Smart Efficient Decision System

A Bin Packing Approach

Gonçalo Marques Raposo de Magalhães

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor: Prof. Luís Manuel Marques Custódio

Examination Committee

Chairperson: Prof. José Fernando Alves da Silva

Supervisor: Prof. Luís Manuel Marques Custódio

Member of the Committee: Prof. João Manuel de Freitas Xavier

May, 2020

I thank my wife for all her patience.

For her and for our newborn daughter, Maria do Rosário.

Ad Majorem Dei Gloriam.

Abstract

The field of agriculture is a complex system with impact in the entire world. In recent years, we have seen immense technological advancement and investment in this sector, mostly related to automation of countless tasks. This field presents problems which relate both environmental variables, such as weather conditions, crop features, energy and water savings, and financial variables playing a role in human interests, such as the cost of producing a given product, the energy price distribution throughout the days, among others. Indeed, such problems have a great amount of variables. It is because of this complexity that most farmers choose not to look at this in detail, resulting in an efficiency loss which is commonly unknown.

The present thesis addresses the irrigation problem of a single system with one water pump and one fertilizer injection pump, a challenge involving many variables and decision processes which all farmers need to address, if a smart and efficient irrigation is desired. The irrigation problem in this thesis relates forecast predictions throughout one week, the difference of debit among each valve on the system and its relation with the water pump's full capacity, the temporal evolution of the amount of water in each part of the crop and its relation with the amount of water which each crop desires, the distribution of energy price throughout each hour of the week, and the necessity of fertilizer for each present crop.

The implemented solution aims at reducing the financial cost of the irrigation decision as much as possible, given a defined irrigation system with already selected crops. At the same time, the solution algorithm prevents the death of the crop and its over-watering, striving to keep each section of the soil as close as possible to its desired amount of water. Finally, the algorithm handles the problematic of injecting fertilizer in the system without over-fertilizing some section of the crop.

In the end, results show that an efficient irrigation decision is successfully created for an entire week. The algorithm effectively aims at using the cheapest energy hours of the week, without compromising the crop's health. A variety of variables is properly handled to create an intelligent and informed decision which was almost impossible before.

Keywords: decision, bin packing, scheduling, energy prices, evapotranspiration, efficiency.

Resumo

O campo da agricultura é um sistema complexo com impacto no mundo inteiro. Em anos recentes, temos visto um avanço e investimento tecnológicos imensos neste sector, maioritariamente relacionado com a automação de incontáveis tarefas. Este campo apresenta problemas que relacionam tanto variáveis ambientais, como condições meteorológicas, características da cultura, poupança de energia e água, como variáveis financeiras que desempenham um papel nos interesses humanos, como o custo de produção de um determinado produto, a distribuição do preço da energia ao longo dos dias, entre outras. De facto, tais problemas possuem um grande número de variáveis. É devido a esta complexidade que a maioria dos agriculturas opta por não olhar para isto em detalhe, resultando numa perda de eficiência que é frequentemente desconhecida

A presente tese visa o problema da rega de um único sistema com uma bomba de água e uma bomba injectora de fertilizante, um desafio que involve muitas variáveis e processos de decisão que todos os agricultores precisam de endereçar, se uma rega inteligente e eficiente for desejada. O problema da rega apresentado nesta tese relaciona previsões meteorológicas ao longo de uma semana, a diferença de débito entre cada válvula do sistema e a sua relação com a capacidade máxima da bomba de água, a evolução temporal da quantidade de água em cada parte da cultura e a sua relação com a quantidade de água que cada cultura deseja, a distribuição do preço energético ao longo de cada hora da semana, e a necessidade de fertilizante de cada cultura no sistema.

A solução implementada procura reduzir ao máximo o custo financeiro da decisão de rega, dado um sistema de rega definido com culturas já selecionadas. Ao mesmo tempo, o algoritmo solução previne a morte da cultura e a sua sobre-rega, esforçando-se por manter cada secção do solo o mais próximo possível da sua quantidade de água desejada. Finalmente, o algoritmo trata da problemática de injectar fertilizante no sistema sem sobre-fertilizar alguma secção da cultura.

No fim, os resultos mostram que uma decisão eficiente de rega é criada com sucesso para uma semana inteira. O algoritmo procura eficazmente utilizar as horas de energia mais barata na semana, sem comprometer a saúde da cultura. Uma variedade de variáveis é apropriadamente tida em conta para criar uma decisão inteligente e informada, que era quase impossível de ser feito antes.

Palavras-chave: decisão, bin packing, calendarização, preços energéticos, evapotranspiração, eficiência.

Contents

1 Introduction

The field of agriculture is one of the most complex fields in existence. This is due to the fact that countless variables enter in scene, to create a complex system dependent on biological, ecological, meteorological parameters, as well as irrigation specifications, energy prices, water consumption, economic growth and technological development.

1.1 Motivation

One must not forget how crucial the agricultural system is for the entire world. An epidemic can suddenly target a great portion of Spain's growing pistachio production[1], destroying trees which took seven years[2] to start giving fruit (and financial return to the producers); a small miscalculation of nitrate-based fertilizer input in a barley crop for cattle feeding can poison all the existing cows in the field[3]; the unawareness of rainy days in corn's harvesting period will force farmers to endure additional costs in high temperature dryeration[4], thus reducing their profit.

About 70% of the water which is taken from rivers and from groundwater is used for agricultural irrigation[5]. This gives a rough image of the enormous amount of water which the agricultural field needs. The agricultural field uses this water to produce rice, wheat, corn, tomatoes, strawberries, i.e. food for humans, but also for animals, which in turn provide once again food for humans. It is evident how important water usage is, not just for the agricultural field and the economy which it involves, but also for life on Earth. The efficient use of such an important resource can have great benefits in various ways.

Indeed, the field of agriculture is both full of risks and full of optimization opportunities. The simple fact of predicting when it will rain and irrigating accordingly can help a municipal governing body saving €100.000 in water consumption for its city gardens[6]. Although this is not an agricultural set, one sees the value of applying such optimization and predictive procedures in the agricultural field, where savings can have a much bigger impact.

The effect of energy price variation on agriculture is something known and studied[7]. The energy expenses due to pumps and filters, machines and motors, is a great part of the overall costs of the farmer. A big influence in the costs is not just in the mean energy price of the year, but also in the energy price distribution throughout the day and throughout the week. It is evident that a corn farmer irrigating its crop during the day will have a smaller profit margin in the market than another corn farmer irrigating its crop during the night, which is usually when energy price is in the lowest values. What's more, the second farmer may even come to sell its corn at lower costs, leading to a big disadvantage for the first farmer and to a better corn price in the market. Leveraging this opportunity concerning the energy price distribution, as illustrated by this small example, can not only provide advantages for the farmers who explore this complexity, but also enrich the free market and the population overall, given that farmers can have better profit margins and consumers are provided with better corn prices.

Another pie of the whole expense comes from the use of water resources, although often one manages to extract water from groundwater, rivers, lakes, which once again leave the expense on the pumps' work, a crucial energy cost. Either way, the use of water has impact on two levels. First, the use of water as such should always be perfected and optimized. This is a conclusion drawn from the fact that water is an indispensable resource for life as we know it, not to mention the fact that some places on Earth lack water resources. Second, the misuse of water leads either to less quality crops or to an excess use of pumps when much of the energy cost could be removed there. When one uses water efficiently, there are visible advantages concerning the amount of energy hours that are being used and the quality of the crop.

Evidently, managing the quality of the crop is complex. The proper way of addressing the needs of the plants is to know how thirsty it is. This is the same - or at least very proximate to - as knowing the level of water in the soil and in the crop. This, together with the water-related characteristics of the plan and its phenological phase, can tell the farmer how much water the plant needs in order to be in its ideal water level as much time as possible. This is a complicated task for one single plant

vase, where water inputs need to be accounted for as well as the rate of evaporation and transpiration of the plant. To leverage the task of tracking the levels of an entire agricultural field is quite heavy-lifting.

Lastly, the proper tracking of water level on the crop is not just for present information purposes. Indeed, this information is only relevant if it can help in deciding the next irrigation action. For an efficient decision, weather forecasts need to be accounted for, because they will tell if it is going to rain (thus, no need for water and energy spending) or if it will come great waves of heat which need smart irrigation responses. Weather does not just impact the efficiency of one irrigation decision, but it influences all components of crop production and food production[8]. A smart prediction of how weather will develop in the coming days can actually save crops and assure a normal state in the food market.

There are much more aspects which need to be properly handled when entering the agricultural field. The enormous amount of information and variables turn this field into a highly complex network of problems. Indeed, the usual response is to ignore some of the problems, or just account for the risk factor in the investments. Because of this, and because the majority has conformed with the idea of living with all such problems and inefficiencies, the field of agriculture has a variety of opportunities in optimization. The exploration and solving of such problems can lead to major improvements on the overall quality of life, potentially becoming a disruptive force.

1.2 Structure of the thesis

In the present *Introduction* chapter, the problem handled in this thesis will be explained, going through the various inputs and algorithmic stages.

Following this, a revision is done to the state of the art concerning the development of intelligent systems and algorithms to solve various irrigation problems. The review is divided in approaches based on sensor readings, phenotyping, evapotranspiration, and others. A chapter on algorithms is presented next, divided into *Packing Algorithms* and *Linear Programming*.

Finally, two chapters explain the *Solution Implementation* and the *Conclusions*, respectively. The first one is divided into three different studied approaches: linear programming approach, combinatorial approach using bin-like objects and a *greedy decisions* approach. The second chapter draws some conclusions from the various results and provides thoughts on possible future improvements.

1.3 Basic Concepts

Before fully formulating the problem that is addressed by this thesis, a few concepts need previous explanation:

- Irrigation section or sector: a space of same kind crop which is irrigated by one single controlled valve, which may or not pour water from more than one place in that space, and which is assumed to have the exact same meteorological and moisture conditions.
- Irrigation system: a group of irrigation sectors which get water and fertilizer from the same pump and injection pump, respectively.

1.4 Problem Formulation

Given the variety of variables in the agricultural field, it is quite difficult to make an efficient irrigation decision which not only handles the current conditions but also the future ones. The present thesis intends to develop and present a smart efficient decision algorithm for any given irrigation system, based on various inputs from the surrounding environment and features/limitations of the given system. The solution optimality will be assessed by comparing it with different strategies of packing. Packing is a type of algorithmic approach which intends to divide a problem in small containers which need be filled (this will later be explained in detail). The goal is to show that the algorithm solves the problem with such efficiency that the small time complexity will make it worth the optimization cost, achieving an algorithm which can indeed be used for real situations.

1.4.1 Inputs

Various key inputs are to be considered throughout the decision process.

1.4.1.1 Evapotranspiration

Weather conditions critically affect the water levels on the soil. Thus, one simply cannot ignore weather conditions on the spot, as well as the forecast for the next days. Major meteorological factors are the heat, sun exposure, rain, humidity and wind. All these can be assessed by computing the evapotranspiration index. This index quantifies the loss of water in the soil, in evaporation and culture transpiration.

It is assumed that the system is fed with hourly forecast for the next seven days, which result on hourly evapotranspiration indexes. The Penman-Monteith equation is the recommended method for determining reference evapotranspiration [9]. In this equation, one can see that the evapotranspiration indexes are based on a variety of meteorological parameters, such as irradiance, wind measures, atmospheric conductivity, air density, specific humidity, heat, and others.

1.4.1.2 Current water balance

The current water balance corresponds to the amount of water that is present on a given irrigation section. For an algorithm to successfully create an efficient irrigation plan for various days, the water balance of each sector must be tracked and constantly estimated. Thus, the initial water balance of all sectors is assumed to be given.

1.4.1.3 Energy prices and consumption

An important aspect in an irrigation plan is the price of energy, and the amount of consumption. Indeed, the price of the energy varies with the hour of the day and with the day of the week. The energy consumption is mostly attributed to the water pump. This means that it is preferable to irrigate in cheap time slots. Furthermore, the water pump has an energy consumption curve which varies with the amount of water it is delivering. Its sweet spot (meaning the point where ratio between energy consumption and water delivery is lower) is near to the maximum of its capacity. Thus, an efficient plan also corresponds to an irrigation plan where the water pump is working near the sweet spot in most irrigation slots.

The energy prices are assumed to be given for all the hours of the irrigation plan.

1.4.1.4 Culture parameters

Each culture has its own watering needs, fertilization needs, absorption limitations, etc. These parameters can also vary with the phenological phase in which the culture currently is. Thus, each culture will have attributed to it a minimum, maximum and ideal water balance. The minimum water balance corresponds to the lowest amount of water in soil which guarantees the survival of the culture. The maximum water balance corresponds to the point where the culture can no longer absorb that amount of water, meaning that the water balance will never get higher than that value - it is the point where water is just being wasted on that culture. The ideal water balance is the sweet spot for that culture on that specific phenological phase.

All these values are assumed to be given in the beginning of the algorithm.

1.4.1.5 Irrigation network limitations

As previously discussed, the water pump has a maximum capacity, which corresponds to its energy consumption sweet spot. Furthermore, each sector has a specific flow rate, which is unchanged. If we also consider the fertilizer pump, its fixed debit must also be accounted. These parameters describe the irrigation network and constraint the irrigation problem. They are given in the beginning of the algorithm as well.

1.4.1.6 Fertilization complexity addition

Most agricultural irrigation scenarios need to account for yet another parameter, which can actually drastically increase complexity if we're looking for any sort of optimal irrigation plan - the fertilizing procedure. Fertilizers provide cultures with essential nutrients for their healthy growth. In the present thesis, the accounted fertilizers will always be liquid fertilizers which are applied to the cultures through an injection pump which directly injects the substance into the water flowing in the irrigation network.

Fertilizers can also be used to balance the soil's pH, in order to achieve acidity/alkalinity sweet spot for the growing culture. Accordingly, the basal soil's pH will influence the amount of fertilizer needed and the fertilizer to choose, especially since the soil's basal pH level can vary between regions[10]. Since a fertilizer may have extreme levels of pH, it is very important to inject the right amount of it throughout the culture growth. Furthermore, an injection pump is an expensive material in the agricultural field; if fertilizers precipitate inside it, it can damage it beyond repair. That is why it is necessary to always clean the conducts after fertilizing (just normal irrigation can be used to clean it, while opening a leakage valve).

For the problem in hand, the fertilizing procedure creates a higher level of complexity due to two main issues:

- The fertilizer is entirely dependent on having water flow inside the irrigation network, which means that the fertilization procedure can only happen while irrigation is taking place and it also means that the fertilizer will be applied in all the sectors which are being irrigated.
- The injection pump has a fixed debit, and the fertilizer dilutes along the entire irrigation network. This entails that the fertilizer supply rate for a given sector is dependent on the amount of sectors being irrigated at the same time and on the water debit for each of those sectors.

One sees where the complexity starts. The problem addressed by this thesis deals with duration times. Indeed, if a given sector needs a certain amount of irrigation, we know, regardless of which sectors are being irrigated at the same time, how long it will take to provide that sector's culture with its desired amount of water. In the fertilization procedure, one only knows the volume of fertilizer needed by a given culture. The amount of time which the injection pump takes to provide that quantity is inescapably dependent on which sectors are being irrigated in that period of action.

The present thesis will aim at tackling a problem with the added complexity of fertilization. Though a variety of issues rise with the use of fertilization in an irrigation system (network cleaning, injection pump cleaning, fertilizer mixing, as well as different ways of fertilizing a system such as continuous or pulsed fertilization procedures), this thesis will assume the use of a single fertilizer, a single injection pump, and no cleaning procedures are accounted for. This does not weaken the practicality or feasibility of the presented algorithms. Indeed, one could think of a simple program capable of abstracting the entire fertilization complexity into a single fertilization necessity for each of the cultures, and cleaning procedures could be accounted inside the irrigation moments. Some minor loss of optimality could be associated to this simplification, nevertheless.

1.4.2 The algorithm

This thesis intends to provide an algorithm which generates a complex irrigation plan for a possible model of an irrigation system. The algorithm is responsible for providing an optimized solution for the model in hand, translating to an efficient set of irrigation decisions. The algorithm will be evaluated based on specific cost functions, time complexity and resource consumption, among others.

1.4.2.1 Taking energy tariffs into account

The variation of the energy price across the hours of the day and throughout the days of the week is a big influence in the irrigation decision. For example, the impact is evident if the energy at night is half of the price in the hours with sun exposition. If the crop can be irrigated just in these less expensive hours, at the end of the farming campaign one will have spent 50% of what would have spent if one were to irrigate only in the expensive hours. Furthermore, if irrigating the crop during the weekend is less

expensive, it would be highly beneficial if the irrigation plan had this into account.

Energy costs is a constant concern in the agricultural field. For instance, even a small change in energy costs can potentially lead to major changes in the food prices [11], since agriculture profit margin would decrease, leading to higher retail prices. It is true that agricultural machines are getting more efficient as technology evolves, but still the irrigation procedure can drastically benefit from more efficient irrigation plans.

- **pump efficiency**

The water pump is the most critical machine in the irrigation process. When an irrigation network is designed and installed, a given water pump is selected with specifications suited for the existing sectors. Its location is also important if you want to somehow minimize the losses of pressure during the irrigation process.

Every water pump has a specific efficiency curve, which relates pressure with debit. The following graph is an example of it:

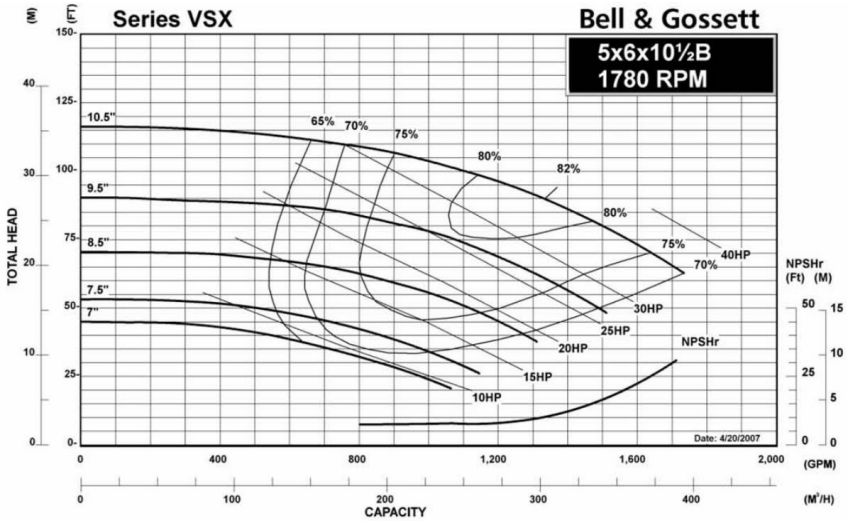

Figure 1: Pump curves. [12]

The nominal debit of the water pump is usually the maximum debit that the pump can output without significant efficiency loss. In agriculture, one can usually have a pumping pump with or without a speed variator. When a speed variator is used, the pump curve is able to move as to work in a given pressure value regardless of its debit, although it looses efficiency if the debit is too low. Roughly speaking, when a water pump has a speed variator, the pump efficiency is mostly guaranteed, and the major energy loss is in pressure drops. For instance, if a pump is irrigating 2 sectors, one needing 10 bar from the watering pump and the other needing 2 bar, the same pump will have to work to provide 10 bar. Assuming the irrigation network has a device in each sector which assures the pressure is reduced to the desired value, that irrigation process is watering the second sector with a loss of 8 bar, i.e. about 80% is being lost in pressure drop. In this situation, the way to minimize energy waste is to aggregate sectors by resemblance in pressure needs: first irrigating the sectors needing 2 bar, then irrigating the sectors needing 10 bar. The speed variator assures the right debit is given for that amount of pressure.

