay or not with probability $1 - p_{cons}$ and $1 - p_{ncons}$.

This algorithm is more complex and looks better. It is less probable that Eq. 20 does not follow. However, it is not perfect, and it has similar problems to the offline algorithm First Fit. The order matters. For example, if $CEF(i+1) < 1$, imagine there are N servers with $\epsilon < \frac{1}{2}$ space. Then 2N services arrive that will only last 1 slot. The first N have size ϵ, the other $1 - \epsilon$. The servers are going to be all free in the slot $i+1$. For the first N services, H_2 decides to place them now, if a condition on $CEF(i+1)$ holds: $CEF(i+1) > \frac{1}{\frac{P_{idle}}{\epsilon P_0}+1}$ (Where $P_0 = P_{max} - P_{idle}$). For the second N services, it is better to put them later. However the optimal solution is to delay them all. In the worst case situation, namely $\epsilon \leftrightarrow \frac{1}{2}$ and $C \leftrightarrow \frac{1}{\frac{P_{idle}}{\epsilon P_0}+1}$, the ratio of the two solutions is the one displayed in Eq. 23. As can be seen, it is smaller than the ratio in H_1 as long as $P_{MAX} < 2P_{idle}$. Notice that in this scenario, H_1 would give a better performance.

$$1 + \frac{\frac{P_{MAX}}{P_{idle}} - 1}{2} \tag{23}$$

We have seen that it is more difficult to find worse-case situations for H_2 than for H_1. We have seen however that H_1 can be better than H_2 in some situations and H_2 can be better than H_1 in some other situations.

Here are some proposals for other heuristics before we enter in a qualitative discussion between the two first (that have been studied more thoroughly)

- Sorting the items that can be delayed (or anticipated) in H_2 before actually choosing in which slot can we put them should increase the performance of H_2.
- Calculating P_{AC} in each iteration of H_2 can also increase the performance of H_2 (this way we would avoid situations in which delaying a service even using one more server can be useful because then we can use more P_{AC} to lower the temperature, etc.).
- Instead of decide on delaying all services or just one by one, we could pack the services (by size, by sum of sizes) and decide to delay or not these packs (in a similar fashion to the harmonic algorithm for the BPP).

As for the competitive ratio, there are also some observations to be made:

- Due to the fact that there is a limit to the possibility of delaying services, one must choose wisely when to delay them in order to save as much as possible. However, deciding to delay only when a particular $CEF(i+1)$ is holding, might lead to problems (because it might never happen during a month).
- Choosing to delay the services too early might result in not having possible delays in the end, while waiting for too long might cause to not delay a service when it is needed. A good recommendation would be to try to limit the number of delays per day, ensuring that every day ($CEF(t)$ is easier to predict each day) we can delay some services yet we do not burn our boats to soon.

4.3 Simulations and visual demonstrations

4.3.1 *Simulating* $CEF(t)$ *and requests*

Unfortunately, for this project no reliable data on the overall carbon emissions coefficient from an EP has been given. The same has happened with a statistic of the service requests given by the users. Therefore, we have to try to implement a few realistical scenarios that include some of the worst-case situations as well.

What is a worst-case scenario in terms of $CEF(t)$ and requests? Well, if there are many requests but $CEF(t)$ is very low, then we have no problem, we can even anticipate services if we have information, or advise, about the future. On the other hand if $CEF(t)$ is very high but there is a very low demand of services (or running services at the moment), we can just pause whatever service we have to pause and keep on going with the others. But if we have a huge amount of requests and running services *and* at the same time $CEF(t)$ has a very high value, then we are in problems. The worst of them when we are peaking in both $CEF(t)$ and requests.

For the sake of simplicity we have assumed that $CEF(t)$ is a smooth, daily periodical function that has one, two or three local maxima per period. We are not adding any randomness for two reasons: First, it is unlikely that the EP will account for small perturbations in the value of $CEF(t)$. Second, the heuristics proposed are weak in front of small perturbations (and we will propose at the end of the section some improvements). Concretely, we will interpolate using cubic splines a series of data given at certain hours, and we will have the two different graphs for two days that can be seen in Figure 5.

On the other hand, the assumption for the service requests is a bit different. We assume that each half an hour of each working day

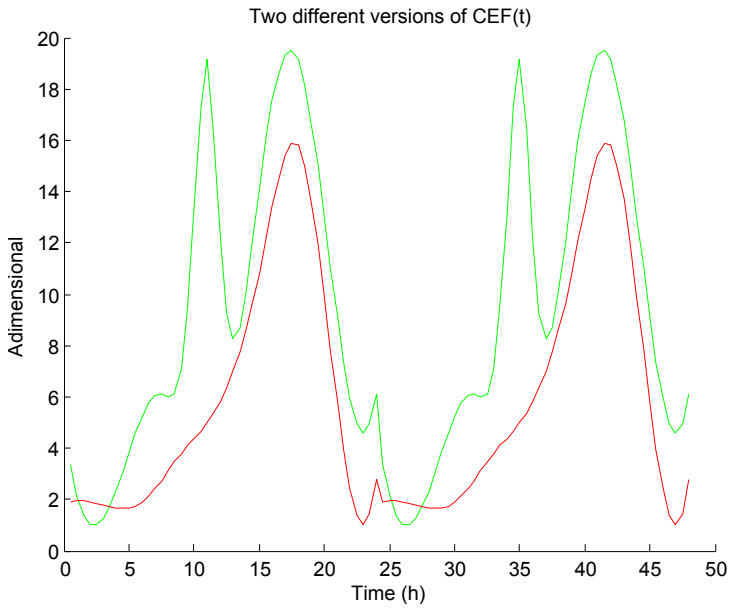

Figure 5: Two different versions of CEF used

(in reality it would change, as well as CEF, for festive days) receives service requests following a Poisson distribution. We generate then random data following this distribution. A result for two days can be found in Figure 6.

These are the conditions for our future simulations. Probabilistic study has been made regarding the different distribution of arrivals for the processes, and we will not go into detail with that.

4.3.2 *Consolidation is important*

We already know that, but let us show some related results. In a MATLAB simulation we will compare the energy consumption (unweighted) using a FF (no consolidation) algorithm for a dynamic BPP and using a FFD (thus migrating every single service in each slot). The code for the simulations can be found in an attached file. We do not include P_{AC} in the calculations, in order to show clearly the differences between the two methods. We can not compare with the optimal algorithm for obvious reasons, but as has been commented in Chapter 2, we can assume that the FFD algorithm will give in average an optimal solution.

Figure 7 shows us the difference when every 30 minutes arrive 10 services with a size uniformly distributed between 0.1 and 0.9 that last at most 12 hours (the duration of a service is defined by a geometric distribution of mean 6 that is truncated in 12). The simulation represents 48 hours to reach a stable situation. We assume P_{idle} to be 115W, using the results in [13] and $P_{idle} = 0.7P_{max}$. In this situa-

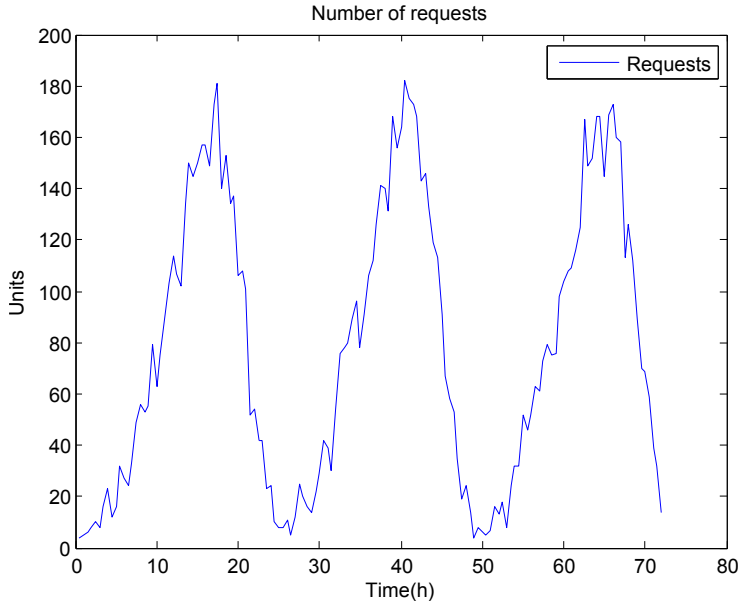

Figure 6: Poisson arrivals, regard the "period" (daily)

tion, but extending the simulation to 100 days, the FFD was an 11.8% better than the FF on average.