When dealing with an irrigation pump without speed variator, the desired working value is the maximum nominal debit of the pump, which provides highest efficiency. The problem of maximizing the pump proves to have resemblance to the bin packing problem, as it will be explained. The added complexity of the speed variator not only constraints the packing problematic but it also deemphasizes it: maximization of the pump capacity is not as important as packing sectors with similar pressure needs.

Given that this thesis is looking at the bin packing approach on creation of an irrigation plan, the problem to be resolved will not have a speed variator. The speed variator problem is considered a more relaxed packing problem with an added constraint.

1.4.2.2 Energy and Water-Balance Problem

As previously stated, energy tariffs create variations in the energy price throughout the hours of each week. As a result, it is preferable to irrigate in certain hours of the day and to avoid others, even to irrigate in certain days of the week as opposed to more expensive ones. An irrigation plan cannot simply water the crop in the cheap time slots of the week, though. Indeed, the irrigation plan is bound to the needs of the crop as well, and it cannot leave it in too much stress.

To leverage the complexity of knowing how much and when to water a given sector's crop, the concept of **water-balance** needs to be introduced. Water-balance is a name used to describe the technique of tracking the amount of water in the soil and/or in the crop. A given crop on a given phenological stage can be assumed to have three important tabulated water levels:

- Minimum water level - it corresponds to the least amount of water with which the crop can survive.
- Maximum water level - it represents the amount of water which saturates the soil. This means that the water-balance level cannot go higher than this value, which is the same as saying that this is the point where the irrigation starts to just waste water on the crop.
- Ideal water level - it is the amount of water with which the crop best demonstrates a healthy aspect in that given phenological stage.

Irrigation events should be done during cheap energy time slots, guaranteeing, at the same time, that neither the maximum nor the minimum water levels are reached. The loss of soil water with time is given by the evapotranspiration.

The first algorithm developed in this thesis intends to create an efficient irrigation plan, taking the mentioned constraints into account, not forgetting of trying to maximize the pump efficiency. The algorithm has a decision resolution of 1 minute, and the generated plan corresponds to an entire week.

1.4.2.3 Energy and Water-Balance Problem with fertilization

As previously noted, fertilization adds an entirely different complexity level. The irrigation plan has now more culture necessities to fulfill than mere thirst. Fertilization needs to be added to the field according to each sector's necessity. Due to an existing variety in the sectors debit, the fertilizing procedure duration will vary for each culture according to many parameters:

- The number of simultaneously opened sectors will change the fertilizer dilution along the irrigation network, meaning that a higher number of opened sectors at the same time will lead to an increase of process duration.
- The debit of a given sector will tell the percentage of diluted fertilizer which will go to that given culture. E.g.: if an irrigation network is providing 10.000 liters per hour in total for the currently opened sectors, a valve with 100 liters per hour as debit will deliver 1% of the overall diluted fertilizer to its culture.
- As already stated, given that the fertilizer will be provided to all opened sectors, an irrigation plan may need to close irrigation valves before the culture's watering needs are fulfilled, in order to maintain the fertilization procedure without over-fertilizing the previous sector's culture.

These points are added to the previous ones addressed in the problem above. The second algorithm developed in this thesis, besides all the rest, must make sure that fertilization needs are roughly fulfilled. By roughly one means that it is normal for a given culture not to receive the amount of needed fertilizer in a given day or week, as long as those absences are successfully bridged shortly after. In essence, it signifies that the algorithm will aim at fulfilling fertilizing daily needs, but in more extreme cases it will leave the

fertilization for another week. The amount of fertilizer that was not injected is therefor accumulated for the following week, which will become a separate problem. The postponing of fertilization events should not happen when a given fertilization deficit is finally reached. This mentioned algorithm also aims at balancing the water level around the ideal water level of each crop, assuring the best possible growth. The algorithm has a decision resolution of 1 second, and the generated plan still corresponds to an entire week.

1.5 Contributions

The present thesis successfully correlates some of the most key variables in the irrigation decision, developing a method which creates an informed decision for any given irrigation system with a single pump. The decision process taking place in irrigating a system is commonly simplified due to its complexity. The proposed algorithm gives farmers and irrigation controlling systems the power of not overlooking some of the key players in the agricultural field, thus regaining an efficiency gap which had been lost and almost forgotten.

This thesis presents a method of saving energy and water which in short is due to the compact arrangement of each sector's irrigation. Such task is not unknown in agriculture procedures. However, its efficient performance is complex, especially if one aims at taking into account the water in the soil and forecast predictions. Furthermore, the compact arrangement of watering with fertilizer in the system is a task which is not seen in any irrigation decision processes, due to its higher complexity.

Finally, in the present world of connected devices and intelligent controllers, this algorithm can be successfully implemented in practice, leading the thesis' solution out of the mere theoretical sphere, and enabling an actual impact in the farming industry. Moreover, the account of the actual level of water in the soil in each part of the crop can enable the use of the algorithm with a daily or hourly feedback. This means that, although the created decision is a week irrigation plan, the corroboration of the actual weather conditions and new hourly forecasts gives the algorithm a much more precise power of effectively knowing the crop's conditions and playing an informed decision in the field. This is a breakthrough instrument for precise irrigation and resource saving.

2 State of the Art Review

The automation problem on the field of agriculture is not a new one. Most jobs in the field involve heavy-lifting and monotonous tasks; lots of hectares of crop need to be irrigated with enough water for them not to die, although excess water would critically reduce the farmer's profit; weather and soil variables can vary each plant's moisture level; energy costs vary throughout the day and throughout the week. These are some of the problems that need to be addressed when thinking about irrigation efficiency and automation. As said, these problems are not unknown, and some studies have tried to tackle them. Although this thesis intends to provide a theoretical approach and algorithm analysis regarding efficient ways to plan irrigation events, it is worth mentioning some practical approaches which have recently been studied throughout the world. This gives a wider setup to the problem that is in hand, entailing what was not yet done and what was already tried, what has been accomplished and what is missing.

2.1 Sensors-based approach

The Internet of Things technology enables the possibility of having affordable live parameter monitoring within the environment. Particularly on the agricultural field, soil temperature and moisture readings are valuable for characterizing the state of given irrigation sectors. The existence of digital sensors and microcontrollers which can read from them and communicate its data opens doors all over the agricultural sector.

2.1.1 Live Remote Monitoring

A study[13] from the Institute of Management and Technology, Jalandhar, Punjab, achieves live IoT-based monitoring using a so-called "Agriculture Stick", an Arduino Mega 2560 with an ESP8266 Wi-Fi module and temperature and humidity sensors to extract these parameters from a given soil and send them to a computer, which handles its display for the user.

Evidently, this solution only handles the data extraction part, enabling the farmer to perform an informed decision over their irrigation systems, concerning whether or not the culture needs watering, and how much it needs, although we're supposing that the farmer will be able to digest the information received from the monitoring system and provide the right output. It essentially means that the farmer has now useful information, but the irrigation tasks were not automated, as it is not the purpose of the mentioned study. However, remote live monitoring is something that is definitely interesting for the development of an autonomous and efficient irrigation decision. Indeed, finding a way to know the level of water or humidity in the soil seems to be a necessity for efficient farming.

Furthermore, the mentioned work uses an Arduino Mega board. Arduino boards are suited for a development phase. However, the amount of peripherals, memory, and ports presents it as an overshoot for the specific task of live sensor monitoring. Finally, the ESP8266 Wi-Fi module is affordable and stable enough to be used for a production environment. Nevertheless, it is worth noting that it is bound to having some router or hot-spot near it, or a local network nearby, as it will communicate using Wi-Fi.

2.1.2 GPRS monitoring and Relay action

Following this, a study[14] published on the International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering tackles the next key component of a decision system, which is to decide the effective action based on the readings.

This work deploys automatic control of irrigation of each sector based on humidity and temperature sensors on it, a PIC microcontroller, GPRS protocol and a relay. It also uses a Zigbee module for wireless communication between the systems at stake. The sensors will give information to the microcontroller remotely, and the later will decide whether it should water or not. Finally, the decision is applied by switching the relay state.

This is definitely a step forward from the former study. Instead of just giving live information to the user, the system is autonomous enough to decide for itself what to do, based on the gathered information.

The GPRS allows for internet communication on any place that is covered by the operator. Thus, the controller has a real-time engagement with the irrigation system, but the present moment is the only thing that matters. As it can be deduced, it lacks the prediction and planning part of a normal efficient system.

2.2 Customizable automated irrigation system - HTP

A research article[15] from Iowa State University, Arnes, Iowa, United States of America, presents a "cost-effective and customizable automated irrigation system", used to control water availability for each plant without much monetary resources, thus enabling automation for high-throughput phenotyping in drought stress studies.

The implemented solution used an Arduino Mega as the controlling brain of the irrigation process, industrial data loggers for high accuracy measurement recording, a multiplexer to connect up to 48 sensors, and three 16-relay boards, all powered by a battery. Each relay serves as a on-off watering switch for a specific plant.

Though this research does not intend to find the most suited model for normal irrigation of agricultural sectors on uncontrolled environments, there are a few specifications that are worth mentioning about this study:

- Water availability control solutions on the market are costly, as the aforementioned work specified, noting that the research article was published on June 5, 2018.
- The system controls water availability for each of the plants, as if each one of them were to have a dedicated controlled irrigation process to provide the specific water volume desired and sensor monitoring for that given plant.

The industrial data logger and multiplexer are used to register sensor readings for many different plants and with high precision, bigger than what the Arduino controller would provide if standalone. As mentioned on the research article, a data logger can cost approximately $1,600, and a multiplexer can cost about $600. Although this could be considered a cost-effective solution regarding the needed specifications of the experiment, it would be an overcharge for normal irrigation automation, as we usually do not need that degree of precision.

We mentioned before that the Arduino would not be the best solution in the market, but it is interesting to note that once again the relay option is being used. Relays usage is an affordable solution for converting software decisions to real irrigation valves closing and opening events.

2.3 Evapotranspiration-based irrigation control

Another approach for irrigation control is the use of evapotranspiration monitoring. Evapotranspiration is the joint phenomenon of evaporation and transpiration happening on every plant. Food and Agriculture Organization of the United Nations defines this parameter as follows:

> "Evaporation and transpiration occur simultaneously and there is no easy way of distinguishing between the two processes. Apart from the water availability in the topsoil, the evaporation from a cropped soil is mainly determined by the fraction of the solar radiation reaching the soil surface. This fraction decreases over the growing period as the crop develops and the crop canopy shades more and more of the ground area. When the crop is small, water is predominately lost by soil evaporation, but once the crop is well developed and completely covers the soil, transpiration becomes the main process."[16]

A study[17] from North Carolina State University evaluates how efficient is an evapotranspiration based irrigation control system. The study compares two control models: one based on estimates of evapotranspiration (ET) and another on feedback from soil moisture sensors, applied in turf. The ET is estimated based on recorded weather data thanks to a weather station on the site. The study tries to

understand which system results in the best combination of water efficiency and turf quality.

The study concludes that the "on-demand" sensor-based system presents the best results. It means that sensor readings are best suited for water control than weather data, even if the weather data is specific from the plantation site. However, as it also states, it is the most expensive one. It is also worth pointing out that the researchers are not using a valuable perk of weather, which the sensors do not have: forecasts. Though sensor-based systems can effectively control on the present moment the water level on a soil, it cannot predict how the water level will evolve in the future. Weather forecasts can tell the control system that watering is not required now, though soil water availability is starting to decrease from the ideal point, because in a few hours it will start raining.

On the contrary, evapotranspiration models can never measure water availability with the same precision as on-site sensors. An ideal system would have sensor reading inputs, as well as real time weather conditions and weather forecasts. Due to its cost, it must be established if investing on a system with on-site sensors is worth it, i.e, if sensors give enough contribution to an evapotranspiration and forecast based model that justifies the investment. In conclusion, there is definitely an interest regarding weather and evapotranspiration use in efficient farming, and a justification for that.

2.4 Other Works

2.4.1 Control over Moisture Probes and Live Weather Data

Interestingly enough, a not so recent research[18] from University of Boumerdes, Algeria and University of Versailles S-Q, France presents an automation solution of a multi-mode control for an irrigation system, using Windows XP operative system on a computer. Although out of our box of interest concerning the chosen controller technologies, it is worth mentioning for a few reasons. First of all, it shows that automation of the irrigation process has been a concern for many years. Second, the proposed system autonomously manages the irrigation process using the current weather conditions and soil moisture probes.

Using these input parameters, the system tries to solve the following contemporary problems:

- It prevents irrigation during occasional raining periods, thanks to live weather conditions monitoring precisely on the irrigation sector.
- Using two moisture probes, a start probe and a stop probe, the system does not allow under-watering or over-watering.

Weather parameters are also interesting, not only to spot current rain, but to decide whether or not the current conditions are suited for an irrigation. Although not part of the mentioned study, these parameters could, for instance, tell the system not to water if the wind is reaching great speeds, which can drastically reduce irrigation efficiency. This is particularly interesting when dealing with sprinklers or irrigation pivots.

Of course, live weather monitoring has its limitations. If a given sector is watered half an hour every day at 9 a.m., but on a given occasion it will rain from 10 a.m. to 11 a.m., the irrigation of that day will be useless. This means that, though live weather monitoring is key for preventing watering when it is raining, weather predictions could improve significantly the autonomous model. The prediction and planning part of the efficient irrigation is still lacking.

2.4.2 3DEP and NDVI

On december 2016, the U.S. Department of the Interior issued a fact sheet [19] presenting the benefits of the 3D Elevation Program (3DEP) on precision agriculture and other farm practices.

> "The 3D Elevation Program (3DEP) (...) provides the programmatic infrastructure to generate and supply superior terrain data to the agriculture industry, thereby reducing costs and risks, which would allow farms to refine agricultural practices and produce crops more efficiently. The 3DEP infrastructure uses data acquisition partnerships that leverage funding;

> develop contracts with experienced private mapping firms; capitalize on technical expertise; maintain first-tier lidar data standards and specifications; and, most importantly, provide public access to high-quality 3D elevation data".

This program developed models for improving the precision of soil surveys, displaying all physical characteristics of a given field: slopes, elevations, soil patterns. These field features impact irrigation efficiency. For instance, soil water tends to travel to lower parts of a field, which will reduce irrigation efficiency on elevated parts of a given plantation. Similarly, a hillside, if facing a certain direction, can either receive much more solar radiation than the rest of the field or it can be too deprived of it, which affects how much water or fertilizer is needed. Knowing *a priori* the field characteristics can contribute to a more smart and efficient irrigation decision.

Satellite imaging can contribute to agriculture with lots of added vision features. A common index that analysts use in remote sensing is the Normalized Difference Vegetation Index, or NDVI. GISGeography [20] defines this parameter the following way:

> "Normalized Difference Vegetation Index (NDVI) quantifies vegetation by measuring the difference between near-infrared (which vegetation strongly reflects) and red light (which vegetation absorbs).
> NDVI always ranges from -1 to +1. But there isn't a distinct boundary for each type of land cover.
> For example, when you have negative values, it's highly likely that it's water. On the other hand, if you have a NDVI value close to +1, there's a high possibility that it's dense green leaves.
> But when NDVI is close to zero, there isn't green leaves and it could even be an urbanized area."

NDVI can be used to verify crop growth across a field, detect unhealthy or infested plantations, and even pump leakages or pressure drops (which indeed reflect on the crop). This information is available for free, although it is dependent on the frequency with which the satellites pass over the fields.

2.4.3 ANN based Controller

As a last footnote, it should be mentioned the use of Artificial Neural Networks (ANN) on irrigation systems. A 2010 study[21] from the National University of Science and Technology, Pakistan shows the application of this technology on a controller for automation of an irrigation system. The research proposes a closed loop controller, with fixed and variable inputs. Some of the fixed parameters include:

- Type of soil
- Type of plants
- Leaf coverage
- Stage of growth

Some of the variable parameters include:

- Soil humidity
- Air humidity
- Radiation in the ground
- Temperature

The system computes when to water and how much it should water, as well as whether it should or not open/close walls and roofs of hothouses. Once again, evapotranspiration is calculated, and a distributed time delay neural network is used to perform dynamic decisions.

Although not many specifications are given concerning the ANN, it is definitely a worth-mentioning approach for decision making procedures. The great advantage of the ANN method is that the system can be trained *a priori*, just as training a young farmer how to do its job.

2.4.4 Comparison of scheduling algorithms for drip-irrigated apple trees

An article [22] presented in 2016 from the Center for Precision Automated Agricultural Systems compares 7 different scheduling algorithms using an automatic irrigation control system equipped with a variety of sensors. The authors conclude that algorithmic approaches which are only based on soil characteristics present worse performance than the ones which account for weather conditions and crop variables. They also conclude that, even though plant feedback presents the best results regarding rescheduling the irrigation procedure, weather-based approaches are to be preferred when the price of the solution enters in the equation.

One of the algorithms evaluated in this paper is a "soil water budget model and a temperature-only-based ET [evapotranspiration] equation". Seemingly, it is the algorithm which presents more common features with the solution that is presented in this thesis. The decision system would compute the evapotranspiration and rainfall of the day, and decide on a specific moment whether to add that difference into the crop or not. Evidently, it only relates to a single water balance, i.e. one crop soil water level, showing as well a quite smaller decision granularity than the one which this thesis' algorithm thrives to achieve.

Using a $Watermark^{TM}$ sensor, the article provides another algorithm which, combined with the previous one, leads to an automatic irrigation system with real-time decision on the irrigation of a crop. The decision making follows the presented flowchart:

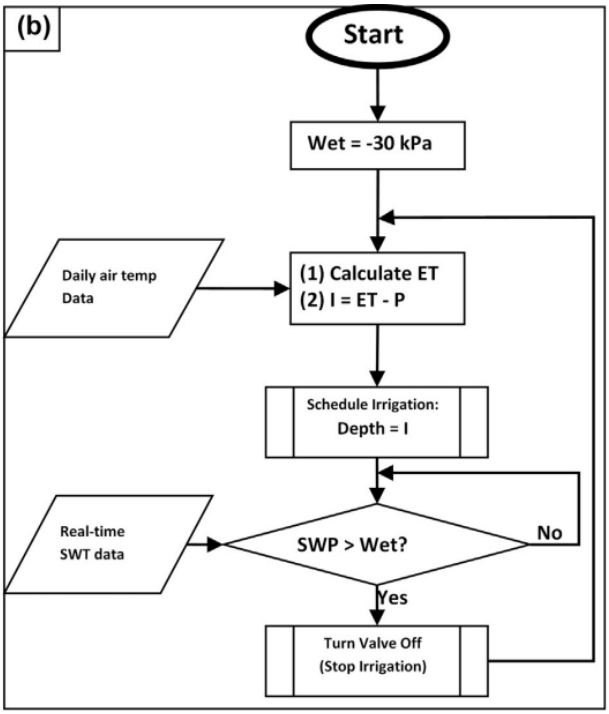

Figure 2: ET + WB algorithm [22]

Indeed, with this sensor, real-time soil water level is provided as feedback to the system. In the present thesis, there is not the question of feedback to the system. Indeed, an algorithm is presented to provide a weekly irrigation plan. The planning is accomplished in a single moment, given all necessary variables. If any automatic control system were to use the algorithm, feedback could be given to it in the form of corrections to the current water balance levels, followed by a rescheduling moment. Whether that is in fact used by a control system or not, it is irrelevant for the problem in hand. It remains, however, noted that this thesis' algorithm can be used with feedback from the system, whether using sensors or other types of estimations. Finally, the article's algorithm only solves a very narrow problem, without leveraging the complexity of distributing water pump resources, energy costs or fertilization needs.

2.4.5 Irrigation Water Use Efficiency

As technology evolves, so does the awareness of a great opportunity to save resources. A 2018 article [23] from *Water* journal provides a summary on various opportunities on the field, as well as challenges, looking specifically to the Australian context, although relevant for all parts of the world. Once again, it is approached the idea of evapotranspiration as a means to track crop water level, as well as the use of sensors.

One of the article's sections tackles the growing of irrigation scheduling methods for water use efficiency. However, it is pointed out that "[t]he major drawback of using these soil-moisture-based tools for scheduling is that they give point-based measurements, while the soil characteristics are known to vary spatially and temporally". This is certainly true when one only looks at the current weather parameters (i.e. current evapotranspiration) or current sensor readings, or even past measurements. However, the drawback is bridged if predictions can be provided, such as weather forecasts, from which one can derive evapotranspiration predictions. This entails that the opportunity of prediction based on weather forecast is not fully grasped in the article.

Furthermore, the article suggests that current irrigation scheduling systems are limited regarding their adoption "because of reasons ranging from complexity to cost". With this, the authors entail that most irrigation scheduling solutions are sensor-based, which also indicates a real-time decision making rather than a behaviour prediction. This is also strongly implicit in the fact that the authors only name scheduling systems which are indeed based on sensor monitoring.

Curiously enough, though the article tackles various opportunities to increase water use efficiency, there seems to be no references to the possibility of saving water resources through intelligent water pump usage. Indeed, the aggregation of sectors which are to be simultaneously irrigated by a single water pump is a problem that every farmer needs to surpass. The algorithm presented by this thesis aims at exploring this problem in order to save water resources as well as energy costs.

2.4.6 Multi-Crop Planning using Particle Swarm Optimization

A 2016 article[24] explored the approach of Particle Swarm Optimization (PSO) on multi-crop planning (MCP) on cultures under deficit irrigation. The model thrives to allocate water in the various crops within certain production and resource constraints. A PSO algorithm is a kind of Evolutionary Algorithms, similar to a genetic algorithm, which is used in this context to relax the non-linear problem into a linear one (LP).

In the selected approach, soil water balance is not tracked, as it is argued that the evapotranspiration ratio will give the algorithm the necessary data for calculation of needed water supply. Indeed, the article does not try to plan how an actual irrigation procedure takes place in time and throughout the weeks, rather it generally assesses the right distribution of crop production in order to achieve the highest possible profit, considered certain general constraints regarding crop area, water availability and crop evapotranspiration. This can be seen by some of the constraints that are chosen for the problem in question:

- Water availability at each stage of the crop growth: in the relaxation case, this constraint is despised.
- Under deficit irrigation the supplied water cannot pass a given value: this is similar to providing a maximum value of water capacity to the crop's soil.

The article in question uses an Evolutionary Algorithm to relax a non-linear problem where general crop needs are present, as well as the objective of increasing profit. In these aspects, it approximates to the problem addressed by this thesis. However, the angle of approach is entirely different, inasmuch as the thesis achieves more profit mainly by arranging crop in a given area and selecting the best products for the general constraints it has, not going in much detail regarding crop growth and avoiding the problem of the irrigation network complexity. Rather this thesis aims at increasing profit in an already settled irrigation setup, by exploring all main resources in the most efficient way possible, thus drastically reducing costs and resource waste. Still, as already stated, the algorithmic approach generally plans water allocation

for each crop, which is something in common with the algorithm presented in this thesis, though that will be achieved and planned with the greatest possible granularity.

2.4.7 Application of Genetic Algorithms for Irrigation Water Scheduling

In 2009, a doctorate's thesis[25] presented by Zia Ul Haq at the University of Southampton explored the possible approach of genetic algorithm to the problem of water scheduling in demanding irrigation systems. These are systems which provide water to a number of farmers which want to irrigate at different hours and having different water needs. The author develops 8 different models to resolve increasingly complex problems. The last model tackles:

- possible simultaneous irrigation
- different discharges per farmer
- water travel to the outlets

An emphasis will be given to this model, not only because it's one of the most complex ones, but also because it proves to be the most efficient one. The author uses a genetic algorithm and the *Stream Tube Approach* (STA), after concluding that integer programming is not a practical solution due to its time complexity. The STA model is formulated so that the number of demanding simultaneous machines does not exceed a certain limit and jobs are scheduled as close as possible to the desired start time. To the problem in hand, it means that each stream tube supplies to one outlet at a time. The algorithm uses vectors and integer values to simplify operations. The chromosomes, i.e. the main unit of the problem is given by the concatenation of two numerical vectors as follows:

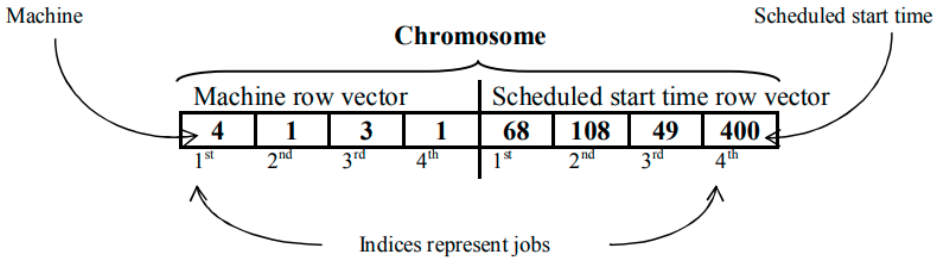

Figure 3: Chromosomal representation of the main algorithm element. [25]

The time complexity of this solution was far better than the other ones. Indeed, for 12 outlets, this genetic algorithm took 48 seconds to execute, while the integer programming option took 133 minutes and the other genetic algorithm 17 minutes.

Although this problem is different from the one approached in the present thesis, it does have some similarities. One has the same issue regarding delivery to a number of outlets with different demands, and a maximum capacity to feed the system. In the case of the present thesis, those outlets are not farmers but various different valves/sectors of one same irrigation network, of whose debits vary between them and of which irrigation is also bound by the capacity of the water pump. Contrary to the author's problem, the present thesis' problem does not have to deliver water to a given outlet at a given time, or at least relatively close to it. However, irrigation of the same sector can happen, more than once per plan, as often does when dealing with a week plan. Furthermore, though the genetic algorithm provides a solution to the problem, that same solution's time size is quite smaller compared to the present thesis' problem. Indeed, the author shows examples where the decision spans a full week. However, the span complexity of a week, for this given problem, does not increase when going to an interval of minutes, or hours, or months, because the irrigation supplied to a machine (farmer) only happens one during the entire time-frame and uninterruptedly. That is, as seen, a necessary characteristic of the solution. But one cannot ignore that this particular feature simplifies the process compared to the problem addressed by this thesis.

3 Background - Algorithms

Artificial intelligence algorithms are not just something of the last decade. Indeed, the technological revolution we are facing in these modern times is enabling the creation and development of a great variety of intelligent systems, able to not only replace the human being on monotonous chores, but also to assist him on more complex tasks, even finding solutions for problems that a human mind would not be able to solve. A 10 year old kid does not question how in the world he is able to play chess against his smartphone, and a 30 year old adult does not even blink when Google Maps provides him in a matter of seconds with the fastest way to go from Lisbon to Madrid while simultaneously avoiding any tolls, which would perhaps take hours for a single person to come up with a close solution, maybe not taking traffic into account. As submerged as we are in this world of intelligent systems, one can miss the fact that these sort of algorithms took about a century to reach to the point where we are now.

A brief review on optimization algorithms will be presented, pointing out only what seems to be relevant to the problem addressed by this thesis.

Using such algorithms serves the purpose of reducing memory and/or time complexity. It is evident that a solution to an optimization problem can be found if one goes through the spectrum of all possible solutions, comparing them and choosing the one which presents the smallest cost function value possible. This can be called the "brute force" method. It is the same approach of a baby trying to fit a form one by one into a hole with a given shape. However, for most optimization problems this is something practically unfeasible, not to mention unintelligent.

If one were to "brute force" its way into a digital file with a 4-numbers pin password, it would not take that much time to find the desired password among the 10.000 possible ones, using a computer program, and even if it'd take a few minutes, one probably wouldn't mind waiting to reach the desired goal. However, a 10-numbers pin password would take much more time, not to mention if the pin could have any existing character as a possibility for each digit. Considering the energy-only model with a 1440-minutes day and only 5 irrigation sectors, each with a desired irrigation time of 20 minutes, this would correspond to a set of $5.2913 * 10^{44}$ possible solutions, assuming the irrigation pump can irrigate the 5 sectors at the same time and already discarding the solutions where the 5 sectors are not being simultaneously irrigated. Even if it were to be a feasible way of solving the problem in hand, we are not looking for such a method, but rather a fast and intelligent strategy of finding, within seconds and for much more complex situations, the quasi-optimal, or even optimal, solution.

We now move on to a list of already existing algorithms that were considered relevant for this thesis, explaining how they contribute in an algorithmic brainstorm for the problems in hand.

3.1 Packing Algorithms

A possible approach to the problem concerning this thesis is through dynamic programming methods. One can see that the algorithm for the irrigation plan formulation will forcibly have the insertion of sectors in time slots throughout a given time interval. Each time slot cannot surpass the maximum irrigation capacity of the pumping pump. This means that each time slot has a maximum of sectors that fit inside it, and that maximum is given by the irrigation pump. One sees that, in the majority of cases:

- The pumping pump cannot irrigate all existing sectors on a given time slot.
- The valves of some sectors may debit enough water as to work the pumping pump to its maximum capacity, meaning that a sector as such must be irrigated separately from all other sectors.
- Some sectors may fit a given time slot together as to reach the pumping pump working sweet spot, while others may not fit well with each others and result in a time slot where the pumping pump is working well below its maximum capacity (this is not optimal, energetically speaking).

For now, let us assume the problem of the present thesis is as simple as that. It appears to be a problem where resources must be split as best as possible, and organized in "boxes" in such a way as to have the least amount of boxes possible, i.e. as filled as they can be.

3.1.1 Bin packing - general algorithms

The Bin Packing Problem is yet another common dynamic programming target. It is a problem of combinatorial optimization, presenting a different abstraction to a problem which is very similar to the previously mentioned one. The Bin Packing Problem can explore more than one dimension, but the focus will be on the one-dimensional problem. "Given a set of numbers and a set of bins of fixed capacity, the problem is to find the minimum number of bins needed to contain all of the numbers" [26]. One can think of it as many cylindrical boxes (or bins) with the same base and height, and a variety of different cylindrical batteries, with the same base as the boxes but with smaller and different heights, that have to be jammed inside the least amount of boxes possible.

For better illustration, let us consider the example of a battery shipper, who gets paid by the amount of batteries it ships, but the cost of the bins is discounted on the monthly salary. Evidently, the goal here is to have the least amount of empty spaces in the bins, thus having more compact bins, which leads to a lesser number of boxes for a given number of batteries. We can split possible algorithms into two types of problem:

1. The batteries appear one by one, meaning that each must be inserted into some place without knowing about the remaining batteries to come.
2. The shipper has from the beginning access to all the existing batteries that need to be jammed into the least amount of bins.

The problem addressed by this thesis fits into the second category of situations. However, the algorithms for the first category problems are an introduction to more complex ones. It must be noted that it is difficult to guarantee the best solution for the Bin Packing Problem without compromises in time complexity. This means that sometimes the most popular approaches may be rather simple to save time and guarantee not an optimal solution but a satisfactory one.

3.1.1.1 Next Fit

The Next Fit approach consists on a simple test of whether the item finds a space on the current bin or not. If it does not fit, that bin will be declared as closed, and a new bin will be used for the item. Thus, the following item will have a new bin to test, skipping the bins that were declared closed. This is a simple algorithm with O(n) time complexity, where n is the number of items.

An example illustrates it better. The shipper has now 4 batteries with sizes $[4, 7, 5, 6]$ and size 10 cylindrical bins. The shipper will place the size 4 battery in the first bin. The size 7 battery does not fit in the first bin, so this bin will be declared as closed, and the present battery will be placed in the second bin. The same will happen to the other two batteries, so in the end the shipper will have 4 bins, each with one battery.

Given that the items of every two consecutive bins will sum up to more than the quantity of one full bin (if this was not as such, the item would not have passed to the next bin due to not fitting the first one), one can conclude that this algorithm will use at most twice the number of bins of the optimal solution (one without wasted space, though in certain cases it may not be possible).

Although this algorithm presents very simple time complexity, it is not an intelligent strategy to reach optimality, rather a way of reaching some solution. Indeed, one sees in the previous example that size 6 battery would have fit into the first bin with the size 4 battery, thus reaching optimality in this simple case.

The next algorithm presents a great advantage without being much complex.

3.1.1.2 First Fit

The First Fit approach is similar to the Next Fit algorithm, with the difference that no bins will be declared as closed. Considering the example given for the Next Fit approach, the shipper now will not declare any bins as closed. This means that size 6 battery will be tested for fitting the first and second

bin. It does not fit any, so it will be placed on a third bin. We thus reach the optimal number of bins for this rather simple problem.

The time complexity increases in this algorithm in comparison to the previous one. Indeed, its time complexity increases with the square of the number of items to insert, $O(n^2)$. However, this increased complexity is a compromise that is definitely worth it, given the great advantage it presents on the quality of the solution found. Contrary to the Next Fit approach, it can be said to be an intelligent strategy.

The First Fit approach still has lots of limitations, that are not present in the previous example. There is a problem that arises when the strategy consists on inserting the item on the first bin that fits the given item. The problem is that the selected bin may not be the more intelligent place to insert the item in hand.

Let's say the shipper now has 4 batteries of sizes $[7, 8, 2, 3]$. Evidently, this is another very simple problem where we can immediately see the way to achieve the optimal solution. But in this section the algorithms still do not have access to the entire set of items in the beginning of the problem. Using the First Fit strategy, the shipper would insert the size 7 battery into the first bin. The size 8 battery does not fit the first bin, so it would be placed in a second bin. The size 2 battery fits the first bin, so there is where the shipper would place it. The size 3 battery now does not fit the first bin (it only has 1 space left), and it does not fit the second bin either, so it would be inserted in a third and final bin. One can easily see that it was not well-thought to insert the size 2 battery with the size 7 battery. Indeed, if the size 2 battery were to have been inserted in the second bin together with the size 8 battery, the shipper would have been able to fit the size 3 battery in the first bin together with the size 7 battery, thus reaching the optimal solution of 2 completely filled bins.

The next algorithm aims to solving the First Fit problem that was previously illustrated.

3.1.1.3 Best Fit

The Best Fit approach, instead of placing a given item inside the first possible bin, it will see what bin among the currently used will leave the least amount of space if the given item is to be inserted there. This strategy has the advantage of always finding, if it exists, the bin that will be completely filled with the insertion of the item in hand. This resolves the problem of the previous example. Indeed, the size 2 battery would have been paired up with the size 8 battery and the size 3 battery with the size 7 battery. The time complexity of this algorithm is the same as the First Fit approach, $O(n^2)$.

There are yet some issues that bind this new strategy. First of all, this or any other algorithm that does not have access to the entire set in the beginning of the problem will always have disadvantages regarding an algorithm with that knowledge. Given that the problem addressed by this thesis has access to all said items (irrigation sectors) in the beginning, the use of any of these algorithms will not explore that advantage. Furthermore, and mostly due to this aforementioned issue, contrary to what seems to be at first glance, the Best Fit strategy does not always result in the best combination of items per bin.

Let us consider yet another initial set for the untiring shipper of all these examples. The shipper has now a set of 5 batteries of sizes $[2, 3, 4, 5, 6]$. Looking at the entire set, one sees that the optimal solution will combine the batteries in order to use two bins, one with sizes $[2, 3, 5]$ and another with sizes $[4, 6]$. Instead of these combinations, one sees that, if the shipper is to opt with the Best Fit approach, it will insert the batteries of sizes $[2, 3, 4]$ into the first bin. The size 5 battery will go to a second bin and the size 6 battery will go to a third bin.

Here is where the Bin Packing Problem finds itself with a big combinatorial problem. Even if the algorithm were to have access to the entire set of items in the beginning, which is by itself an advantage, if the algorithm intends to find a better solution than the Best Fit one, it is going to face the task of finding the best possible combination of items to fit the bins. It seems that finding the optimal solution is only possible by testing almost the entire set of possible combinations of items per bin, which has a complexity that grows exponentially with the number of items. However, it is possible to enhance the previous algorithms in order to find an almost optimal solution, without great impact on time complexity,

though increasing it is inevitable.

The Bin Packing Problem seems specially interesting for the problem addressed by this thesis. Indeed, the time slots behave precisely as bins to the irrigation sectors that need to be distributed, and time slots have the size correspondent to the pumping pump's maximum capacity. Distributing the irrigation of sectors through time slots, as items through bins, seems to be the right strategy for finding the solution that minimizes the energy consumption: less time spending energy. The energy does not have the same price in every time slot, but to guarantee the minimum energy consumption possible one simply has to fill bins starting by the cheapest bins, all the way to the more expensive ones. It is evident that this strategy still does not account for the water balance of each sector throughout a week. That more complex problem will be addressed later.