Now, assuming that the services come in a Poisson distribution, we simulate three days and we obtain Figure 8 where we can see both FF, FFD results and the number of requests. In this situation, the FFD also improves the FF algorithm in a 11.2% on average.

How does affect the duration of the services to this improvement? Some Data Centers put a limit on the number of hours of execution of a VM. For example, UPC's Virtual Machines run for about 2 hours, and usually less. At first glance one can understand that the power consumption will be smoother the longer the services last. For example, in Figure 9 we show the difference between having services that last 6 hours on average and services that last 2 hours on average, always using a FFD algorithm. In Figure 10 we see how the improvement changes when we change the average time of the service. This is relevant because the DC can choose a limit for the duration of the services they offer. As we can see, the improvement of the FFD versus the FF goes in average around a 12% except when the services last around $1-2$ hours, which can be quite the common case.

Studying these differences we would see that they lie on using more servers, which is clear because the services are the same always both for FFD and FF. It is only fair to assume that, including the delaying to the mix, the differences would be greater. And it is indeed the case here, as will be seen later.

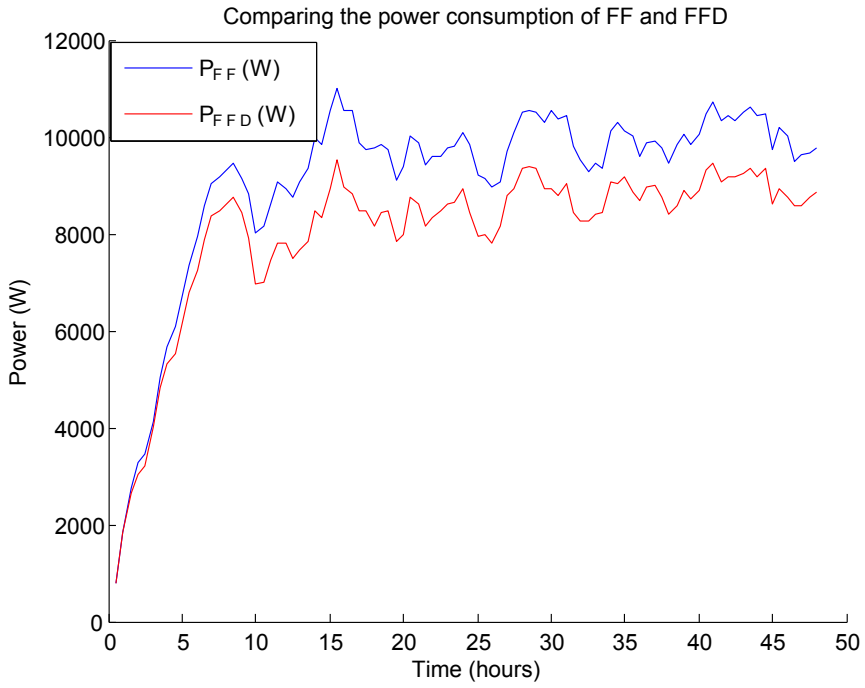

Figure 7: FF vs. FFD, note the peaks in FF

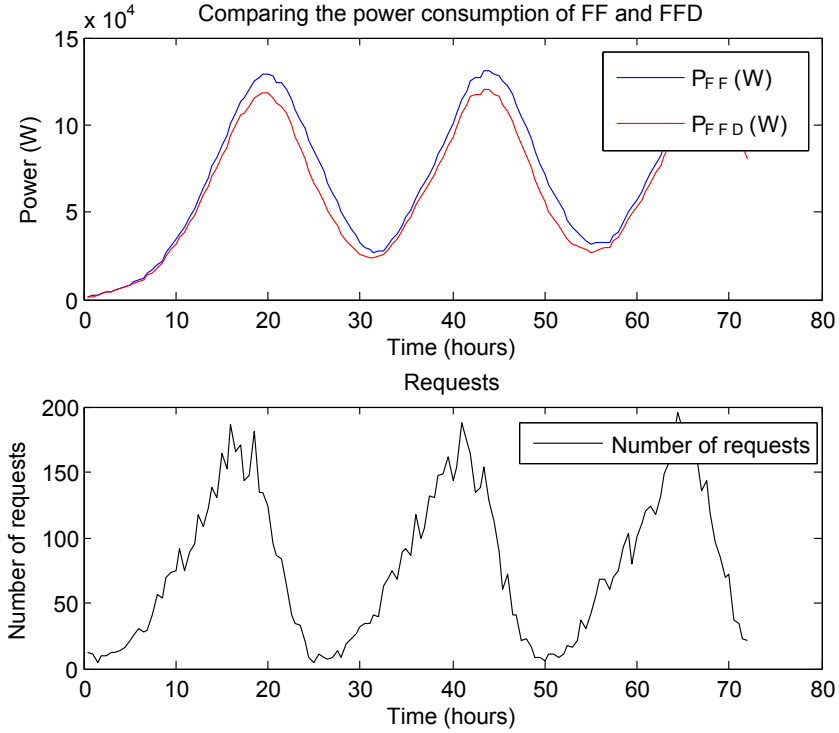

Figure 8: FF vs. FFD, Poisson requests

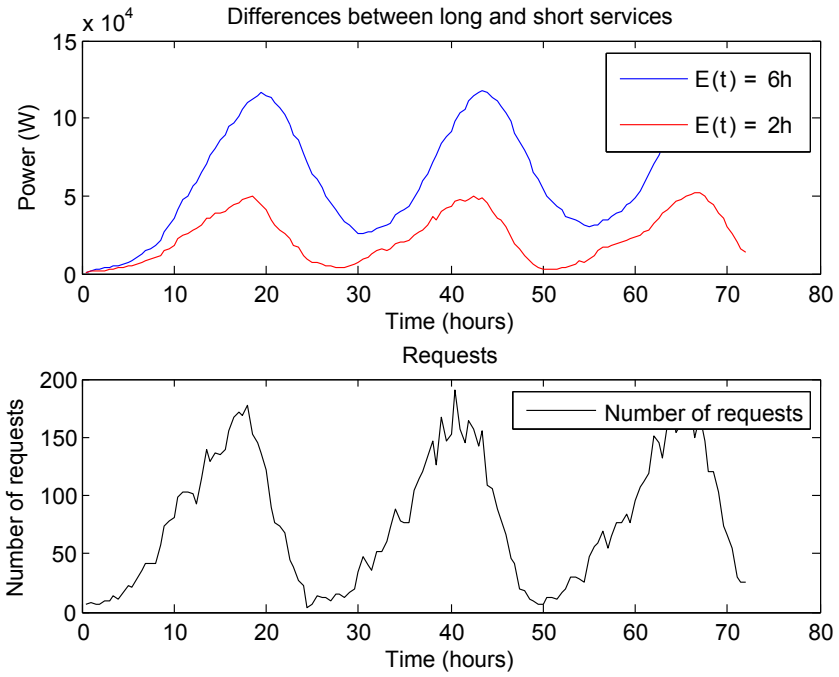

Figure 9: Notice that, apart from being higher, P_{FFD} is also smoother when $E(t) = 6h$