The algorithms that are to be addressed next are variations to these general algorithms that were presented.

3.1.2 Bin packing - genetic algorithms

Genetic algorithms consist on imitating the biological evolution of beings in a much faster rhythm. "Genetic programming is a systematic method for getting computers to automatically solve a problem starting from a high-level statement of what needs to be done. Genetic programming is a domain-independent method that genetically breeds a population of computer programs to solve a problem. Specifically, genetic programming iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations. (...) The genetic operations include crossover (sexual recombination), mutation, reproduction, gene duplication, and gene deletion" [27].

The first algorithm of this kind to be presented is a simple algorithm that expands from the First Fit approach, presented on Jordan Junkermeier's paper *A Genetic Algorithm for the Bin Packing Problem* [26]. A more succinct explanation of what a genetic algorithm consists on is presented there: "Genetic algorithms are heuristics based on biological evolution that simulate reproduction with variation and selection according to fitness, like that of a true biological population" [26]. It also shows the general logic structure of a genetic algorithm, in the following image:

```
Generate random initial population;
While (not done)
{
     For i=1 to population size
     {
          Select two parents;
          Crossover to produce an offspring;
          Mutate the offspring;
          Insert offspring into new generation;
     }

     Offspring replace parents;
     Report the best solution in the population;
}

Report the best overall solution;
```

Figure 4: The general structure of a genetic algorithm, by Jordan Junkermeier. [26]

Specifically for the Bin Packing Problem, in a genetic algorithm, "each candidate solution can be represented by a permutation of the elements in the input set" [26]. It will be assumed that the shipper that has been accompanying the exposition of these algorithms has now access to the entire set of items from the beginning of the problem.

3.1.2.1 First Fit Genetic Algorithm, by Jordan Junkermeier [26]

In the presented algorithm, each random permutation will be accomplished by applying the First Fit strategy to the input set. Given that it is not possible to alter the values inside the set of items, "only the order of the elements in the input set will differ", meaning that the new generated solutions come from applying the First Fit algorithm on new randomly shuffled input sets. The author of this paper defines a parameter called "fitness", which is adequate for the biological evolution metaphorical abstraction. For each input set, the fitness will correspond to "the number of bins needed to hold the elements in the input set". The author also names the combination of the input set ("permutation" of genes) and its fitness a "chromosome". The objective of the genetic algorithm presented here is to minimize fitness, because "chromosomes with smaller fitnesses are better solutions". Strangely, it seems counter-intuitive that in biological evolution the chromosomes with worse fitness are the ones surviving against the ones with more fitness.

The algorithm begins with two "parents", i.e., two sets of items randomly shuffled. The selection of parents is made by selecting two random chromosomes from the population, and comparing the fitness of both selected chromosomes, each of them computed in this case with the First Fit algorithm. The one with the lowest fitness will be the first parent. The process is repeated with another two randomly selected chromosomes to get the second parent.

The author chooses various parameters that characterize the algorithm:

- Population size: **100**
- Number of generations: **100**
- K-tournament k: **2**
- Probability of crossover: **100%**
- Probability of mutation: **10%**

Figure 5: Chosen parameter values for the genetic algorithm, by Jordan Junkermeier.[26]

The crossover between two "parents" is accomplished here by adding to the "offspring" the first element of the first "parent" chromosome, and then adding the first of the second "parent", then the second element of the first "parent", then the second element of the second "parent", and so on, ignoring the duplicate elements, thus achieving an "offspring" chromosome with the same elements as the parents (relevant for this problem), only with a different arrangement [28]. It should be noted that, if we were to apply this algorithm to the previous example of the battery shipper, two size 2 batteries would have to count as two different elements, thus not allowing element repetition but at the same time enabling the possibility of having same size elements.

After crossover, the resulting set, i.e. the "offspring" chromosome, must pass by a "mutation" to insert variation in the population. The author's mutation will consist on selecting two random elements, or "genes", of the chromosome and switching their positions in the set.

Finally, the "offspring" chromosome will be inserted in a list of new generation chromosomes, which will replace the older generation population. The author chose a population size of 100 chromosomes, and 100 generations. So, in every iteration of the algorithm, 100 "offspring" chromosomes will be generated. The chromosome with the best fitness (i.e. the minimal fitness) of this new generation will be stored if its fitness value is smaller than the best chromosome of the previous generation. And that's it.

The author tested the algorithm in three different sets, varying also in the size of the set and the capacity of the bins. Although, as the author concludes, "the genetic algorithm managed to produced superior solutions to those of the *first-fit* algorithm for two-thirds of the problem instances", the improvement rates are extraordinary small. Indeed, the best case scenario only finds a way of saving one more bin than the simple First Fit algorithm, while increasing substantially the time complexity. For a population of 500 size chromosomes, being distributed along 1500 size bins, the presented genetic algorithm only managed to decrease the number of bins from 134 to 133 in the B and C sets, and it got the same fitness in the A set.

Although the results are not that impressive, one cannot simply ignore the merit of this paper. The use of the genetic algorithm, although a probabilistic method (but uncertainties may be reduced with the number of generations and mutations, it is a question of finding the right balance between decreasing uncertainty and decreasing time complexity), seems to have potential. The idea of reordering the item set is also an interesting way of trying to find a better solution, while remaining with the simple First Fit approach. One wonders as well if the genetic algorithm would work with better results if the Best Fit approach were to be used instead of the First Fit strategy. Time complexity may in fact be similar, and the result seems to promise more. However, some suggest that the Best Fit strategy is just "a seemingly better heuristic, which can, however, be shown to perform as well (as bad) as the FF [First Fit], while being slower" [29].

There is another issue with this and other such genetic algorithms, and that is the simplistic fitness cost function. The author uses the number of used bins of a solution as the fitness of a chromosome. Indeed, there is no doubt that in a Bin Packing Problem one desires to minimize the number of used bins. That is actually the whole purpose, and the great struggle of this problem. However, in genetic algorithms the fitness parameter is far more important than the mere solution value, which it is in this case. The fitness of the chromosome is used by the genetic algorithm to evaluate the aptitude of the mentioned chromosome in its environment. Survival of the fittest indeed. But in the search place of the bin packing solving algorithm, one should look into more chromosome features than just the value of bins in use, because those features may be key to be passed on to the next generation.

Because of this, there is a cost function that is commonly used in such problems, proposed by Falkenauer and Delchambre [30], represented below:

$$f_{BPP} = \frac{\sum_{i=1,N}(F_i/C)^k}{N} \tag{1}$$

The parameters of this function are [29]:

- N - number of bins used in the solution.
- F_i - sum of sizes of the items in the bin i (i.e. the fill of the bin)
- C - the bin capacity
- k - constant greater than 1, expressing the concentration on the most filled bins in comparison to the less filled ones

The following algorithm uses this cost function as the fitness guiding the search, and it is a hybrid genetic algorithm that was proposed as an approach of energy minimization on virtual machines placement in cloud.

3.1.2.2 Hybrid Genetic Algorithm using Best Fit Decreasing approach for infeasible solutions, by Mohamed Kaaouache and Sadok Bouamama [31]

Mohamed Kaaouache and Sadok Bouamama do not just merit for the algorithm they presented on their paper, but also for their satisfactory analysis of rather important genetic algorithms proposed for the Bin Packing Problem throughout the history. It is worth mentioning at the very beginning that this paper does not show sufficient evidence that the presented algorithm behaves well when dealing with large sets of items, which renders it unsuitable for the problem addressed by this thesis. However, it has various interesting features. But first of all, let's briefly expose what is the Best Fit Decreasing approach.

The Best Fit algorithm has been previously explained. It consists on testing how each item fits in all non empty bins, and inserting it in the bin where it will leave the least amount of empty space, and a new bin is selected whenever the item does not fit any non empty bin. This simple algorithm does not have access to the entire set of items in the beginning of recursive insertions. The Best Fit Decreasing approach, however, has this knowledge, and uses it to order the selection of items: the smallest item of the remaining set is the one that is picked for packing. This is similar to the First Fit Decreasing strategy, which also arranges the items in the same fashion, varying only in the way of packing per item.

Although the Best Fit Decreasing approach does not improve the solution in all cases, it does prove to be more efficient than the plane Best Fit strategy in the majority of cases if one were to randomly initialize various different sets of items and apply both algorithms.

Among the algorithms named by the authors, there are two rather important ones: Martello and Toth's branch-and-bound procedure (MTP), and Falkenauer's hybrid grouping genetic algorithm. The first one consists on an heuristic and exact approach with the key feature of reducing the search space with recourse to a so called "dominance criterion" [32]. The second one "uses a special encoding scheme for groups of items and a local optimization heuristic inspired by the dominance criterion of Martello and Toth" [31]. Falkenauer's algorithm wins over the MTP algorithm when the set of items is large, both in the optimality of the solution and in its performance time. "The superiority of the hybrid GGA over the MTP procedure of Martello and Toth was confirmed by an extensive experimental comparison" [29]. For this reason, the MTP algorithm will not be addressed. Falkenauer's approach to the bin packing problem will be later examined.

In the algorithm that is being exposed, the set of items is arranged in the same exact manner throughout the entire process, and bins are the ones that change order: "we are moving bins instead of moving items" [31]. This is a feature that is not present in any of the previously exposed Bin Packing algorithms. This bin oriented scheme means that chromosomes refer to the same item each time, which leads to infeasible chromosome solutions, i.e., chromosomes that exceed the bin's capacity. The Best Fit Decreasing is used to solve these infeasible chromosomes, by packing those items with that strategy.

Unfortunately, the authors don't enter in much more detail concerning the core of the algorithm that is used in the paper, rather they are more interested in explaining the approach they use for solving infeasible chromosomes. This approach seems to make the bin oriented scheme a viable one. However, as Falkenauer mentions and the authors warn, this representation has more drawbacks than illegal solutions. Indeed, the bin oriented scheme creates a problem of redundancy, "which exponentially increases with the number of bins". Although the authors also suggest that this problem does not seem sufficiently backed up by much experimental results, it still is a critical issue, adding to the fact that they do not present themselves any results for large sets of items, as previously stated.

Concluding the exposition on genetic algorithms, Falkenauer's already mentioned strategy of grouping is presented next. Interestingly, this author also proposes a sort of bin oriented scheme, but combined with a way of significantly reducing the space of solutions.

3.1.2.3 Hybrid Grouping Genetic Algorithm, by Falkenauer [29]

Falkenauer observes the fact that the Bin Packing Problem is a type of Grouping Problems, where "the aim is to find a good partition of a set, or to *group* together the members of the set". The author defends that a grouping genetic algorithm is more suited than the normal genetic algorithm type for Grouping Problems. Great differences between the two appear on the encoding of the chromosomes, which now have genes that are themselves groups of items, and special operators for this new type of chromosomes. Falkenauer observes that "the structure of the simple chromosomes (...) is *item* oriented, instead of being *group* oriented. (...) Indeed, the cost function of a grouping problem depends on the *groups*, but there is no structural counterpart for them in the chromosomes above [i.e. the simple ones]".

Each chromosome represents the solution to the Bin Packing Problem represented in groups of items per bin. The example of the author's paper will be given.

Bins	A	B	C	D	E
Items	0	2, 5	3	1	4

The chromosome can, thus, be written as $\{0\}\{2,5\}\{3\}\{1\}\{4\}$. This representation means also that the genetic operators must be able to "handle chromosomes of *variable length*". Using the groups as genes means that the algorithm pays attention to groups of items as them being the "meaningful building blocks" of the Bin Packing Problem, rather than an item by itself. Given that "the very idea behind

the GA paradigm is to perform an *exploration* of the search space, so that promising regions are identified, together with an *exploitation* of the information thus gathered, by an increased search effort in those regions", the algorithm allows for the building blocks of this Grouping Problem to be exploited, "i.e. transmitted from parents to offspring, thus allowing a continuous search in their surrounding". This encoding scheme is crucial for the performance of the presented strategy.

For chromosomes with variable length, the crossover part of the genetic algorithm presents itself as a major problem. Indeed, given the fact that the hard constraints and the cost function vary among different grouping problems, the ways groups can be combined without producing invalid or too bad individuals are not the same for all those problems". This means that the crossover procedure will vary with the chromosomes that are taking part of it. The author presents the crossover pattern of a Grouping Genetic algorithm for the two parents in the following manner:

1. Select two random sites in each parent, which will delimit the *crossing section*.
2. *Inject* the *groups* of the *crossing section* of the first parent to the *crossing section* of the second parent.
3. Eliminate repeating *items*, by removing the second occurrence from the groups that were members of the second parent.
4. "If necessary, *adapt* the resulting groups, according to the hard constraints and the cost function to optimize. At this stage, local problem-dependent heuristics can be applied". This point will be explored later on.
5. Repeat points 2 through 4 to the same parents but with their roles permuted, thus originating a second child.

The Grouping Genetic algorithm also uses an inversion operator. This operator switches genes which can result in good genes being close to each other and thus being more probable of both being passed on to the offspring.

Falkenauer proposes an algorithm of local optimization, "producing high-quality bins which are then efficiently processed by the GGA [Grouping Genetic algorithm]", which is inspired by the Dominance Criterion proposed by Martello and Toth, on their aforementioned MTP algorithm for Bin Packing problems [33]. The criterion is based on a "simple yet powerful observation", as Falkenauer states. Indeed, it can be illustrated in following image, taken from the paper being exposed.

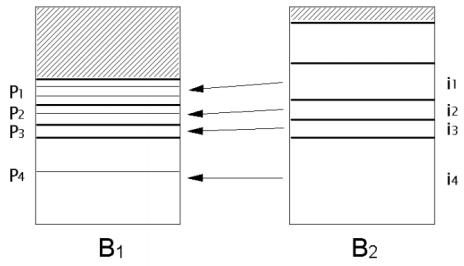

Figure 6: The Dominance Criterion: B_2 dominates B_1.

The mentioned observation is the following: "given the contents of two bins B_1 and B_2, if there exists a subset $\{i_1, ...i_n\}$ of items in B_2 and a partition $\{P_1, ...P_n\}$ of items in B_1 such that for each item i_1 there is a no bigger corresponding P_i, then B_2 is said to *dominate* B_1, because a solution obtained with B_2 as one of the bins requires no more bins than one with B_1". The usual procedure of a Bin Packing algorithm, as the author states, "would repeatedly find a bin dominating *all* others, add that bin to the solution and *reduce* the problem by removing the items just assigned. However, that procedure would run in exponential time. In order to obtain a procedure of a reasonable complexity, [Martello and Toth, 90a] propose to consider only sets of size three or less". And that is how the searching for the dominating

bins does not run in exponential time, by fixing lower bounds. Approximate algorithms can *relax* these bounds, as Falkenauer continues to explain. It remains to explain how the mentioned *local optimization* will be accomplished in this Hybrid Grouping Genetic algorithm. This will be exposed within the more detailed explanation of the chromosomes' crossover method.

The fitness is computed by the aforementioned Falkenauer and Delchambre's cost function. The crossover method was already explained in 5 points. However, point number 4 promised clarifications, and the others have subtle differences. Indeed, after selecting two random sites in each parent to delimit *crossing sections*, the first parent's *crossing section*, i.e. the bins inside that part of the chromosome, is injected in the *first* crossing site of the second parent. Note that the same procedure will be applied later but with the parents' roles permuted. After this, the repeated items of the first parent will be eliminated. To accomplish this, the crossover method removes the original first parent's bins which contain items that are present in the bins injected from the second parent. As one can already conclude, "[w]ith the elimination of those three bins we have, however, most probably eliminated items which were not injected (...). Those items are thus missing from the solution". Here is where a *local optimization* takes place. "Taking one by one the bins so far in the solution, we check whether it is possible to **replace** up to three items in the bin by one or two items from those currently missing in the bins, in such a way that the total size of the items in the bin increases without overflowing the bin. If so, we perform the replacement, which means that some of the previously unassigned items are assigned to the bin, while some of the items previously assigned to the bin become 'unassigned'". This is an intelligent way of improving the offspring beyond the nature of the parent chromosomes. If replacements occur, the offspring chromosome will have at least a bin better filled, which will also leave "more space in the other bins (indirectly), which makes them more capable of accommodating additional items, and that leads ultimately to the desired reduction of the number of necessary bins". Evidently, the missing items will be smaller "than before the exchange, which makes easier the task of adding those items to bins already in the solution". This replacement process is repeated until there are no more possible exchanges. If missing items still exist at this stage, the remaining items will be inserted into the solution using the First Fit Decreasing strategy. The crossover with bound optimization is thus explained.

Finally, the author proceeds on explaining the mutation operator. "Given a chromosome, we select at random a few bins (i.e. *genes*) and eliminate them. The items which composed those bins are thus missing from the solution and must be inserted back. To do so, we again use the ideas applied for the crossover. Namely, the eliminated items first serve as the basis for eventual improvement of the bins left unchanged, during a stage of replacement. When no replacement is possible anymore, the FFD heuristic is used to complete the solution". The mutation operator is, indeed, simple, following the same pattern of the crossover final optimization. Falkenauer's experimental results are quite satisfactory, surpassing the MTP solution.

This concludes the genetic algorithms exposition. It seems that the Grouping Genetic algorithm is the most adequate of all genetic algorithms to use in such a problem as the Bin Packing problem. Indeed, Falkenauer presented quite compelling arguments supporting it. Rather important is the local optimization achieved by replacement of items in bins, which propagates to an actual global optimization, as the author also explains. For the author of this present thesis, it is an almost necessary feature to have at least some replacement method in the solving algorithm to be presented. Accordingly, Falkenauer's local optimization, inspired by Martello and Toth's Dominance Criterion, or some variation of it, may be used in the final algorithm. Of course, this feature has its limitations, and one needs to juggle between more possible replacements and less time complexity. The genetic algorithm approach also seems to bridge the reduced possibilities of lower bounded replacements.

Switching to an exact type algorithm, the theoretical exposition is concluded with the presentation of the Bin Completion strategy, by Richard E. Korf, and later enhanced with the help of Ethan L. Schreiber.

3.1.3 The Bin Completion Algorithm

Richard E. Korf presented its algorithm for Optimal Bin Packing first in 2002 [34]. However, in 2003 he presented a paper with the title "A New Algorithm for Optimal Bin Packing", which was an improvement on the original 2002 algorithm. For this reason, it is the 2003 algorithm that is referred as "Korf's original algorithm", as it is mentioned in a footnote on his more recent paper of 2013 [35]: "Korf wrote two papers on bin completion, when we say original, we are referring to his 2003 paper which had some implementation improvements over his 2002 paper". In that later paper of 2013, Korf presents a yet improved algorithm, with Ethan L. Schreiber. The algorithm of 2002 is now to be presented.