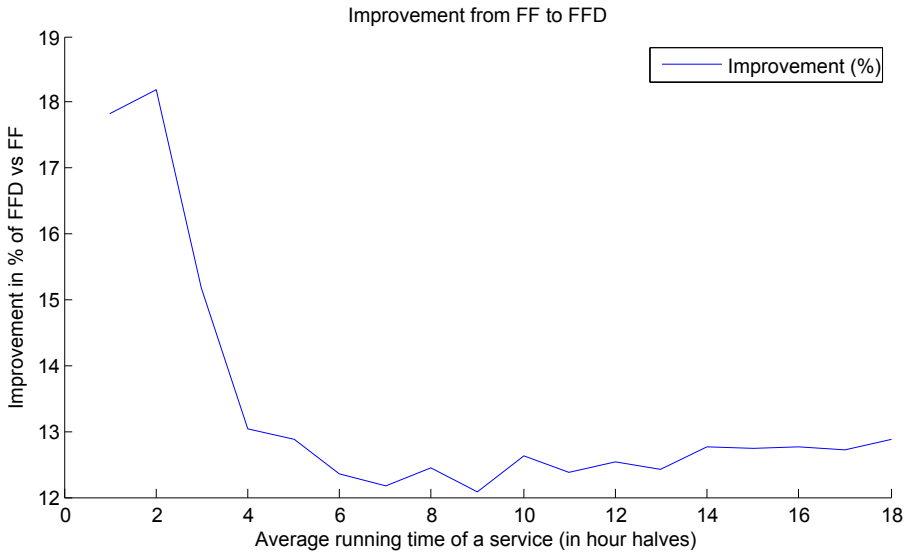

Figure 10: Notice that for short duration processes the difference is bigger

4.3.3 *Delaying the services*

Following the guidelines in the previous sections, we will limit the number of delays per day, thus making it possible to assume that every time a service request arrives, there is a fixed probability P_d that it can be delayed. In these first simulations we will not allow for anticipation of any kind of service, nor we will include calculations for $P_A C$.

The first heuristic and second heuristic follow a different strategy when it comes to delay services. The first heuristic does not delay any service until the prediction for CEF starts to lower, while the second heuristic is constantly checking if delaying a service can locally improve the result. We have already discussed some worst-case situations for both heuristics, however let us do a qualitative probabilistic analysis.

If the services were to last 1 slot at most, the second heuristic would be indubitably better, because we would never have the problem of using less servers now but more later. So all we have to do is check whether delaying the service will help or not and do it. Of course we are not pretending this will be an optimal solution: An optimal solution should check all the subsets of delayable services and select the subset that gives a better solution in the slots $i, i+1$. But the thing is that each slot is independent, i. e.services that start in slot i finish in slot i, so the optimal solution for the slots $i, i+1$ would give, in this case, an optimal solution as well for all the slots. The first heuristic would lose a lot of opportunities to delay services thus will be probably worse. This is confirmed by simulations (the graphs are not very clearly different), where the average improvement given by the first heuristic is roughly a 0.5% and the average improvement given by the second heuristic is around 0.6%. In these simulations there is no advise on the future requests and therefore the second heuristic is not performing as well as it could, but still it is better than the first heuristic as we wanted to show. We are doing all the simulations allowing for consolidation of services.

Now, as the services last longer, the first heuristic starts to improve considerably for the following reason: If each slot arrive on average $\lambda(i)$ services, that last on average m slots, we can expect that at the slot $i+m$ we will have $\sum_{j=i}^{m} \lambda(j)$ services, of which a $100P_d\%$ can be delayed. This means that more services can be delayed and the first heuristic will delay them all when the opportunity rises, while the second heuristic not only will have already delayed a few in the past but delaying locally one by one service will show smaller improvement (because the other services will still be adding up to the total power consumption). In layman terms, the more services we can delay, the bigger the "packs" of services to delay.

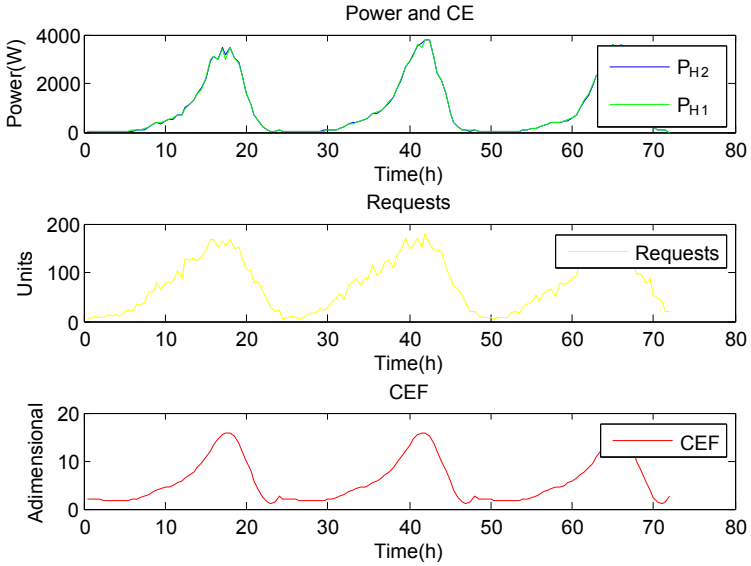

Figure 11: 2 slots, H1 is a 0.27% better than H2

So what the first heuristic does is trivially select all the services that can be delayed and delay them, while the second heuristic tries all the services one by one and decides whether to delay that one or not. The optimal solution should be in the middle, delaying a subset of all the delayable services. But, as a first attempt, we see that the first heuristic behaves better than the second heuristic, at least under the conditions given in the simulation[1], when the services last long enough. As an example we will see Figures 11, 12 and 13. We can see in Figure 14 that if we can delay more services ($P_d = 0.4$), then the difference is even larger. It can be also observed in Figure 15 that before the peak H2 is already delaying services, which slightly improves the energy consumption but prevents the big reduction in the peak (note that the first heuristic gives the same result as FFD until it delays services). Note also the difference between being able to consolidate or not in Figure 16, in both cases consolidating leads to an 11% of improvement to not consolidating. All in all, it seems that the second heuristic is more robust and smoothens better the function of total energy, but we are not interested in smoothing it.

4.3.4 *Anticipating service execution*

Now we add the possibility to anticipate service executions from services that will come in the slot $i+1$.We assume that we can anticipate all of them and we know all that will come. It is a bald assumption

1 Note that the service requests are not as relevant as the CEF function, because the delaying decision does not depend on the requests, and the assigning algorithm (which depends on it) is solving a simple BPP. That is the reason why two different CEF functions have been given, in both of them a peak coinciding with a peak in requests