3.1.3.1 Bin Completion - A New Algorithm for Optimal Bin Packing, by Richard E. Korf [34]

Similarly to Falkenauer in its previously mentioned Hybrid Grouping Genetic Algorithm, Korf drinks from Martello and Toth's wiseness. Indeed, he presents the MTP algorithm as the "best existing algorithm for finding optimal solutions to bin-packing problems". Just as Falkenauer, Korf gives credit to the Dominance Criterion for MTP's efficiency: "[t]he main source of efficiency of the Martello and Toth algorithm is a method to reduce the size of the remaining subproblems". Interestingly, though this paper dates from 2003, and Falkenauer's paper dates from 1996, it seems rather unlikely that Korf was unfamiliar with the proposed Hybrid Grouping Genetic Algorithm. It is rather curious that, even though Falkenauer presents experimental results proving the superiority of its algorithm over the MTP algorithm, Korf still mantains that the MTP is, as stated before, the "best existing algorithm for finding optimal solutions to bin-packing problems".

In the view of this thesis' author, Korf makes important remarks on the nature itself of a Bin Packing Problem where the information of all existing items is present right from the beginning of the solution search. It is worth noting, of course, that these remarks are mainly based on the primary observations of Martello and Toth. Indeed, the observation of the Dominance Criterion and Dominance Relations between items is a major breakthrough in the Bin Packing Problem scenario, and Korf proposes a similar algorithm to that of Martello and Toth, but while making "more effective use of these dominance relations". First, the mentioned dominance relations must be presented. It is just the simple fact of the Dominance Criterion of Martello and Toth, but this same observation enables a number of conclusions regarding the Bin Packing Problem, which in turn simplify drastically the search heuristic. The mentioned criterion was already presented by Falkenauer, with a simple illustration. Korf gives this definition: "[d]efine a feasible set as any set of elements whose sum doesn't exceed the bin capacity. Let A and B be two feasible sets. If all the elements of B can be packed into a set of bins whose capacities are the elements of A, then set A *dominates* set B. Given all the feasible sets that contain a common element x, only the undominated sets need be considered for assignment to the bin containing x".

An example can be given for better understanding how a certain set *dominates* another. The former shipper protagonist of the Bin Packing Problem example mentioned before has the following items: 14, 11, 7, 4, 2. The shipper wants to fill a given bin which still has an empty space of size 17. This means that the possible sets to introduce in that bin are 14, 11, 7, 4, 2, {14+2}, {11+2}, {11+4}, {11+4+2}, {7+2}, {7+4}, {7+4+2}. A variety of possible insertions. When testing if a given set A is dominated by another set B, one must suppose that the elements of B are bins with different capacities. If the elements of A can all be distributed by those bins, than A is said to be dominated by B. The item 2 option is dominated by all other sets. Item 4 is also dominated. Indeed, if one considers the example of the set {11+2}, one sees that item 4 can be inserted in the supposed bin of capacity 11. The item 14, on the contrary, does not fit in any of the supposed bins of this set. Still, set {14+2} dominates the set {14}. However, {14+2} does not manage to dominate set {11+4}. Indeed, one cannot take the elements 11 and 4 and distribute them by a bin of size 14 and a bin of size 2. Still, set {11+4} does not dominate set {14+2} either. Applying this logic, one concludes that sets {14+2} and {11+4+2} are not dominated by any other set, although neither of them dominates the other. If an algorithm is searching for the optimal solution, it must consider these two possibilities, and according to the Dominance Criterion it does not need to consider the other possible sets. The shipper would pick one of these final sets to insert in the bin. The Bin Completion algorithm is a branch-and-bound algorithm. Upon finding these two

dominating sets, it would produce a "two-way branch in the search".

To demonstrate how this criterion can originate Dominance Relations capable of simplifying the problem, the author presents three relation examples.

1. Let's suppose one has an optimal solution for a certain Bin Packing problem. "Consider two elements x and y, such that $x + y = c$, where c is the bin capacity". Let us also assume that these elements are in different bins contained in the mentioned optimal solution. This means that the other elements filling the bin containing x will sum to at most the size of y, which also means that the element y can swap with those elements (indeed, y will fit in the bin with x and the other elements that were in the bin with x will fit the other bin where y was), and the number of bins in the solution will not be increased. The author, therefore, concludes: "given a problem with two values x and y such that $x + y = c$, we can always put x and y in the same bin, resulting in a smaller problem. (...) [T]his does not extend to three or more elements that sum to exactly the bin capacity".
2. Consider another element x, "such that the smallest two remaining elements added to x will exceed c". This means that, if the two smallest items do not fit together in a bin with x, then the bin containing x will only be able to have one more item. Thus, if we pick the largest remaining element y "such that $x + y \leq c$", the bin containing x will be more filled than what could ever be, and the optimal solution will not be compromised.
3. "As a final example, assume that y is the largest remaining element that can be added to x such that $x + y \leq c$, and that y equals or exceeds the sum of *any* set of remaining elements that can be added to x without exceeding c." Because "any other set of elements that were placed in the same bin as x could be swapped with y without increasing the number of bins", then we can put x and y in a bin for them without again "sacrificing solution quality".

The Bin Completion algorithm is, as said previously, a branch-and-bound algorithm, just like the MTP. However, contrary to Martello and Toth's algorithm, it "searches a different problem space", considering "each bin in turn", instead of "considering each element in turn, and deciding which bin to place it in", and considering also "the undominated feasible sets of elements that could be used to complete that bin". The algorithm begins by sorting "the elements in decreasing order of size", and then it considers "the bins containing each element in turn, enumerating all the undominated completions of that bin, and branching if there are more than one. In other words, we first complete the bin containing the largest element, then complete the bin containing the second largest element, etc". Although Korf says this to be a bin oriented scheme, and it definitely is, it may also be described as a "largest element" oriented scheme, where the extremities of the ordered set of items are considered, and where the smaller items are considered as possible companions to the largest element.

The Bin Completion algorithm also makes use of a lower bound function, computing a "lower bound on the minimum number of bins needed". That function is based on the L2 Lower Bound method of Martello and Toth [33]. Korf's algorithm for computing the lower bound behaves as follows: with the help of the Dominance Relations and some optimistic suppositions, the algorithm aims on computing the estimated wasted-space of the optimal solution. "We consider the elements in decreasing order of size. Given an element x, the residual capacity r of the bin containing x is $r = c - x$, where c is the bin capacity. We then consider the sum s of all elements less than or equal to r, which have not already been assigned to a previous bin. There are three possible cases. The first is that r equals s. In that case, there is no estimated waste, and no carry over to the next bin. If $s < r$, then $r - s$ is added to the estimated waste, and again there is no carryover to the next bin. Finally, if $r < s$, then there is no waste added, and $s - r$ is carried over to the next bin. Once the estimated waste is computed, it is added to the sum of the elements, which is divided by the bin capacity, and then rounded up". This is, evidently, an enhancement to the simple lower bound calculation method of summing all existing elements and dividing by the bin capacity. That simple method assumes an optimal solution of all elements being distributed as to successfully fill all used bins. This is an impossibility in practically the entire spectrum of Bin Packing scenarios. Korf's method, though sacrificing time, evidently, computes more efficiently the lower bound. The author will use this calculation for his Bin Completion algorithm.

The general algorithm presented in this paper is as follows:

1. The Best Fit Decreasing solution is computed.

2. The lower bound of the entire problem is computed using the wasted-space estimation explained above.

3. "If the lower bound equals the number of bins in the BFD [Best Fit Decreasing] solution, it is returned as the optimal solution". Of course, in the majority of complex scenarios, this will not be the case. "Otherwise we initialize the best solution so far to the BFD solution, and start a branch-and-bound search for strictly better solutions". Let us assume the algorithm continues along this road, instead of stopping in the Best Fit Decreasing solution as the optimal one.

4. As previously described, the elements are considered "in decreasing order of size", all the feasible undominated sets completing "the bin containing the current element" are generated. "If there are no completions or only one undominated completion, we complete the bin in that way and go on to the bin containing the next largest element. If there is more than one undominated completion, we order them in decreasing order of total sum, and consider the largest first, leaving the others as potential future branch points. Whenever we find a complete solution that is better than the best so far, we update the best solution found so far".

5. During the search, if the partial solution of one of the branches "uses as many bins as the best complete solution found so far", that branch is pruned. "For a lower bound on a partial solution, we use the sum of all the elements, plus the actual space remaining in the bins completed so far, divided by the bin capacity and rounded up to the next larger integer". As the author explains, "[t]his lower bound is more effective than the estimated wasted-space bound described above, because it includes the actual wasted space in the completed bins". This is evident, given that a part of the estimated wasted space is really present in the partial solution. The bound computing is also time efficient. Actually, it is a constant-time function, as it computes "by just accumulating the amounts of wasted space in the completed bins". The author also guarantees that "it doesn't help to additionally compute the estimated wasted space in the remaining bins". If this computed lower bound for the partial solution is equal to or greater than the "number of bins in the best solution so far", that branch is pruned.

6. Since what consumes the most amount of time in this algorithm is the computing of the feasible undominated sets for completion of a given bin, the author proposes the generation of "a subset of the feasible completions", and these are tested for dominance. "Given a particular element x, we first find the largest element y for which $x+y \leq c$. We then compute all feasible pairs w and z, such that $x + w + z \leq c$, which are undominated by the single element y, i.e. $w + x > y$, and such that no pair dominates any other pair". This is where the problem gets more complicated. Indeed, the author already mentioned that it is in computing the undominated feasible sets where the algorithm looses the majority of its time. The way of computing the mentioned feasible undominated pairs, therefore, must be as efficient as possible, thus minimizing the time consumption and the overall time complexity of the algorithm. Korf describes its way of computing these pairs in the following manner: "[g]iven two pairs of elements, $d + e$ and $f + g$, all four elements must be distinct, or the pair with the larger sum will dominate the other. Assume without loss of generality that $d > f$. In that case, g must be greater than e, or $d + e$ will dominate $f + g$. Thus, given any two pairs of elements, neither of which dominates the other, one must be completely 'nested' inside the other in sorted order". This is a crucial observation. Given a list of items 9,7,6,3,1, we see, by applying the aforementioned logic, that to generate two pairs of four distinct elements where neither pair dominates the other, one of the pairs must be nested in the other in that ordered list. Indeed, pairs {9,3} and {7,6} do not dominate each other, neither pairs {9,1} and {6,3}. As of pairs such as {7,3} and {6,1}, clearly one dominates the other, given that 6 is smaller than 7 and 1 is smaller than 3. The author's conclusion is evident, and rather important. "We can generate all such undominated pairs in linear time by keeping the remaining elements sorted, and maintaining two pointers into the list, a pointer to a larger element and a pointer to a smaller element. If the sum of the two elements pointed to exceeds the bin capacity, we bump the larger pointer down to the next smaller element. If the sum of the two elements pointed to is less than or equal to y [note that y is the size

of the previously mentioned largest element for which $x + y \leq c$], we bump the smaller point up to the next smaller element. If neither of these cases occur, we have an undominated pair, and we bump the larger pointer down and the smaller pointer up. We stop when the pointers reach each other". All undominated pairs are, thus, computed.

7. The algorithm still proceeds on computing triples, i.e. sets of three elements "d, e and f, such that $x + d + e + f \leq c$ and $d + e + f > y$, then all quadruples, etc. To compute all triples, we choose each feasible first element, and then use our undominated pairs algorithm for the remaining two elements. For all quadruples, we choose all possible feasible pairs, then use our undominated pairs algorithm for the remaining two elements, etc. These larger feasible sets will in general include dominated sets". It is rather clear that the Achilles' heel of Korf's algorithm lies in computing these feasible sets. Indeed, time complexity of calculating feasible sets starts increasing rapidly. For a large number of small items and a large bin capacity, the algorithm seems to loose itself in computing such feasible sets, with no limitations whatsoever. However, there is an added bonus to computing the undominated sets rather than testing for the one dominating all the others. "Given two feasible sets, determining if the one with the larger sum dominates the other is another bin-packing problem. This problem is typically so small that we solve it directly with brute-force search. We believe that we can significantly improve our implementation, by directly generating all and only undominated completions, eliminating the need to test for dominance".

It is worth mentioning that Korf comes to a different conclusion than that of Falkenauer regarding the choice of using the Best Fit Decreasing strategy over First Fit Decreasing approach. Contrary to Falkenauer, who states that Best Fit Decreasing does not prove to be better, rather it just contributes to spending more precious time, Korf mantains that "it was not worthwhile to also compute the first-fit decreasing (FFD) solution, since the FFD solution very rarely uses fewer bins than the BFD solution".

As the experimental results demonstrate, the "bin-completion algorithm dramatically outperforms the Martello and Toth algorithm on the larger problems". Indeed, the algorithm "appears to be asymptotically faster than the best existing algorithm [MTP], and runs more than a thousand times faster on problems of 60 elements". One must notice that the solution implemented by Korf seems to benefit from large value distributions along the elements comprising the Bin Packing problem, but the problem addressed by this thesis will most probably have small value distributions (there is also the fact that none of these algorithms present the possibility of having elements with the same value, although that will necessarily have to be contemplated in the solving algorithm presented in this thesis).

This depth-first search is an interesting and important example of the branch-and-bound algorithm. It is a rather distinct approach to that of the genetic algorithms, in which an evolution of a population of solutions occurs, using the "survival of the fittest" method. Here, a search is made with logical constraints which guide the heuristic in order to achieve a better solution than the previous one, until either reaching the lower bound or until exhausting all possibilities (which comprise a very reduced search space than that of usual Bin Packing Problem searches).

3.2 Linear Programming

The linear programming approach is studied due to its characteristic of finding the optimal solution of the problem, if it exists. A linear programming problem can be described with the following general mathematical formulation:

$$
\begin{array}{lccccccccc}
\text{Minimize} & c_1x_1 & + & c_2x_2 & + & \cdots & + & c_nx_n & = & z \\
\text{Subject to} & a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\
& a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\
& \vdots & & \vdots & & & & \vdots & \vdots & \vdots \\
& a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m \\
& x_1, & & x_2, & & \ldots, & & x_n & \geq & 0.
\end{array}
$$

Figure 7: General formulation of the linear programming problem, by Catherine Lewis [36]

Considering the above formula, Catherine Lewis comments [36]:

> In linear programming z, the expression being optimized, is called the *objective function*. The variables x_1, x_2 ... x_n are called *decision variables*, and their values are subject to $m + 1$ constraints (every line ending with a b_i, plus the nonnegativity constraint). A set of x_1, x_2 ... x_n satisfying all the constraints is called a *feasible point* and the set of all such points is called the *feasible region*. The solution of the linear program must be a point $(x_1, x_2, ..., x_n)$ in the feasible region, or else not all the constraints would be satisfied.

The linear constraints build a convex feasible region. A two dimensional example is shown below.

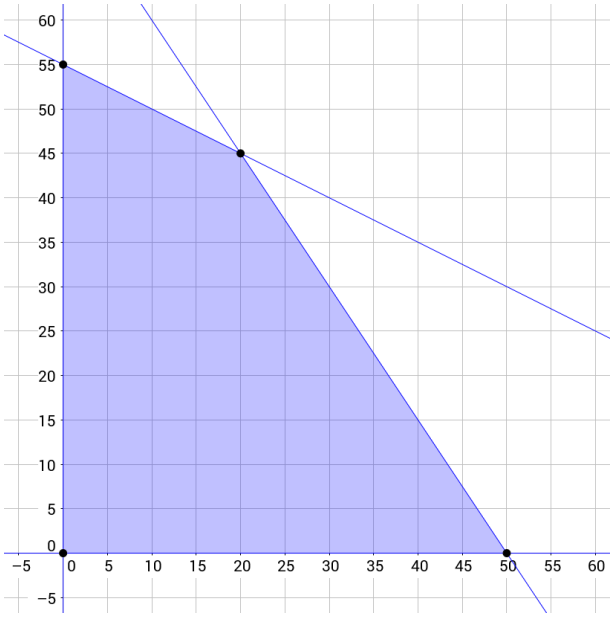

Figure 8: 2D linear programming example displayed in a graphic [37].

The purple region corresponds to the feasible region. It can be shown that, if an optimal solution exists, at least one of the extreme points (in 2D, the extreme points correspond to the plane corners of the feasible region) is an optimal solution. This is described and demonstrated by Catherine Lewis, which concludes with the enunciation of the following theorem [36]:

Theorem 6.2. *If an optimal solution exists, then an optimal extreme point exists.*

Linear programming problems have known efficient ways of finding an optimal solution, if it exists, such as the Simplex method, which is also explained by Catherine Lewis. The limitation of the linear programming approach is the complexity of the problem when there are lots of variables in play. In two dimensions, one can picture the feasible region and its extreme points. In three dimensions, it is still possible to imagine a 3D solid with a finite number of corners. Once we have tens or hundreds of variables, one can hardly grasp the dimension of the problem.

Nevertheless, one can imagine that, even though a problem can have a big number of dimensions, i.e. a great number of variables and constraints to be satisfied, it seems that the problem is always solvable. Even more interesting than that, if the problem does have an optimal solution, the linear programming approach will find it. Considering the problem this thesis addresses, the time complexity of the linear programming approach is enormous and extremely unpractical. However, if it is possible to simplify the problem in hand to a linear programming problem and eventually find an optimal solution, we can use it to measure the optimality of the solutions that are found by the other algorithms that will be used and that do not guarantee an optimal solution. The linear programming problem solution will, thus, be the benchmark for some of the testing that is to be done in the coming algorithms.

4 Solution Implementation

The algorithms developed for this thesis are here presented. As previously mentioned, the Linear Programming approach is the one which sets the goal mark for the remaining algorithms, which are not as exact, but undoubtedly have less time complexity. As one can notice, the algorithms are impregnated with the bin packing problem approaches and concepts, which abstract this thesis' complex problem into smaller and simpler ones. All the algorithms were written in the Python programming language.

4.1 Linear Programming Approach

All linear programming models hereby presented leverage the PuLP package. As stated in its documentation [38]:

> PuLP is a free open source software written in Python. It is used to describe optimisation problems as mathematical models. PuLP can then call any of numerous external LP [Linear Programming] solvers (CBC, GLPK, CPLEX, Gurobi etc) to solve this model and then use python commands to manipulate and display the solution.

The selected solver was Gurobi 9.0 [39], using an academic license. This solver exceeded by far the performance of the default PuLP's solver.

4.1.1 1D Bin Packing

First of all, a linear programming algorithm for the simple one-dimensional bin packing problem was developed. The bin packing problem was transformed into a linear programming one, with constraints on variables and a cost function that, in this case, was to be minimized. For simplicity, the declared variables assume a boolean type, i.e either 0 or 1. Thus, the problem is actually a more complex version of the linear programming model: it is a **0-1 integer programming problem**.