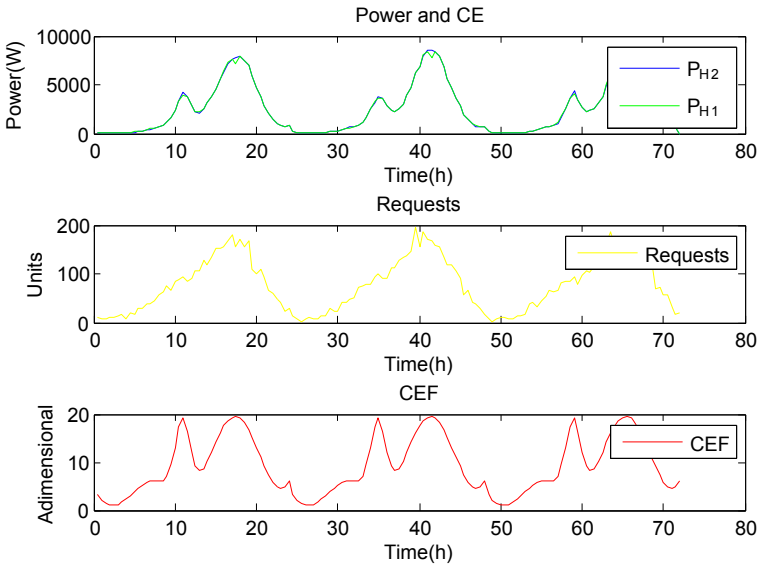

Figure 12: 4 slots, H1 is a 0.37% better than H2

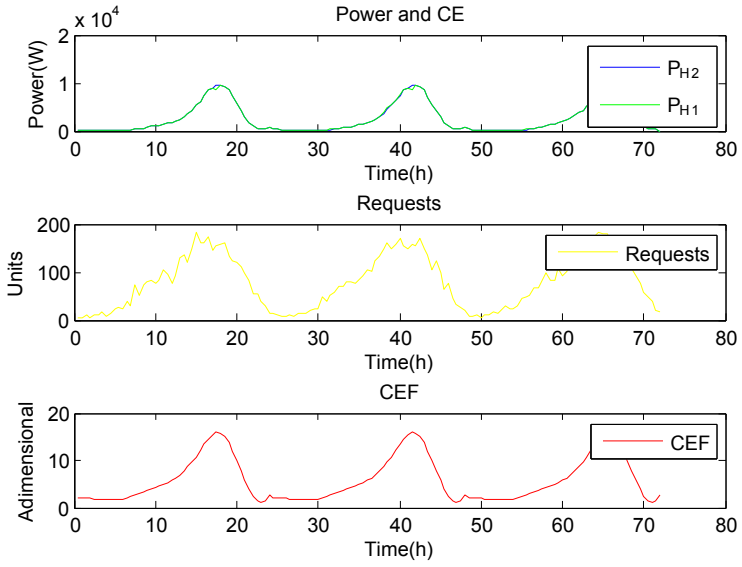

Figure 13: 6 slots, H1 is a 0.42% better than H2

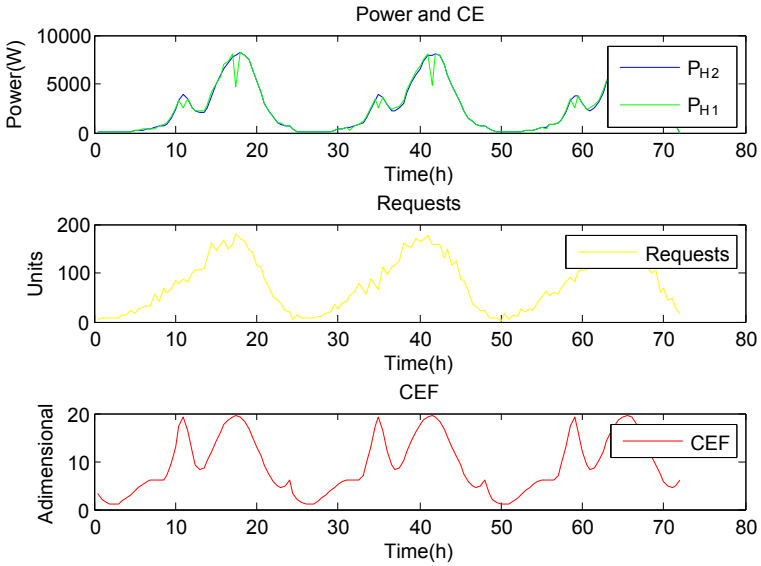

Figure 14: 4 slots, $P_d = 0.4$ instead of 0.1

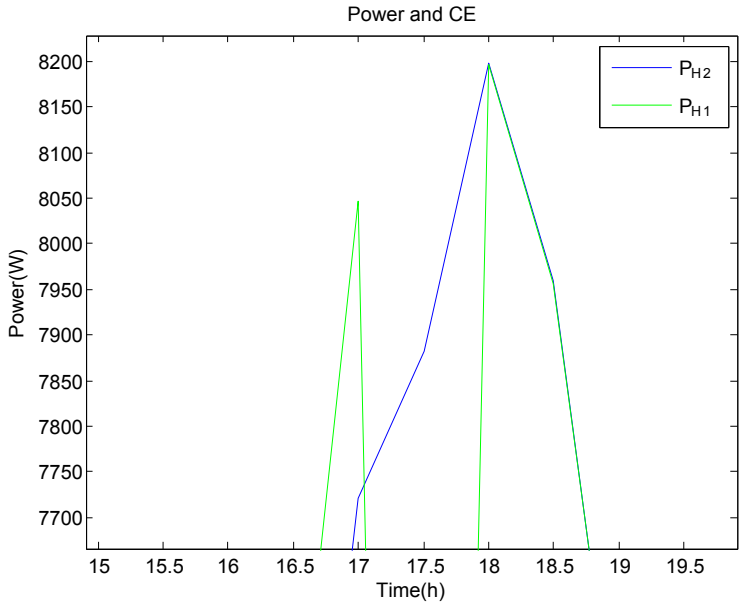

Figure 15: 4 slots, note the big leap that H1 gives

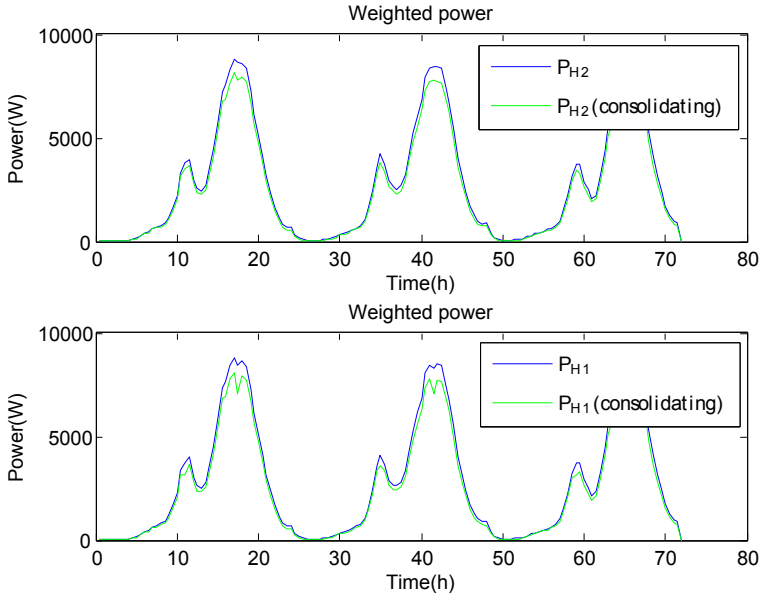

Figure 16: Consolidation vs. non-consolidation in both H1 and H2

but it is the opposite extreme to the previous section. In anticipating the execution of a service we don't increase the times the service may be delayed. Amazingly, the ability to anticipate services is increasing the performance ten times in the case of the first heuristic, to a 5% when the probability of having a delayable service is 0.1. But in the case of the second heuristic it makes almost no difference. This is because the second heuristic treats the anticipating possibility as adding the services now and deciding whether to delay or not them, one by one. And because it is, ironically, more myopic than heuristic 1. The cornerstone is that the first heuristic will anticipate as many services as it can, thus reducing the impact on the comeback after the delaying, as can be seen in Figure 17. By doing so it might use more weighted power than a simple FFD or than the second heuristic in the slots where it anticipates, but the overall consequence is reducing the total weighted energy consumption. It is at first surprising that the results change so drastically between being able to anticipate and not being able to, and that the difference is so big between the first and second heuristic, but yet we will try to explain it later.

4.3.5 *Air Conditionate power*

First of all, we must remember that finding the optimal solution to the problem is not the same as finding the optimal solution to the delay+assignment problem and then the optimal solution to the AC problem given such an assignment: It can be that we use more server weighted power but less AC weighted power and the sum is lower. However, there is an interesting result in Lemma 4.3.1, that says that if by any chance we found an assignment that lead to the optimal

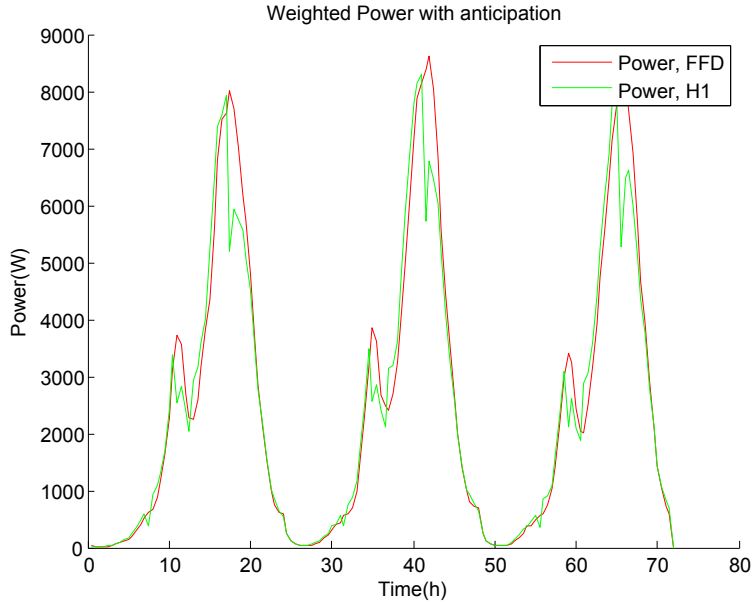

Figure 17: Adding anticipation, note that we use more energy when CEF increases

overall solution, then finding the optimal P_{AC} for that assignment means finding an optimal overall solution.

Lemma 4.3.1. *Given an input instance of the problem for two slots* i *and* $i+1$*, if the optimal solution is such as* $B_1, \ldots, B_m, P_{AC}(i), P_{AC}(i+1)$*, then* $P_{AC}(i), P_{AC}(i+1)$ *is an optimal solution for a problem in which* $B_1, \ldots, B_m$ *were already given to us.*

Proof. It is inmediate, if P^*_{AC} is a different solution, then the combination $B_1, \ldots, B_m, P^*_{AC}(i), P^*_{AC}(i+1)$ is a different solution for the initial problem, using at least as much weighted power as the original one. □

Furthermore, if we reduce our scope to the slots $i, i+1$, then, disregarding the limit on the maximum power (because P_{AC} is already limited between 2300 and 3489 Watts), given a fixed $P_{serv}(i)$ and $P_{serv}(i+1)$, whatever it is the optimal value for $P_{AC}(i)$ and $P_{AC}(i+1)$, it must be that $P_{AC}(i+1)$ ensures that the temperature is T_{max} [2] (see Lemma 4.3.2). This means that we can express $P_{AC}(i+1)$ as a function of T_{max}, $P_{servers}$ and $P_{AC}(i)$. This $P_{AC}(i+1)$ is only useful, of course, to make sure that the solution is feasible for the future, but it will be changed in the next slot.

Lemma 4.3.2. *Let* Q *be the problem with inputs* $CEF(i)$, $CEF(i+1)$, $T(i-1)$, T_{min}, T_{max} *and the coefficients needed to calculate* $T(i)$ *(that depend on* $T(i-1)$*). We want to minimize*

$$CEF(i)P_{AC}(i) + CEF(i+1)P_{AC}(i+1)$$

2 Or 2300 Watts if no matter what, the temperature is already lower than T_{max}

subject to[3]

$$T_min \leqslant GT(i-1) + c_1 P_{serv}(i) - c_2 P_{AC}(i) = T(i) \leqslant T_{max}$$

$$T_min \leqslant GT(i) + c_1 P_{serv}(i+1) - c_2 P_{AC}(i+1) \leqslant T_{max}$$

Then, whatever the optimal solution is, we have that

$$P_{AC}(i+1) = \frac{GT(i) + c_1 P_{serv}(i+1) - T_{max}}{c_2}$$

Proof. It is also simple, if $P_{AC}(i+1)$ was bigger than that, the function would be bigger and we would be below T_{max}, we could lower $P_{AC}(i+1)$. If it was lower than that, the solution would not be feasible. $P_{AC}(i+1)$ does not affect the restriction on $P_{AC}(i)$ and therefore we can change it freely without affecting $P_{AC}(i)$. □

The only thing left to know is how to calculate $T(i)$. As has been said in the first chapters, this is only for academical purposes and each DC should have its own formula, the results given here should not change substantially.

In our case, we approximate the coefficients the following way: $G(T(i-1)) = \sqrt[3]{\frac{T_{env}}{T(i-1)}}$. This means that if no server or AC was operating, the temperature would stabilize at an 'environment' temperature given by us (we will use 25ºC) in one hour. For c_1 and c_2 we will not use formulas but the following approximation:

- For c_1, we will assume that a DC will go from T_{min} up to T_{max} in half an hour if AC is closed and 20 servers are working at a 100%. We will assume that it will go from T_{env} up to $1.2T_{max}$ in the same time in the same situation, and it will go from T_{max} up to $1.4T_{max}$ in the same conditions.

- For c_2 we will do the opposite. The DC will go from T_{max} to T_{min} in half an hour if AC is consuming $3000W$ and no servers are on, from T_{env} to $0.8T_{min}$ and from T_{min} to $0.6T_{min}$ in the same conditions.

Rough as this approximation is, it has proven to be both useful and fast when doing calculations. In order to find the best solution we simply try them all for the range of possible values of P_{AC} (increasing a parameter Δ), and take the value of P_{AC} that gives the minimum weighted power. We show a couple of examples in Figures 18, 19. We only show the examples for the first heuristic because it is the one that has proven to be better as we have been seeing. The temperature in both cases does not go far beyond or above T_{env} due to the way we have defined the coefficients.

As a final remark, in Figure 20 we can see how the first heuristic improves both FFD and FF with respect to the probability of having a service delayed P_d.

3 This is not a linear problem! c_2 might depend on $T(i-1)$ and thus on $P_{AC}(i-1)$

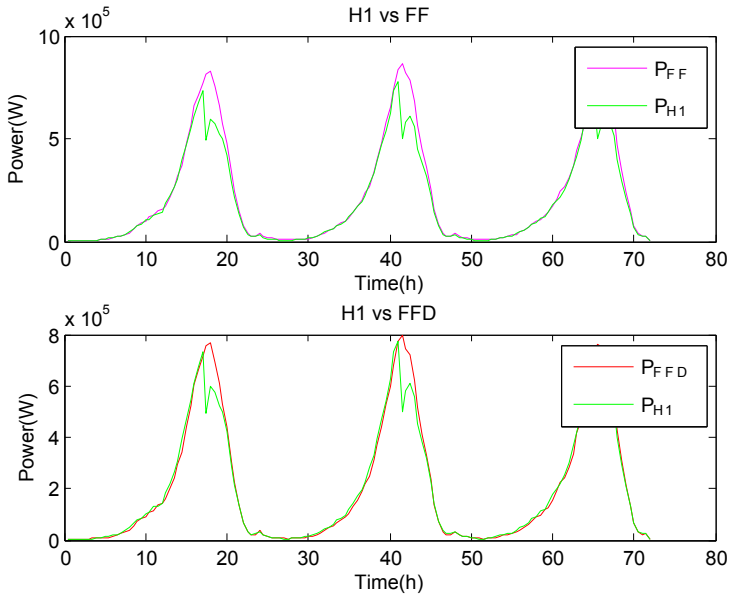

Figure 18: Note that even when anticipating, it uses less weighted energy than FF