4.1.1.1 General algorithm

The algorithm starts by assuming the worse possible scenario that N bins will be used for N existing items, which correspond to an item per bin. However, this does not entail that all these bins will be used in the end. The problem variables are the following:

- bin usage: there is a variable associated to each existing bin, which will be 1 if that bin is used in the solution and 0 if not.
- presence of item x in bin y: each of these variables associate with a bin and an item, and it will tell if the item y is in the bin x (value 1) or not (value 0).

This means that, for a problem with N different items, there will be $(N+1)N$ variables in the problem.

The problem has a constraint for each item and a constraint for each bin.

- item constraint: the sum of all *presence of item* variables associated to a given item must equal 1, i.e each item is present in only one of the existing bins.
- bin constraint: the total size of all present items in a given bin must be less or equal to its bin capacity multiplied by its *bin usage* variable, i.e the items present in a bin must not exceed its maximum capacity. These constraints also mean that no items are present in a given bin if that one is not used, which will force *presence of item* variables to be 1 only on bins that are used.

Again, for simplicity, looking on the fact that the thesis' problem aims at reducing the number of used time slots, i.e bins, the 0-1 integer programming problem will be solved by minimizing the sum of all *bin usage* variables.

With a few simple random examples, it becomes evident how critical the time complexity gets with the amount of items to be analysed.

4.1.1.2 MTP algorithm, by SCIP

The already mentioned Martello and Toth's MTP algorithm for Bin Packing problems was also used for testing. The procedure is implemented in Python in the coding examples of *Python*'s *PySCIPOpt* library. As its documentation explains, "*This project provides an interface from Python to the SCIP Optimization Suite*" [40], where *SCIP*[41] stands for *Solving Constraint Integer Programs*. As it is specified in *SCIP*'s documentation [42]:

> "SCIP is currently one of the fastest non-commercial solvers for mixed integer programming (MIP) and mixed integer nonlinear programming (MINLP). It is also a framework for constraint integer programming and branch-cut-and-price. It allows for total control of the solution process and the access of detailed information down to the guts of the solver. (...)
>
> SCIP is a framework for Constraint Integer Programming oriented towards the needs of mathematical programming experts who want to have total control of the solution process and access detailed information down to the guts of the solver. SCIP can also be used as a pure MIP and MINLP solver or as a framework for branch-cut-and-price.
>
> SCIP is implemented as C callable library and provides C++ wrapper classes for user plugins."

4.1.1.3 Results

For performance registration of the bin packing integer programming solution, the same problem setups were fed to the more commonly known simple bin packing algorithms: Next Fit Decreasing, First Fit Decreasing, Worth Fit Decreasing and Best Fit Decreasing. It is generally assumed that for the majority of cases the "decreasing" variation of the algorithms presents greater performance. These algorithms surpass by far the integer programming solution as far as time complexity goes, as the results will show. The aforementioned algorithms were implemented in *Python 3.6.5*, following its general and simple algorithmic structure. The problem setups were in general randomly generated in *Python*, with some specific case studies manually generated as well. The MTP algorithm and the Linear Programming algorithm were fed with the same setups.

The tests were performed on a machine with an Intel Core i7 processor of 8th generation (with 12 cores), 12GB of RAM and an Ubuntu 18.04 operating system. A few things need to be mentioned regarding limitations on the algorithms:

- Gurobi solver is capable of heavy multiprocessing, which allows for a nice load distribution throughout all 12 cores. In more difficult cases, the load average could go to 12, corresponding to 100% of load on all cores.
- The implementation of the MTP algorithm seems to be limited to one of the cores, which may give a great performance disadvantage to it in comparison with Gurobi solver.
- When either Gurobi or the MTP implementation reaches the first estimation of the gap between the current solution and the optimal one, the program can be manually stopped and that solution is given by the algorithm. The manual stop was used a number of times, or either one of the algorithms would go on for hours. In the worst cases that the machine can support, they could go for days.
- Even after a few hours, the MTP algorithm is sometimes incapable of achieving a non infinite gap in relation to the optimal solution. Due to that, the tests become limited to the MTP's capacity.

The following bar chart generally describes the various testing setups:

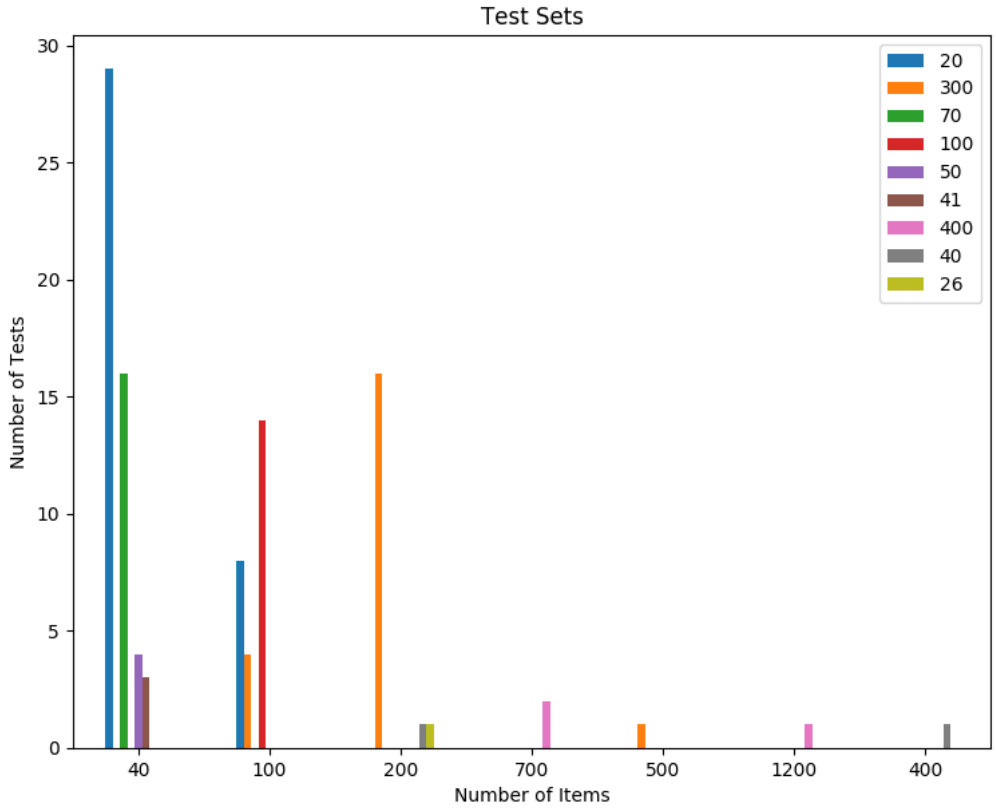

Figure 9: Bar chart of Bin Packing testing sets.

In total, 101 tests were made to the aforementioned bin packing algorithms. Each test was generated by randomly picking a number from 1 to the maximum bin capacity as the size of each given item. The previous graph shows the number of tests that were accomplished for a given number of items to be packed and a given value of bin capacity. Each bar color represents a different bin capacity value for a test.

Each method can be given a performance value in percentage, derived from the bins used by comparison to the method which got the minimum bins used in that test. Looking into the performance of each of the simpler methods, described as its sole capacity of achieving the least amount of bins possible, i.e packing optimality, we get the following results:

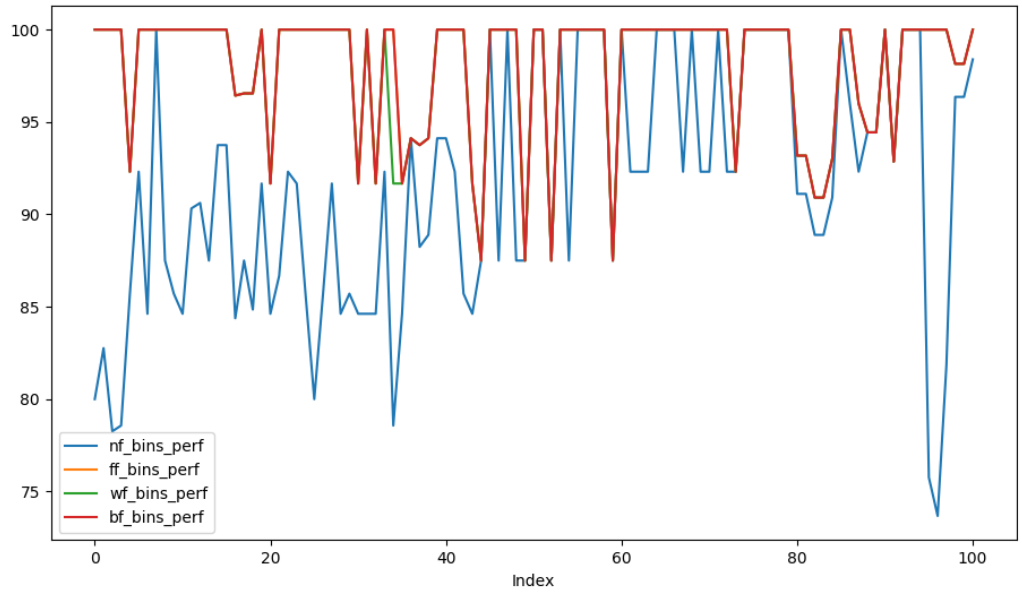

Figure 10: Performance (%), or packing optimality, per each test index, of the simpler bin packing methods.

We immediately notice, as expected, that the packing capacity optimality of the Next Fit Decreasing method is poor compared to the other ones. Thus, this method is to be ignored in the optimization analysis. We get the following graph:

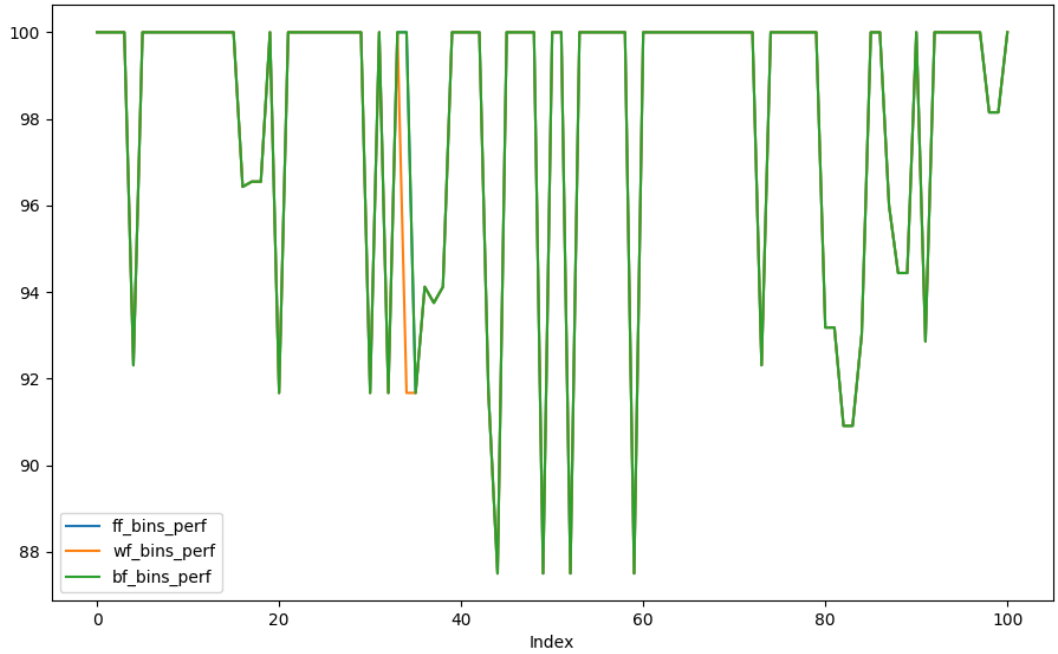

Figure 11: Performance (%), or packing optimality, per each test index, of the simpler bin packing methods, ignoring Next Fit Decreasing.

If one were to say that First Fit Decreasing, Worst Fit Decreasing and Best Fit Decreasing have the packing optimization capacity, one would not be as distant from the truth as it would be initially thought, as these results demonstrate. Indeed, in only 2 out of the 101 tests did the Worst Fit Decreasing method fail to reach 100% of packing optimality, staying a little under the mark of the 92%, and under the 100%

mark of the 2 other algorithms. As for the First Fit Decreasing and Best Fit Decreasing methods, as curious as it may be, they had precisely the same packing optimality in the entire set of tests. This, evidently, does not mean that one method may win over the other in very specific setups. In terms of absolute optimality, these methods are able to reach the best possible packing of bins in the majority of the test cases, and their performance do not fall under the mark of the 87%.

Focusing our attention, now, on the MTP and Linear Programming algorithms, we see the following results using the previous analysis:

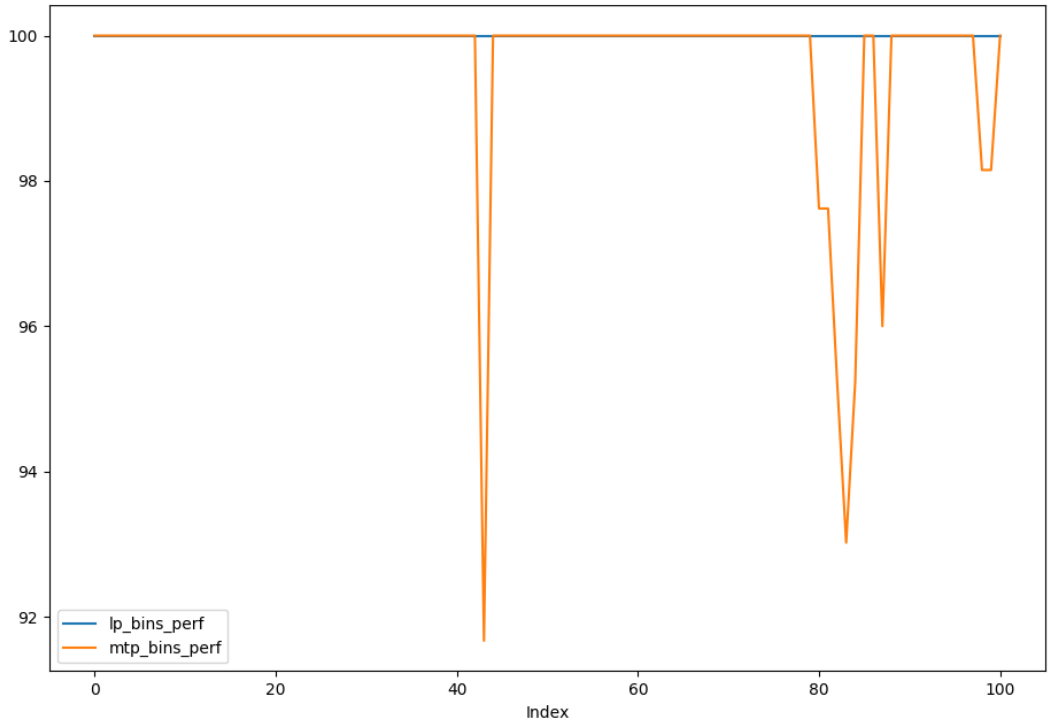

Figure 12: Performance (%) of Linear Programming and Martello and Toth packing algorithms.

The Linear Programming method, as expected, is always capable of achieving the best possible packing of bins. The MTP method, though it never dropped under the mark of the 91%, failed to reach optimality in 7 out of 101 tests.

In the end of this analysis, we can affirm that Linear Programming had a global mean bin packing optimality of 100%, MTP reached 99.63%, Best Fit Decreasing and First Fit Decreasing reached the same mean optimality of 98.02%, Worst Fit Decreasing reached 97.93% and Next Fit Decreasing reached 91.47%.

We proceed to the time performance analysis. Indeed, as it was previously mentioned, the MTP and Linear Programming algorithms were, in many cases, manually interrupted. The presented duration corresponds to the time a given test took to achieve the presented packing of bins, although, if it were not for the manual stop, that same time could have been much bigger and packing optimality greater.

Again, as expected, the methods' performance in time seems inversely proportional to packing performance. Indeed, Next Fit Decreasing spent the least amount of time in all tests. As for the other typical bin packing methods, i.e First Fit Decreasing, Worst Fit Decreasing and Best Fit Decreasing, none of them went over the 0.02 seconds mark, even though these algorithms were written in Python, which tends to be outperformed by more low level solvers (written in C or C++) given that it is an interpreted language [43].

As per the Linear Programming and MTP algorithms, we can see that they are the ones which reach

the greatest duration times. The Linear Programming algorithm manages to find the optimal solution faster than the MTP method in 53 out of the 101 tests, corresponding to about 52.48% of the tests. However, some worthy points to mention that are not shown from the time registrations are that, firstly, manual stop had to be done much more times to the MTP algorithm, and secondly, in these long duration cases, the Linear Programming method was able to find at least one possible solution in a much faster time interval than the MTP, which had to have much more time to achieve some possible solution, and only then be manually stopped.

All in all, these are the results of the bin packing tests:

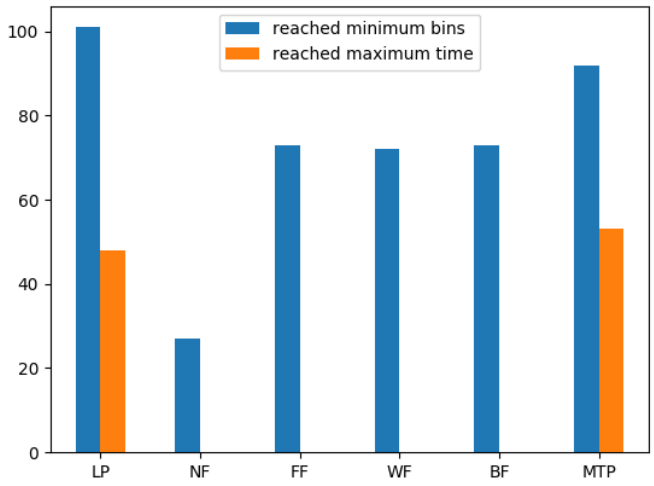

Figure 13: Overall performance, in absolute counting (number of tests), of the Bin Packing methods in the testing set.

The Linear Programming method wins over the MTP method and all the simpler ones, when looking at the packing optimization. It also succeeds on overcoming the MTP algorithm regarding time usage. The Next Fit Decreasing achieves the minimum bins in 27 out of 101 tests, i.e 26.73% of the tests, which confirms its poor packing capacity, entailed by the simplicity of its method. The First Fit Decreasing and Best Fit Decreasing methods reach the minimum bin number in 73 out of 101 tests, i.e 72.28%. In 70.30% of the tests, the Best Fit Decreasing method managed to consume less time than the First Fit Decreasing approach. The MTP algorithm achieves the optimal bin mark in 92 tests corresponding to 91.09% of tests. The Linear Programming method seems to only have advantages over the MTP algorithm, both in packing performance and time spending. However, First Fit Decreasing and Best Fit Decreasing, though loosing in packing optimization, significantly win regarding time consumption.