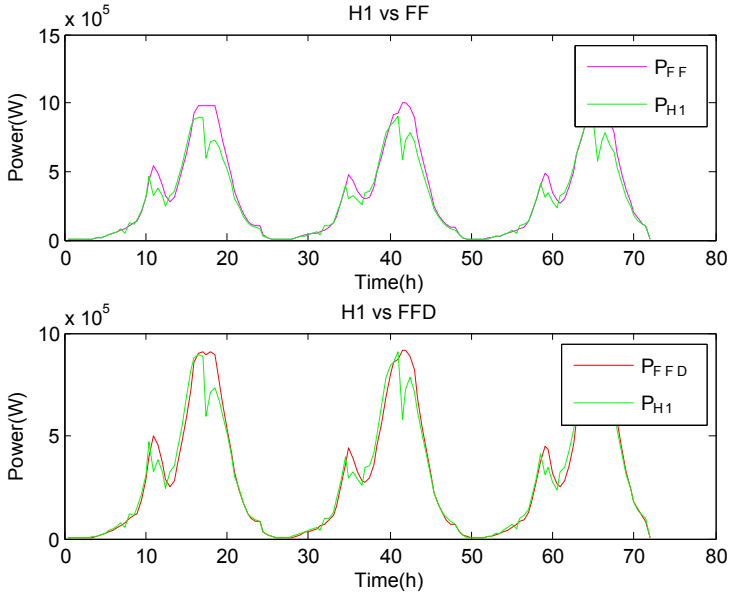

Figure 19: Note that the first peak causes a decrease in the performance later

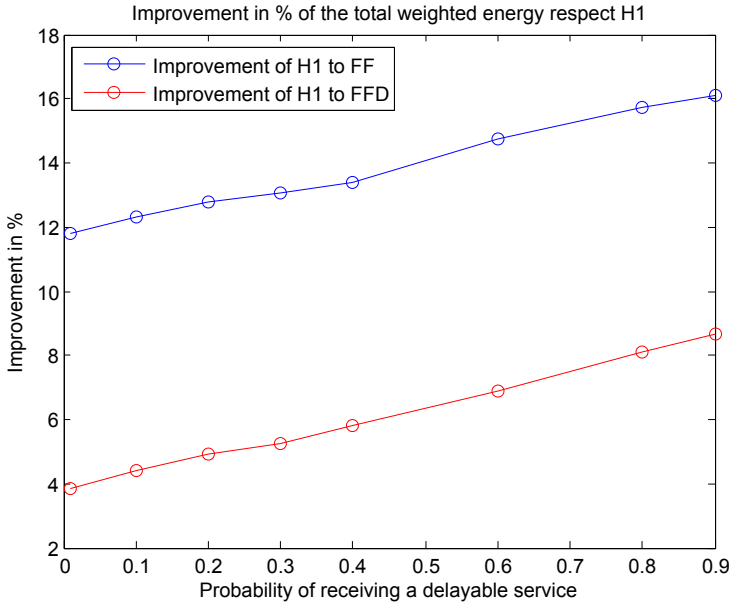

Figure 20: Surprisingly, it is a linear function differing by a constant 8%

4.3.6 *Further requests from the EP*

The EP can request the DC to apply some tight restrictions on the power consumption, e. g.consume less than a 90% of what was consumed in the last hour. Being the objective to minimize the total weighted energy consumption, and assuming that these requests will usually come in moments of high-valued CEF, it is fair to assume that the optimal solution should follow this request.

However, if the solutions here proposed failed to achieve this restriction, the only thing that could be done would be trying to delay more or less services (or anticipate less or more, respectively). This can be easily done by listing all the delayable services, ordering them from bigger to smaller and change their state from delayed to not-delayed depending on the objective. If even then the solution is not feasible, we will have to reject the request made by the DC.

4.3.7 *A Genetic Algorithm proposal*

A proposed way to improve the second heuristic is using the meta-heuristic known as Genetic Algorithms, that has been explained in Chapter 2. The idea here is to find which subset of all the delayable services is better to actually delay.

The guidelines for this method would be:

1. Encode all the delayable services as a binary string: bit j is 1 if and only if the delayable service in the position j is going to be delayed.

2. The fitness function is the same function (minimizing the P_{AC} as has been explained previously). We select the best options.

3. The crossing operator is very simple, select two random positions and exchange the bit string in the middle between solutions.

4. Mutation is also simple, $P[b_j \leftarrow b_j XOR 1] = p$ for each bit in the string.

Genetic Algorithms have proved to be very robust, and thus this proposal will probably be more robust even than the second heuristic. However, it probably would have the same problem because it will still be myopic with respect to CEF and service requests.

4.4 FINAL COMMENTS

We have seen that, although in theory the second heuristic is more robust, in the practice the first heuristic delivers better results at least for the situations specified in the simulations. However, it is true that the second heuristic is, in general, more robust, although it is not so effective. It is important to note that these robustness and effectiveness are a problem of the online situation we are dealing with, i. e.the lack of knowledge of the future CEF and requests.

Note as well that the second heuristic can be extended to an offline heuristic, where the 'only' thing that has to be done is studying the ramifications of each decision regarding each service that comes in the slot i. This is comparable to having a tree for each service that starts with two leaves and then ramificates for each service in the future; in the second heuristic the decisions made with the other services in the slot i do not affect.

Actually the optimal solution can also be understood as finding a branch of a tree: For each delayable service, we study all the possibilities that result from delaying/non-delaying that service (plus the optimal chain of P_{AC}) and find the 'optimal' in a recursive way. By optimal we mean the best combination of delaying/non-delaying for all services from 1 until N assigning the services with a FF/FFD algorithm or another BPP algorithm.

Another way would consist in encoding the delayed services in the manner of the Genetic Algorithm. Then we would have a tree with $2^{Del(i)}$ leaves in each level where $Del(i)$ is the set of delayable services, that might include the ones that are susceptible of anticipation. Be careful with this way of understanding the delaying because in the FF case, the order in which we delay and place them later matters (but we can avoid the problem just by sorting the delayed services anyway). Note that the genetic algorithm proposed in the previous section would also be extendable in this manner to the offline problem, giving a robust approximation to the 'optimal' solution.

However, going back to the online problem, this whole strategy has a big handicap: The knowledge of CEF. By not knowing it, we are fooled to think that $sgn(CEF(i+2) - CEF(i+1))$ will be the same as $sgn(CEF(i+1) - CEF(i))$, or that $CEF(i+j) = CEF(i+1)$ for all j, depending on how we analyze it, and that is where the heuristics make a mistake. For example, if $CEF(i)$ is a non-descending linear function, then the second heuristic is better than the first if we tune correctly the assumption for the probability of using one unnecessary server in the future, which empirically has been found to be equal to the size of the service. But while the second heuristic has been delaying some services when CEF was increasing, as soon as the CEF starts to lower, the first heuristic gains a massive advantage. Probably if the 'extended' offline version knew more about the future (or had more advice, in the online decision problems' jargon), it would decide not to delay the services that would be running until a certain peak in number of requests weighted by CEF.

However, the first heuristic is also not perfect: It is easier to 'deceive' it with a small peak in CEF which will cause it to delay services and lose the chance to delay them later. Even a big peak in CEF that does not coincide with a big peak in the current available services (meaning the ones that are working plus the requested) will deceive the first heuristic and the second heuristic. How can we avoid this problem?

The proposal for continuing the work done in this project is to try to add a hysteresis to the first heuristic in terms of the power consumption obtained by a simple FFD. If a FFD algorithm is not decreasing 'too much', or 'above variance', then assume that the peak is just some random noise, however if the FFD algorithm starts decreasing the power consumption in a very fast manner, then that is the moment to delay as many services as possible. This is similar to what a predictor in stock markets does, it tries to decide whether the stock price has really reached a peak or it is only under normal perturbations.

The knowledge of the statistics of CEF and the number of requests can also help to predict this peak, thus increasing the improvement of these heuristics with respect to the original situation, in which Data Centers use a FFD, FF or even give low priority to energy consumption in front of makespan of the services.

5

CONCLUSIONS

We have presented the problem of energy consumption and carbon emissions in a DC, and the objective of fostering a collaboration between all the members of an ecosystem DC-EP-ITC introduced by the project All4Green. The collaboration is enforced by some special 'Green' contracts
(GreenSLA, GreenSDA) that introduce flexible QoS parameters as well as some collaboration clauses. The goal of the project is not only to reduce the energy consumption and carbon emissions, but to show that this is possible and economically sustainable for all the parties involved.