4.1.2 Full problem with Water and Fertilization - First approach

To create a linear programming model for the problem in hand, an objective function must be created, and variables must be as such that constraints are linear. The objective function is composed by three different cost functions:

- F_e: *energy function*. It accounts for how much the irrigation plan spends on energy due based on the energetic week tariffs. In the following tests, two different energy tariffs were used, although the principle is the same. The objective is to use expensive time slots only if cheap time slots are already occupied and it is absolutely needed to irrigate more. For simplicity, this is translated to the **sum of all used expensive time slots of the week, divided by the quantity of existing expensive hours**. This way, function values will range from 0 to 1, where the objective is to minimize the returning value.

$$F_e = \frac{used.expensive.slots}{existing.expensive.slots} \tag{2}$$

- F_{wb}: *water balance function.* Between two plans with the same energy efficiency, the preferable plan would be the one where each culture is as close to its ideal water level as possible. This function accounts for it by the following not so simple yet linear formula:

$$F_{wb} = \frac{\sum \Delta ideal(for.all.time.slots.and.all.sectors)}{\sum ideal(of.all.sectors.for.all.time.slots)} \tag{3}$$

  $\Delta ideal$ is a quantity which describes how near a given sector is from its ideal water level on a given time slot. Its value is the water level of the culture if that value is below or equal to ideal, and it is the water level minus its ideal water level if the first value is over the ideal. Although a little bit tricky, this guarantees that F_{wb} stays between 0 and 1. If the culture were to be in the ideal water level at all time, this function would return 1.

- F_f: *fertilization function.* It accounts for how well the plan intends to fulfill fertilization needs in the coming week. For simplicity, this is translated to the sum of fertilizer debit in all time slots and for all sectors, divided by the sum of the weekly needs of each sector. This leads to a value ranging between 0 and infinity.

$$F_f = \frac{\sum fert.debit(for.all.time.slots.and.all.sectors)}{\sum fert.needs(of.all.sectors)} \tag{4}$$

The most important cost function is F_e. The others are more detail-oriented. It is worth mentioning that, though F_f reaches larger values compared to the other cost functions (given that it can be greater than 1), its output will be restrained by the constraints that will be set over the problem. Furthermore, F_e is the only function with the objective of being minimized. Without generating more complexity, $1 - F_e$ will enter into the final objective function.

The objective function is, therefore, a combination of the three previously mentioned cost functions. Given that not all cost functions have the same importance, a weight is attributed to each one of them. The final function to be maximized is the following:

$$F_O = \frac{k_e(1 - F_e) + k_f F_f + k_{wb} F_{wb}}{k_e + k_f + k_{wb}} \tag{5}$$

Once the objective function F_O is established and presented, we proceed to the **variables definition**.

- sector irrigating: for each time slot and for each sector, there is a variable which will tell whether the sector is irrigating or not.
- time slot fertilizing: for each time slot, a variable will tell whether the fertilizer pump is on or off.

For a problem setup with N sectors and T time slots, the linear programming model will use $(N+1)T$ variables. Once again, variables will assume boolean values, i.e 0 or 1, corresponding to a 0-1 integer programming problem.

4.1.3 Results

No actual results were registered for the linear programming approach to the entire problem. Indeed, two are the obstacles which were not surpassed:

1. The problem needs to be relaxed for it to be a linear programming problem. There are variables which multiply which multiply with other different variables, and such obstacles force simplifications of the entire problem.
2. The problem is too big to be resolved within a reasonable time-frame. At least for the linear programming model presented previously, tested solvers could not manage to provide a solution.

4.2 Combinatorial Approach with Bin-like Objects

This section explains the first approach to the problem using a Bin Packing abstraction. The algorithm has two main objectives:

- Prevent the culture from dying.
- Use energy in the cheapest possible hours in the energy tariff.

To provide a thorough explanation, one must first present and detail the objects which were created to abstract the problem and to be handled by the algorithm, after which a general description of the used strategy will be given.

4.2.1 General Abstraction

The abstraction is entirely based on the concept of a one-dimensional bin, with a given space for one-dimensional items. In this algorithm, they are called *buckets*.

4.2.1.1 Minute Bucket

The *Minute Bucket* is the smallest and most important object in the algorithm setup, as they are the bin-like objects which need to be maximized. Its size is the maximum capacity of the water pump. Each *Minute Bucket* corresponds to a given minute of the entire time-frame, which can vary in duration. It is used to register key information regarding this minute throughout the entire algorithm:

- Which sectors will be watered during that given minute.
- What is the energy price in that minute (i.e. if that minute corresponds to the cheap energy tariff or the expensive one).
- Which sectors, in any moment of the running algorithm, ever occupied the *bucket*, even if they are not there anymore.

4.2.1.2 Day Bucket

The *Day Bucket* object stores all the *Minute Bucket* objects of the represented day, assigning a price to each hour, and consequently to each minute. Although a day has 24 hours and, thus, 1440 minutes, *Day Bucket* can have a variable number of minutes.

4.2.1.3 Week Bucket

Using the same logic as that of the previously mentioned object, *Week Bucket* stores a variable number of *Day Bucket* objects. It assigns the price distribution for each of the days in the problem setup, and registers the allocated states.

4.2.1.4 Sector State

Finally, *Sector State* corresponds to the object which represents the irrigation sector and its crop's characteristics, and it is not an bin abstraction, like the previous ones. It stores minimum, ideal and maximum water levels of the crop, and also any happening irrigation. Furthermore, it has the information regarding evapotranspiration throughout the entire problem's time-frame, which, together with the irrigation moments, enables the possibility of tracking the instantaneous water balance level any time during the *Week Bucket*.

4.2.2 General Algorithm

The algorithm handles a sector at a time, without knowing about the ones that are yet to come. The first step when handling the sector is to find the minute when its culture's water level will drop below the minimum possible water balance level. When that minute, henceforward called the *death minute*, is reached, the algorithm will sort all available minutes (i.e. minutes which do not yet maximize pump capacity), from the beginning until the *death minute*, by ascending order of price and time, respectively. This means that, if the *death minute* is 10, and we have minutes [0,1,3,4,7,8,9] as with price 1 and [2,5,6] with price 0, they will be ordered as [2,5,6,0,1,3,4,7,8,9].

At this point, each *Minute Bucket* will be evaluated and tested to see if the sector can be inserted there. Is the given *Minute Bucket* full? Is the given sector already present in that *bucket*, i.e. already being irrigated in that minute? If the sector were to be inserted in that *Minute Bucket*, would it pass the maximum water balance level at any time from that minute until the *death minute*?

If at least one of those conditions is true, that *bucket* will be ignored and the algorithm will move on to test the next minute in the ordered list. If none of those conditions is true, the algorithm will try to add the sector to the *Minute Bucket*. Provided the *bucket* has space for it, the insertion will be successfully accomplished, and the algorithm will return to searching for the *death minute*, which may be different now, thus repeating the entire process. That cycle will be repeated until the *death minute* is no longer inside the problem setup's time-frame.

If the algorithm finds out that the *Minute Bucket* does not have enough space for the sector during the process of adding it, it will test the *bucket* and assess whether a switch between sectors is desirable or not. It is inside this process of finding out possible switches for the sector where combinatorial methods are used. If a switch is desirable, then the given sector will be inserted in the *bucket*, and the sectors which switch with it are removed from all the existing *Minute Buckets* of the *Week Bucket*. This means that, on one hand, the given *Minute Bucket* becomes more full than before, and, on the other hand, the removed sectors must be inserted again throughout the entire *Week Bucket*.

The cycle is repeated for the entire time-frame, until the survival of the sector's crop is assured across the *Week Bucket*. This is done for all the sectors of the problem, until all the irrigation is properly packed inside the various *Minute Buckets*.

4.2.3 The Combinatorial Switch

As previously stated, when a given sector is being tested for *bucket* insertion, the later may not have enough space for it. When such thing happens without the *Minute Bucket* being fully filled, the algorithm will assert if it should switch the given sector with some other ones which are inside the *bucket*, thus leaving the later more filled than before, which is desirable. This step is performed with combinatorial logic. With the sectors present in the *Minute Bucket*, the algorithm will generate combinations of $1, 2, 3, ..., N$ sectors, where N is the number of sectors present in the *bucket* that is being handled. Each combination is tested to see if its total size (i.e. its total debit) is smaller than the sector's. If so, that switch combination will be stored. The search will proceed in order to try to find the best possible switch combination, i.e. the one which better fulfills the *bucket*. The search stops either when a combination which fully maximizes the *Minute Bucket* is found or when all possible combinations were generated (thus guaranteeing the return of the best possible combo).

This particularity of the algorithm highly increases time complexity. Indeed, as it will become evident by the results, the Combinatorial Switch provides exponential complexity to the solver. Let us consider a problem of an irrigation network with 5 sectors. Assuming the first 4 sectors were already allocated throughout the problem's time-frame, there is one sector left for irrigation. Whenever the algorithm encounters an unfulfilled *Minute Bucket*, it will test it for a possible switch. Considering the case where the *bucket* has the other 4 sectors inside it, the switch search will generate, at most, 15 combinations of sectors: 4 combinations of 1 sector, 6 combinations of 2 sectors, 4 combinations of 3 sectors, and 1 combination of 4 sectors.

Remembering that the sum of the values in row n in the Pascal Triangle is 2^n, where the first row corresponds to $n = 0$, the total of possible combinations for switch in a *Minute Bucket* with N sectors is $2^N - 1$. Not only is this an indication of a possible tendency for increase in time of the switch search as the number of sectors increase, but also it entails that the running duration of the algorithm will especially grow when the problem that is being handled has a great number of sectors which can fit in the water pump at the same time, thus increasing the number of sectors for combination when a switch is being searched: for 4 sectors, it's 15 combinations; for 10 sectors, it's 1023; for 100 sectors, it's 1267650600228229401496703205375 combinations. Evidently, this algorithm is not at all suited for such problems.

4.2.4 Results

To test the given algorithm, 45 tests were made with the same testing setup, only differing in the number of randomly generated sectors, ranging from 4 to 16 sectors. The following graph provides a scattered plot of the running time per number of sectors for each of the tests:

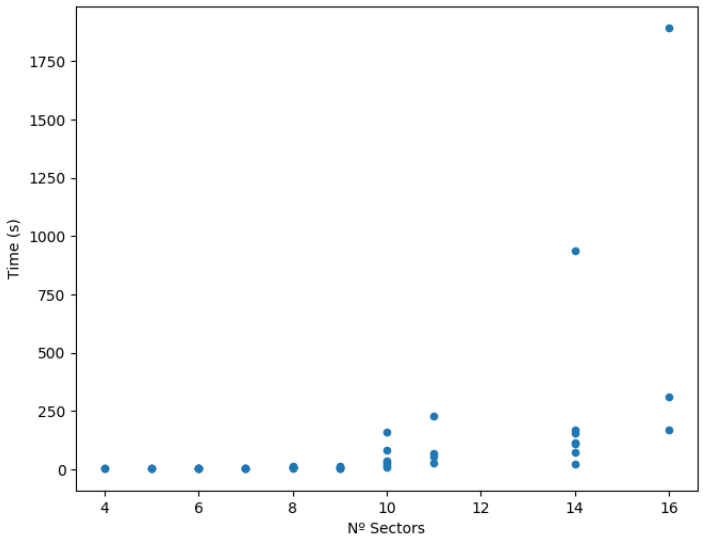

Figure 14: Running time results per number of sectors, across 45 tests.

There are 2 tests which present an abnormal running time (as seen in the previous graph, about 1000s for a 14 sectors test, and about 2000s for a 16 sectors test), due to some factor conjugation in the test. These are to be ignored for now, as they do not follow the normal dispersion. Without them, one can represent once again the previous results and see a roughly exponential increase in running time as the number of sectors increase:

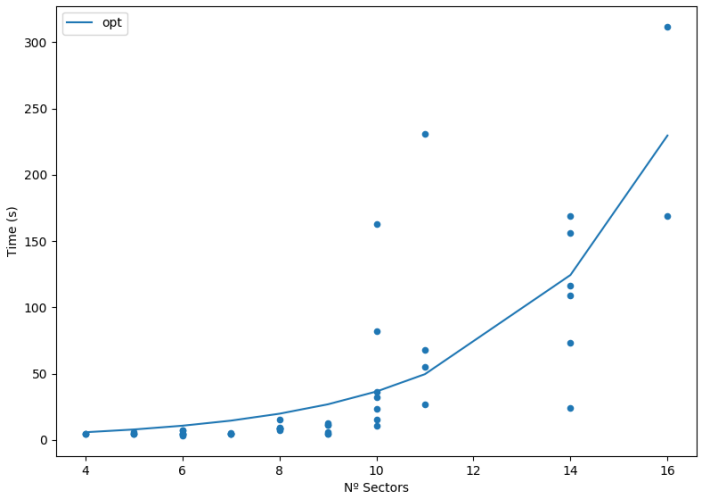

Figure 15: Curve fitting of test running time VS number of sectors.

The following graph shows the linear behaviour of the used memory in the algorithm that is being tested:

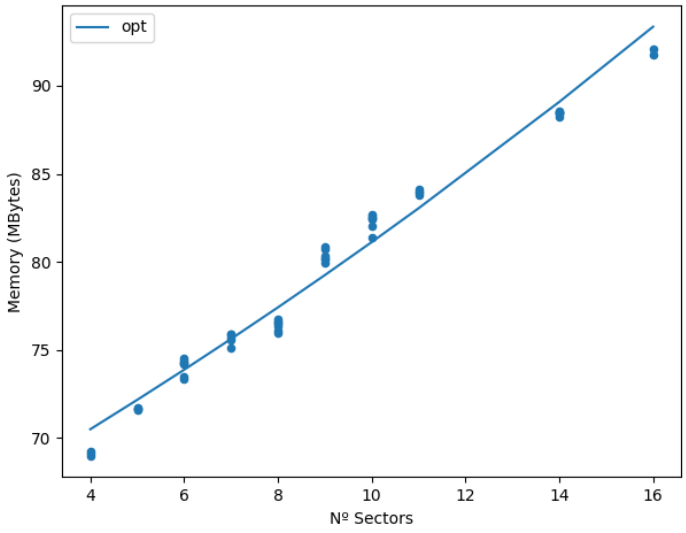

Figure 16: Curve fitting of test used memory VS number of sectors.

As it can be seen, the memory used by the algorithm presents a roughly linear behaviour in relation to the number of sectors in the testing setup.

4.2.5 Conclusions

Based on the previous testing of the aforementioned algorithm, a number of conclusion statements can be drawn.

1. **Memory usage is not problematic**: as seen in Figure 16, memory usage increases with the number of sectors in the problem setup, i.e. in the irrigation network. This is not only expected, as more memory is needed for each new sector, but it is also a controlled increase.

2. **Some problems may reach long running times**: indeed, as it can be seen in Figure 14, there is one test taking almost 1000 seconds (roughly 16 minutes) to plan a week irrigation plan for 14

sectors, and another test which reaches more than 1800 seconds (i.e. more than 30 minutes) of running time for a problem of 16 sectors.

3. **Running time increases exponentially**: using *SciPy*'s *curve_fit* method[44], it can be asserted that running time follows an exponential distribution with the increase of the number of sectors, described by the function $T = 1.7063976 * e^{0.30638935 * N}$, where T is the running time of the algorithm and N is the number of sectors. This entails, as evidenced by Figure 15, enormous amounts of time for problem solving with more sectors. Indeed, following the mentioned function, problems with 30 sectors are expected to take about 16748.5 seconds to complete, i.e. a little over 279 hours, which is more than 11 days of solving time. For 100 sectors, it is expected to take more than 66 million years to solve the problem. Noteworthy, an irrigation network of 100 sectors is not something implausible.

Evidently, the time complexity of the algorithm is a major problem. That is mostly due to two things: the great problematic of having to reallocate each sector through the entire week each time a switch takes place, and the fact that searching for a possible switch requires combinations, which presents exponential behaviour. Furthermore, the presented algorithm demonstrates a few imperfections which are present in any test made before. One is the fact that the algorithm's resolution is bound to 1 minute - it cannot perform a second decision, which would be desirable. The second aspect is the absence of water balancing around the ideal water level. Indeed, in this algorithm, there is no search whatsoever to keep crops as close to their ideal water level as possible, and as much time as it can be.

The following algorithm not only aims at bridging these imperfections, but it also leverages the increasing complexity of allocating fertilization for the entire irrigation network.

4.3 Greedy Decisions Approach

As stated, the former approach aims on providing an exact solution to the problem in hand, with a minor simplification due to the used constraints and cost functions. The exact optimal solution has an enormous time cost. The Greedy Decisions approach is an attempt of significantly improving time complexity with minor costs on the solution optimality.

The algorithm to be introduced is, indeed, a greedy search for a quasi-optimal solution, which is shown on the fact that the mentioned approach, once it makes a decision, it will stick to it. All greedy searches have the evident accuracy costs, but in terms of time complexity the algorithm is exceedingly faster than the exact solution approach. Whether the approach is worth it or not relies on how intelligent each decision is. Those difficult decisions come when the search branches into two or more possible ways of continuing the search. Indeed, the game is played on these branching nodes. To help the decision model take the right turn, or more accurately the best turn, the previously mentioned cost functions are used to evaluate the possible moves.

4.3.1 General Algorithm - Cheap-by-cheap Bin-packing with Greedy Decisions

In general, the algorithm can be viewed as a jumping succession of dealing with sub-problems, each of which being associated to a cheap-energy time interval. Indeed, the problem addressed by this thesis has two major aims:

- Prevent the culture from dying.
- Consume energy with the lowest possible tariffs.

The first mentioned objective is evident, and can also be seen as a constraint to the problem, rather than an objective. Trying to accomplish the second target will lead to the accomplishment of the other major and minor details important to the problem in hand:

- less water consumption,
- optimal water pump functioning,
- predictive weather parameters computing,

- intelligent water balance tracking

For this reason, it is important to take significant attention to the cheap-energy intervals, henceforward referred to as cheap intervals.

As it was being said, the algorithm grabs the first cheap interval it has. The first thing to be careful about is to **assure no cultures are dying already**. If that is the case, one has no option but to irrigate the endangered sectors before reaching the cheap interval in hand. Thus, the algorithm proceeds to allocate irrigation for these sectors with a bin packing approach. This first bin packing phase is called **Bin Packing for survival**. This and the other sections of the algorithm will later be explained with more attention. For now, it is being presented just the big picture of it all.