Focusing precisely on the collaboration, the cornerstone lies on the ability to receive requests from an EP and negotiate with the ITC for being able to delay or pause their services. This project tries to study this particular aspect of the approach of All4Green by modeling the problem and studying its mathematical properties.

To this end, first we have studied the tractability of different mathematical problems and the theory of complexity and tractability of optimization and online decision problems. We have presented the added complexity of online decision problems and the metaheuristics that attempt to be global solvers for any kind of problem. Finally we have studied in depth the famous problem BIN PACKING PROBLEM (BPP). The BPP consists in assigning different items to different bins using as few bins as possible. More precisely we have seen the Dynamic BPP in which services last for a finite time and leave after that.

The Dynamic BPP has been the basis of the model that we have built trying to explain and understand the behaviour of a DC. We only had to add the Air Conditioning and the possibility to delay or anticipate services. This model is not completely precise nor rigorous, but it allows us to develop some intuition in how we should manage the delays and assignation of services. Even though this model is simple enough for us to manage it safely, the problem beneath it (BPP) is already too difficult to solve (it is $\mathrm{NP}-\mathtt{hard}$ and it can not be easily approximated).

For the aforementioned reasons, we have introduced a modification to the Dynamic BPP by adding delay, and provided two different heuristics and possible future improvements to the problem. The different heuristics can be summarized as follows:

1. The first heuristic follows to the letter the specifications given by the project All4Green. It delays as many services as it can

while the prediction for carbon emissions is lowering (Energy Saving State), and it anticipates services as the prediction for carbon emissions is increasing (Extra Energy Mode), thus trying to counter the effect of the rest of the demand to the EP. Then it assignates the services as if two different BPP were going on, and finds the optimal value for the power spent on the AC.

2. The second heuristic is completely dynamic: In each moment it decides whether to delay or anticipate each service it can, trying to see for each service whether it is better to delay it or not. Then it assignates each of these services as in a BPP (one for the delayed and one for the non-delayed), and finds the optimal value for the power spent on the AC.

These two heuristics can be viewed as the two different extremes of a whole chain of different heuristics, and we give a first theoretical analysis to both of them, showing that the second is more robust than the first. We have tried to explain the differences between both heuristics and showed some results of simulations in realistic scenarios for both. We have presented a genetic algorithm and we have related the heuristics and the genetic algorithm to an offline optimal algorithm.

We have seen that both of them do better than non collaborating algorithms, which was one of the objectives of the project, and we have observed that the first heuristic behaves better in general than the second. We have tried to explain the reasons and find improvements to both of them.

We have seen that one of the biggest issues of these approaches is that we do not know in advance the service requests and the CEF, which jeopardizes all the calculations and can mislead to bad results. A solution that we have proposed consists in losing sensitivity to small changes in service requests weighted by CEF (actually, comparing it to a non-collaborating algorithm), trying to emulate the financial algorithms that decide whether a stock price is starting to lower or on the contrary it is just under minor perturbations.

All in all, we believe we have given more than enough reasons to be convinced that the approach taken by All4Green can lead to a major success if the guidelines given here are followed. We have only focused on a very small aspect of All4Green, namely the collaboration, and still we have reached considerable improvements in the most conservative situations. It yet remains to include the flexibility clauses and the possibility to migrate services to another DC in a federation.

Further experimental investigations are needed to attune the parameters and the model to the reality and give more accurate solutions, although even this rough model will probably give good results in the end.

Part III

APPENDIX

A

PROOFS FOR THE RESULTS

A.1 NP-COMPLETENESS AND TRACTABILITY

Lemma 2.1.3. *Suppose that an optimization problem Q_1 is p-reducible to an optimization problem Q_2. If Q_2 is solvable in polynomial time then so is Q_1*

Proof. Suppose that Q_1 is p-reducible to Q_2 via the instance function χ and the solution function ψ, both computable in polynomial time. Then an optimal solution to an instance $x \in I_{Q_1}$ can be obtained from $\psi(x, \chi(x), y_2)$ where y_2 is an optimal solution to the instance $\chi(x) \in I_{Q_2}$ and is supposed to be constructible in polynomial time. □

Lemma 2.1.4. *Suppose that an optimization problem Q_1 is p-reducible to an optimization problem Q_2. If Q_1 is $NP - hard$, then so is Q_2.*

Proof. Suppose that Q_1 is p-reducible to Q_2 via the instance function χ and the solution function ψ, both computable in polynomial time. Since Q_1 is $NP - hard$, there is a decision problem D that is polynomial time reducible to Q_1, let h and g be the two functions seen in Definition 2.1.10. Define two functions h_1 and g_1 as follows: for any instance x of D, $h_1(x) = \chi(h(x))$; and for any solution y to $h_1(x)$, $g_1(x, h_1(x), y) = g(x, h(x), \psi(h(x), h_1(x), y))$. It is clear that these functions are polynomial time computable.

Let x be an instance of D. Then $h(x)$ is an instance of Q_1 and therefore $h_1(x)$ an instance of Q_2 following thus Definition 2.1.10. Now, if x is a instance of D, then $g(x, h(x), z) = 1$ if and only if x is a yes-instance for D and z is an optimal solution for $h(x)$. If $z = \psi(h(x), h_1(x), y)$ then clearly z is an optimal solution for $h(x)$ if and only if y is an optimal solution for $\chi(h(x)) = h_1(x)$. Going backwards, y is an optimal solution to $h_1(x) \in Q_2$ and x is a yes-instance for D if and only if $\psi(h(x), h_1(x), y)$ is an optimal solution for $h(x)$, if and only if $g(x, h(x), \psi(h(x), h_1(x), y)) = g_1(x, h_1(x), y) = 1$ as we wanted to prove.

This proves that the $NP - hard$ decision problem D is polynomial time reducible to Q_2. Consequently, Q_2 is $NP - hard$. □

Theorem 2.1.5. *Let Q be an optimization problem and Q' be a subproblem of Q. If the subproblem Q' is $NP - hard$, then Q is $NP - hard$.*

Proof. Being the problem Q' $NP - hard$, there is a decision problem D that is $NP - hard$ and is polynomial time reducible to Q' via the two functions h and g. Now, if x is an instance for D, then

$h(x) \in I'_Q \subset I_Q$. Moreover, $y \in S_Q(h(x))$ is optimal to Q if and only if it is optimal to Q' (because $S_{Q'} = S_Q$ and $f_{Q'} = f_Q$). Now it is straightforward to see that h and g also work to reduce D to Q, and therefore Q is $NP - hard$. □

Theorem 2.1.6. *Let* $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ *be an optimization problem such that for any* $x \in I_Q$ *we have* $Opt(x) \leqslant p(length(x), max(x))$ *for a given polynomial* p. *If* Q *has a FPTAS, then* Q *can be solved in pseudopolynomial time.*

Proof. Suppose that Q is a minimization problem, i.e.$opt_Q = min$. Since Q has a FPTAS, there is an approximation algorithm $\mathcal{A}$ for Q such that for any $x \in I_Q$, $\epsilon > 0$ the algorithm produces a solution $\mathcal{A}(x) \in S_Q(x)$ in time $p_1(|x|, \frac{1}{\epsilon})$ (p_1 is a polynomial) satisfying

$$\frac{f(x,y)}{Opt(x)} \leqslant 1 + \epsilon$$

In particular, if $\epsilon = \frac{1}{p(length(x),max(x))+1}$ then the solution y satisfies

$$f(x,y) \leqslant Opt(x) + \frac{Opt(x)}{p(length(x), max(x)) + 1} < Opt(x) + 1$$

Given the fact that $Opt(x) \leqslant f(x,y)$ and that $f(x,y)$ and $Opt(x)$ are integers, then forcefully $f(x,y) = Opt(x)$. Furthermore, the running time of the algorithm $\mathcal{A}$ is in this case bounded by $p(|x|, p(length(x), max(x)))$, which is a polynomial in $length(x)$ and $max(x)$. Therefore Q can be solved in pseudopolynomial time. □

Theorem 2.1.7. *Let* $Q = \langle I_Q, S_Q, f_Q, opt_Q \rangle$ *be an optimization problem. If there is a fixed polynomial* p *such that for all* $x \in I_Q$, $Opt_Q(x)$ *is bounded by* $p(|x|)$, *then* Q *does not have a FPTAS unless* Q *is in* P.

Proof. Let $\mathcal{A}$ be an approximation algorithm that is a FPTAS for Q, bounded by $O(\frac{n^c}{\epsilon^d})$ for some fixed c and d larger than 0.

Let h be a fixed constant such that $Opt_Q(x) \leqslant n^h$.

We will distinguish two cases: If $opt_Q = min$, let $x \in I_Q$ and $A(x) = f_Q(x, \mathcal{A}(x))$, by the definition we have that for any $\epsilon > 0$, $\mathcal{A}$ constructs a solution in polynomial ($O(\frac{n^c}{\epsilon^d})$) time a solution with approximation ratio $\frac{A(x)}{Opt(x)} \leqslant 1 + \epsilon$. If ϵ is set to $\epsilon = \frac{1}{n^{h+1}}$ then:

$$1 \leqslant \frac{A(x)}{Opt(x)} \leqslant 1 + \frac{1}{n^{h+1}}$$

Or equivalently

$$Opt(x) \leqslant A(x) \leqslant Opt(x) + \frac{Opt(x)}{n^{h+1}}$$

But the boundary on $Opt(x)$ implies that $\frac{Opt(x)}{n^{h+1}} < 1$, and this together with the previous inequality proves that $Opt(x) \leqslant A(x) <$

$Opt(x)+1$. Being both $A(x)$ and $Opt(x)$ integers, it is clear that $A(x) = Opt(x)$. The running time of $\mathcal{A}$ in this case is $O(n^{(}c + d(h + 1)))$ which is a polynomial.

If $opt_Q = max$ then $A(x) \leqslant Opt(x) \leqslant n^h$, so in polynomial time (if we choose the same ϵ as before) the algorithm (A) constructs a solution to x with the value $A(x)$ such that

$$1 \leqslant \frac{Opt(x)}{A(x)} \leqslant 1 + \frac{1}{n^{h+1}}$$

which gives

$$A(x) \leqslant Opt(x) \leqslant A(x) + A(x)/n^{h+1}$$

Now since $\frac{A(x)}{n^{h+1}} < 1$ again we find $Opt(x) = A(x)$. □

Theorem 2.3.1. *It is* $NP-complete$ *to decide if an instance of BPP admits a solution with two bins.*

Proof. Reduce from PARTITION which is known to be $NP-complete$. The PARTITION problem is as follows:

- I_Q: A n-tuple $\langle c_1, \ldots, c_n \rangle \in \mathbb{Z}$.
- S_Q: A set (if it exists) $S \subset \{1, \ldots, n\}$ such that

$$\sum_{i \in S} c_i = \sum_{i \notin S} c_i$$

The problem consists in deciding if it exists. Given a PARTITION instance, we create an instance for BIN PACKING by setting $s_i = c_i$, $T = \sum \frac{c_i}{2}$. By Corollary and because PARTITION is $NP-hard$, the decision problem of a two bins BPP is $NP-hard$. It is in NP (give as a hint the partition) and therefore it is $NP-complete$. □

Theorem 2.3.3. *BPP is in* APX, *being* 1.5 *the boundary for its approximation ratio, unless* $P = NP$.

Proof. First we prove that BPP is $NP-hard$. In the same line as the previous proof, let $\langle c_1, \ldots, c_n, \rangle \in \mathbb{Z}$ be an input instance for the PARTITION problem (which is NP-hard). Let $T = \sum c_i$. If T is odd, then c_i is a no-instance for the PARTITION problem, so we construct the instance for the BPP in the following way: $\langle T, T, T; T \rangle$ which obviously will use 3 bins. If T is even, then the instance for the BPP is $\langle c_1, \ldots, c_n; T/2 \rangle$ as in the previous proof. Now if and only if c_i is a no-instance for the PARTITION problem, T is either odd and the optimal solution for the BPP problem is 3 bins, or it is even and the optimal solution still uses 3 bins. But if, and only if, c_i is a yes-instance for the PARTITION problem, then T is even and the optimal solution will use 2 bins. Therefore PARTITION is reducible to BPP and BPP is $NP-hard$.

Now, if there was an algorithm $\mathcal{A}$ running in polynomial time such that $A(\alpha) < 1.5Opt(\alpha)$ for all $\alpha \in I_{BPP}$ (recall that $A(\alpha)$ is the value of the objective function for the result given by $\mathcal{A}(\alpha)$), we will prove that we can solve the PARTITION problem in polynomial time: use the same reduction as above and find the optimal solution for the BPP. It will be solved in polynomial time and will find a solution with m bins. If $m \geqslant 3$, since $\frac{m}{Opt(\alpha)} < 1.5$, we get that $Opt(\alpha) > 2$ therefore we can know that the instance is a no-instance for the PARTITION problem (in polynomial time). Otherwise if $m \leqslant 2$ it must be $m = 2$, and therefore the instance is a yes-instance for the PARTITION problem and we found it in polynomial time. This means that the instance is a yes-instance for the PARTITION problem if and only if the approximation algorithm returns a result which uses 2 bins. But PARTITION is $NP - complete$, and therefore unless $P = NP$ there is no polynomial running time algorithm that solves it. Therefore unless $P = NP$ there can be no polynomial running time algorithm that approximates BPP with an approximation ratio less than 1.5. □

BIBLIOGRAPHY

[1] Jyrki Huusko, Hermann Meer, Sonja Klingert, and Andrey Somov, editors. *Energy Efficient Data Centers*, volume 7396 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[2] Sonja Klingert, Thomas Schulze, and C. Bunse. GreenSLAs for the Energy-efficient Management of DCs. presented in 2nd International Conference on Energy-Efficient Computing and Networking, Columbia University, New York, 2011.

[3] Philipp Wieder, Ramin Yahyapour, and Wolfgang Ziegler, editors. *Grids and Service-Oriented Architectures for Service Level Agreements*. Springer US, Boston, MA, 2010.

[4] J ChEn. Introduction to Tractability and Approximability of Optimization problems. *Lecture Notes, Department of Computer Science, . . .*, 2002.

[5] A Koster and Xavier Muñoz. *Graphs and algorithms in communication networks*. Springer, 2010.

[6] W Szpankowski. *Average case analysis of algorithms*. 2010.

[7] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for np-hard problems. chapter Approximation algorithms for bin packing: a survey, pages 46–93. PWS Publishing Co., Boston, MA, USA, 1997.

[8] Joseph Wun tat Chan, Prudence W. H. Wong, and Fencol C. C. Yung. On dynamic bin packing: An improved lower bound and resource augmentation analysis.

[9] Z Ivkovic and EL Lloyd. Fully dynamic algorithms for bin packing: Being (mostly) myopic helps. *SIAM Journal on Computing*, 28(2):574–611, 1998.

[10] E. G. Coffman, Jr., D. S. Johnson, P. W. Shor, and G. S. Lueker. Probabilistic analysis of packing and related partitioning problems. In *in Probability and Algorithms, 87-107, National Research Council*, 1999.

[11] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*. Springer Publishing Company, Incorporated, 2nd edition, 2010.

[12] DA Coley. *An introduction to genetic algorithms for scientists and engineers*. 1999.

[13] R. Basmadjian, F. Niedermeier, and H. De Meer. Modelling and analysing the power consumption of idle servers. In *Sustainable Internet and ICT for Sustainability (SustainIT), 2012*, pages 1–9, 2012.