Once all sectors are in the safe zone, we can proceed with handling the cheap interval we grabbed previously. Firstly, all sectors with fertilization needs are bin-packed into the first time slots of the given cheap interval, in the **Fertilization Packing** phase. Some sectors will end up being enough irrigated as well, some will not. Thus, we proceed to the **Irrigation-only Packing**, where these remaining sectors are packed into time slots. Noticeably, there is a variety of situations that need being covered, such as reaching the maximum irrigation level after having reached fertilization needs, having the water pump functioning at a very low percentage of its optimal point due to small sectors being left out of the packing, running out of cheap time slots, etc. All that will be covered later in detail.

Either because all packing was successfully accomplished without step-backs or because the algorithm decided to, we will move on to the next cheap interval, and the phases cycle will repeat. The **Bin Packing for survival** phase remains in all subsequent loops as a safety measure, although each loop will try to guarantee that no culture will die until the next cheap interval.

4.3.2 Phase 1 - Bin Packing for survival

As stated, the elements that are already dying by the beginning of a given cheap interval are destined to go through a bin packing process in which the objective is for them to reach the minimum level of survival of each culture. This has a big cost on energy consumption, given that the time slots where the sectors will be packed are more expensive. However, without question this irrigation must be made, as the survival of all cultures is a constraint of the problem in hand. Evidently, even the optimal solution obtained by the linear programming approach would have to irrigate in these time slots.

Thus, as always, it is important to pack the sectors as tight as possible in order to use the less amount of time slots possible. A bin packing strategy is applied to the list of dying sectors, using the water debits as item quantities (because this is the crucial information), and the maximum water pump capacity as the size of the one-dimensional bins. The bin packing strategy consists on a bin-oriented function, where a list is given as input and a bin with items is returned.

Because time simply cannot be wasted, the outputted combination is tested on whether all of its comprising items are in need of fertilization. If that is the case, the fertilization pump will be on during the irrigation of this bin. If the bin does not reach maximum water pump capacity, the algorithm will still test whether any other sector in the entire irrigation network fits in this combination or not (note: this test is performed only for one sector to fit inside the bin and not more than one at a time in order to reduce time consumption).

The irrigation of this bin ends when one of its containing sectors reaches minimum water level for survival until the cheap interval. That irrigation time is allocated, the safe sector is removed from the list of dying ones, and we start again with the usage of the bin packing function, this time with a list lacking an item. This phase ends when the list is empty.

If we reach the case where we no longer have time slots to save a given culture, the algorithm will assume that the problem cannot have a valid solution with the specified starting parameters.

4.3.3 Phase 2 - Fertilization Packing

This phase is reached only if all cultures survived until the moment in hand. If that is indeed the case, the algorithm commences to focus on the fertilization needs of all cultures. Thus, it starts by creating a list of all sectors which need to be fertilized. This list is created based on previous irrigation and the fertilization daily needs of each culture. Those which either do not have any need of fertilization or were already fertilized for this day are left out of this part of the algorithm.

Again, in an identical scheme as in the previous phase, we use a bin-oriented function which outputs a bin with items, given the list of fertilization needing sectors. **The goal of this packing is to fulfill all fertilization needs**. Once the bin is given, the model proceeds on finding the limits of this bin's irrigation, and supposing the irrigation of this bin occurs throughout the next time slots, it looks for the first sector to fulfill its fertilization needs and the first sector that will pass its maximum water level. Comparing the times which are involved in this search, it reaches to one of these possible conclusions about this bin's irrigation:

1. One of the sectors will fulfill its fertilization needs before any other major event.
2. Maximum irrigation level is reached on one of the sectors, i.e before fulfilling any fertilization need of the items in the bin.
3. Cheap time slots run out before the goal is obtained.

This is where this phase differs from the previous one. **Phase 1** is an urgent step of the algorithm, where we have no other option but to try and irrigate all dying cultures, without much thinking. Now, we are dealing with the decisions which will determine how optimal the solution will be. We have three possible scenarios that need to be properly handled.

4.3.3.1 Scenario 1 - some fertilization need is fulfilled:

This is the best possible scenario. Indeed, if a sector fulfills its fertilization needs without any other sector having reached its maximum water level and without running out of cheap time slots, the bin irrigation terminates in the most successful way possible, given that the objective of this phase is fulfilling all fertilization needs. The irrigation is allocated, finishing in the moment where the fertilization need of that first sector is fulfilled. The mentioned sector is, therefore, removed from the initial sectors list, and we repeat the process of asking the packing function for a tight bin of items, this time with an item less. In this scenario, there are no tough decisions to take care of.

4.3.3.2 Scenario 2 - maximum water level:

In this scenario, we come into a more interesting crossroad. Indeed, for a sector to reach its maximum irrigation level means that, from that moment on, the irrigation for that culture will just be a waste of water (until, evidently, the water level drops from the maximum level due to evaporation and transpiration). However, the culture still has fertilization needs to fulfill. In the real world, although sometimes it is absolutely necessary, fertilizing when the culture is saturated can lead to fertilizer losses as well. Thus, stopping the irrigation at maximum water level will in general be preferred.

Furthermore, the fertilization needs are not as urgent as irrigation needs. Indeed, though in the problem addressed by this thesis we are dealing with fertilization daily values, it does no harm to the culture to under-fertilize once in a while, as long as the fertilization sum over time stays roughly the same, i.e as long as each under-fertilization is compensated by a later or previous over-fertilization. That being said, the decision relies on how the fertilization needs are throughout the remaining days. If those signal that the sector is entering a critical fertilization deficit, the irrigation proceeds until another event comes up. If not, the irrigation is stopped, the remaining fertilization need of that day is diluted over the following days, the sector is removed from the initial list and the cycle starts over again with another bin.

4.3.3.3 Scenario 3 - cheap time slots run out:

This is the toughest decision. Evidently, this algorithm is not supposed to loose much time on thinking. However, the right balance of simplicity and accuracy has to be met. So many factors are now considered in order to assess what will be the best option. Although cheap time slots just ran out, one cannot simply stop the irrigation if there are still cultures with great need of irrigation.

For instance, if a given culture will not survive until the next cheap interval, it is not a possibility to delay its irrigation. This raises a question regarding the way bins are organized: during bin packing, should we take into account the fact that certain cultures have a greater urgency for being irrigated? This would require the effort of constantly assessing which cultures will die first, and deciding a priority between this urgency and the water pump maximization. As far as bin packing theory goes, drifting from the sole purpose of bin filling maximization leads without question to more used bins, which translates in the problem addressed by this thesis into more time slots of irrigation, leading to greater energy consumption, as well as more frequently shifting from cheap intervals to more expensive ones. To sum up, in the case where not all cultures are safe until the next cheap interval, the model will have no other valid option but to proceed with the irrigation and consume energy in non-cheap tariffs.

If no culture risks being dead until the next cheap interval, or if the previous condition was met but the irrigation was extended through expensive time slots until all cultures were out of their critical zones, a new challenge is faced. Indeed, it is still not quite linear whether or not the irrigation should continue. As previously mentioned, should fertilization be postponed? Should some cultures be irrigated until the ideal level is met? Will the following cheap interval be smaller than the current one? Will it be bigger? These variables should all enter into the decision equation.

4.3.4 Phase 3 - Irrigation-only Packing

If **Phase 2** ends up in the fulfillment of all cultures fertilization needs, we proceed to a third phase: **Irrigation-only Packing**. Indeed, what there is left to do is pure irrigation of sectors. Some sectors were already irrigated until at least their ideal levels. This phase will firstly use a bin packing strategy to irrigate all sectors until their ideal levels are reached. Then, it will proceed to their desired maximum levels of water in the soil. There are also various scenarios relating this phase:

1. The ideal level of a given culture is reached.
2. A desired maximum level of a given culture is reached.
3. Cheap time slots run out.

Though these are quite less complicated scenarios compared to the previous phase, there are still decisions needed to be made.

4.3.4.1 Scenario 1 - an ideal level is reached

When the ideal level of a given culture is reached, the irrigation of the current sectors combination is stopped, and that sector is removed from the list that is being fed to the bin packing function. This constitutes the normal behaviour of this phase. When all ideals are met, the list is initialized again with all sectors, because the algorithm will now aim at the maximum water levels of each culture.

4.3.4.2 Scenario 2 - a desired maximum level is reached

First of all, a definition for the desired maximum level must be given. This corresponds to the value which is desirable to leave in a given culture, so as to be close to the ideal level as much as possible. This has the cost of reaching out to the future cheap interval and evaluate the water level evolution until that time.

For instance, if the ideal water level of a given culture is 40 mm, and from a given cheap interval to the next one the culture is expected to lose 30 mm, then it would be an optimal irrigation if in the first cheap interval the water level was 55 mm, so as to reach the following cheap interval with 25 mm. This

means that, from one interval to the other, it will perfectly surround the ideal water level. Thus, in this case, 55 mm would be the desired maximum level. This method, evidently, has its risks. Indeed, since the model doesn't look back, a decision to irrigate less than what is possible (in this case, if the actual maximum possible level were to be 100 mm, for example) could potentially lead to the culture's dead, notwithstanding the fact that saving the given culture was actually a possibility.

Reaching the desired maximum level of a given culture happens when all ideal levels were already met and the algorithm is now currently going after reaching desired maximum water levels on all sectors. The same is done as in the previous scenario: the combination irrigation is stopped, and that sector is removed from the current sectors list that is being served as input to the bin packing function. When all desired levels are met, **Phase 3** will end and the cycle will terminate as well, proceeding to the next cheap interval.

4.3.4.3 Scenario 3 - cheap time slots run out

When cheap time slots run out, the model presents itself with yet another decision to be made. Indeed, lots of things need to be asserted:

- If irrigation stops now, will all cultures survive until the next cheap interval?
- How big is the next cheap interval compared to this one?
- How will the weather influence the culture in the next days?
- Is it worth spending expensive time slots in order to get certain cultures closer to their ideal levels?

4.3.5 Packing strategy and *filling* parameter

As a general test of the algorithm, 4 different bin packing strategies were used to solve the same problem setup, where a *filling* parameter was varied between 0.7 and 1. This parameter indicates which maximum percentage of each bin will be used, unless the last strategy is used. Each of the packing strategies corresponds to a bin-oriented algorithm which will receive the available items and provide the best possible first bin of the packing solution, according to the algorithm.

These are the 4 used bin packing strategies:

- First *First Fit Decreasing* Combo: this is the simplest among these 4 algorithms. As already explained, the First Fit Decreasing packing strategy inserts an item in the first bin with space for it. The *combo* algorithm will pack the items in this fashion and return the first of all the combos.
- Best *First Fit Decreasing* Combo: similar to the previous one, it involves just one extra step. Instead of returning the first of all the bins, it will return the most filled bin in the pack.
- Best *Best Fit Decreasing* Combo: this *combo* algorithm has the same logic of the previous one, except for the fact that it uses the Best Fit Decreasing packing strategy instead.
- Best *Bin Completion* Combo: although this *combo* algorithm is not identical to the Bin Completion approach, it is indeed inspired in it. The algorithm is quite simple, with its limitations as well. First of all, as all the previously mentioned ones, it sorts the items by decreasing order of size. The immediate decision is to include the biggest item in the returning bin; if no other item can fit with it in a bin, the returned combo will only have that one big item. Now, if the smallest possible item makes a good enough match with the biggest one (i.e. they fit together and they pass the *filling* parameter percentage), the combo is done. If not, the order of the items will be reversed - from smaller to bigger ones. Having the biggest item A and the smallest unexplored item B, if there is any item C bigger than B which makes a good enough possible bin match - A, B and C - then that combo is returned. If that's not the case for any of the items, then a similar approach is done but with four items - A, B, C and D, where C and D fulfill the duty of the former approach's C item. Finally, if no good enough possible bin match is found, the best combo discovered so far will be returned.

To assess the different algorithms, some simple cost functions were created, thus providing a way to compare between them. It is worth noting that each cost function does not necessarily correspond to a given physical unit, rather they're just quantification tools relevant for the problem in hand.

4.3.5.1 Testing setup

For the tests in question, 15 sectors were randomly generated with the following characteristics:

ID	Debit	MinIrrig	IdealIrrig	MaxIrrig	InitialBalance	FertPerDay
1	20	10	30	50	30	3
2	30	15	30	40	30	1
3	35	25	40	50	40	1.5
4	45	30	55	80	55	1.5
5	50	20	70	100	70	4
6	15	10	30	50	30	5
7	25	40	50	75	50	4
8	45	10	25	50	25	1
9	30	10	50	70	50	1.5
10	10	20	35	50	35	3
11	15	15	25	30	25	1
12	25	20	25	30	25	3
13	30	30	60	70	60	2.5
14	15	10	20	45	20	2
15	5	5	20	30	20	2

Figure 17: Test setup's sector characteristics.

Each sector is starting with a water balance level equal to its ideal irrigation level. Maximum irrigation corresponds to the maximum water balance, at which point irrigating the sector's crop is no longer useful, as it reached maximum absorption capacity. The minimum irrigation corresponds to the point when the sector's culture dies of thirst. The unit of these mentioned fields is *mm* - *millimeters*. *FertPerDay* is the crop's fertilizer need in a day, in liters. *Debit* corresponds to the amount of water the sector's valve can output to its crop, in m^3/h. *ID* is used to identify each sector.

The irrigation network is equipped with a water pump with a nominal flow of $50m^3/h$ and an injection pump capable of inserting 200 liters of fertilizer to the network per hour. Daily water level losses are provided in *millimeters* and inserted into the algorithm as hourly evapotranspiration:

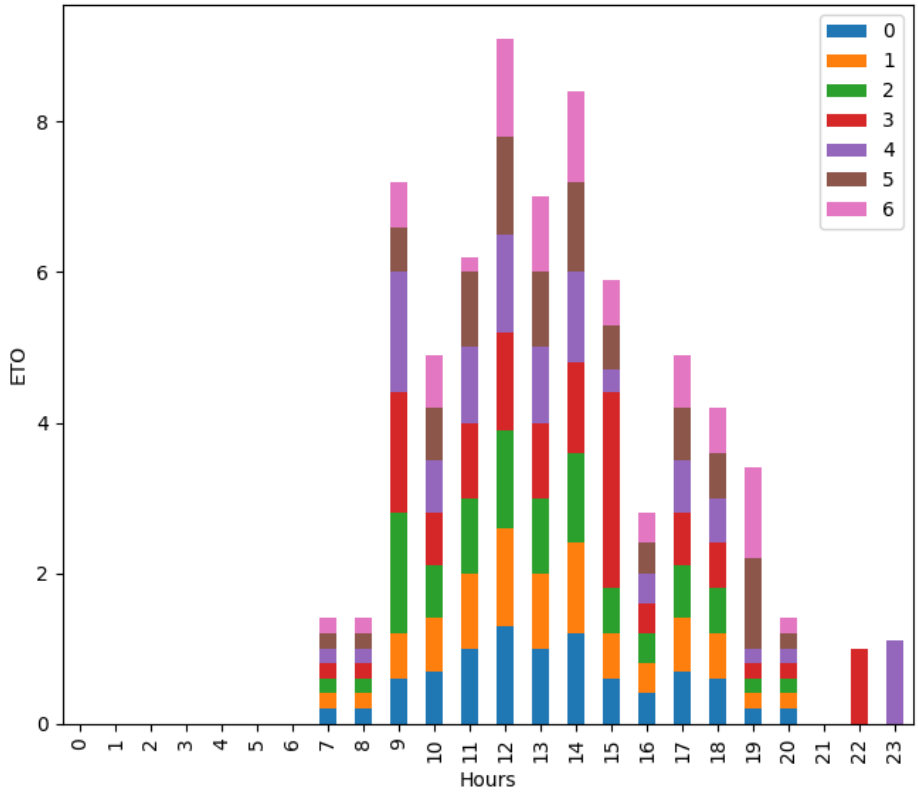

Figure 18: Test setup's hourly evapotranspiration throughout a week, where each color represents a day.

In this previous graph, each color represents a day in the week, ranging from 0 to 6, and the size of the bar represents the amount of evapotranspiration lost on that day on that given hour. Finally, the energy tariffs are generated as cheap or not intervals throughout the hours of the week:

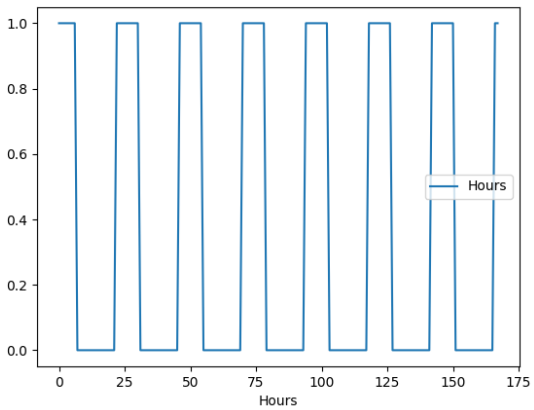

Figure 19: Test setup's cheap or not intervals.

In the figure above, hours with value 1 correspond to cheap hours, while value 0 indicates expensive hours. It displays a bi-hourly energy tariff, where hours are cheap from 0AM to 7AM and from 10PM to 12PM in all days.

4.3.5.2 Energy Cost - Computing

The following algorithm provides the way of quantifying the energy part of the problem in each resulting irrigation plan:

Algorithm 1 Energy Cost Computation

```
1: cost = 0
2: for each irrigation in week_irrigation_moments do
3:    for second = irrigation_start_sec until irrigation_final_sec do
4:       if second is cheap then
5:          cost = cost + 1
6:       else
7:          cost = cost + 2
8:       end if
9:    end for
10: end for
11: return cost
```

Each used second will have either value 1 or 2, if the second is inside a cheap interval or not, respectively.

4.3.5.3 Balance Cost - Computing

The following algorithm provides the way of quantifying the water balance part of the problem in each resulting irrigation plan:

Algorithm 2 Balance Cost Computation

```
1: cost = 0
2: for each sector in network_sectors do
3:    second = 0
4:    while second < final_week_second do
5:       cost = cost + ABS_VALUE(sector_balance_at_second(second) - sector_ideal_level) /
         sector_ideal_level
6:       second = second + 30
7:    end while
8: end for
9: return cost
```

Computation of a culture's water level at a given second is time consuming. Thus, each cycle increments 30 seconds, which does not sacrifice the quantification and reduces time consumption by a factor of 30.

4.3.5.4 Pump Cost - Computing

pump cost is quantified by looking at the mean pump efficiency throughout the irrigation plan. It does not require detailed explanation.

4.3.5.5 Computed Values

The results were inserted in graphs, where the X axis is always the *filling* parameter. The following figure displays the performance of each of the strategies regarding energy cost:

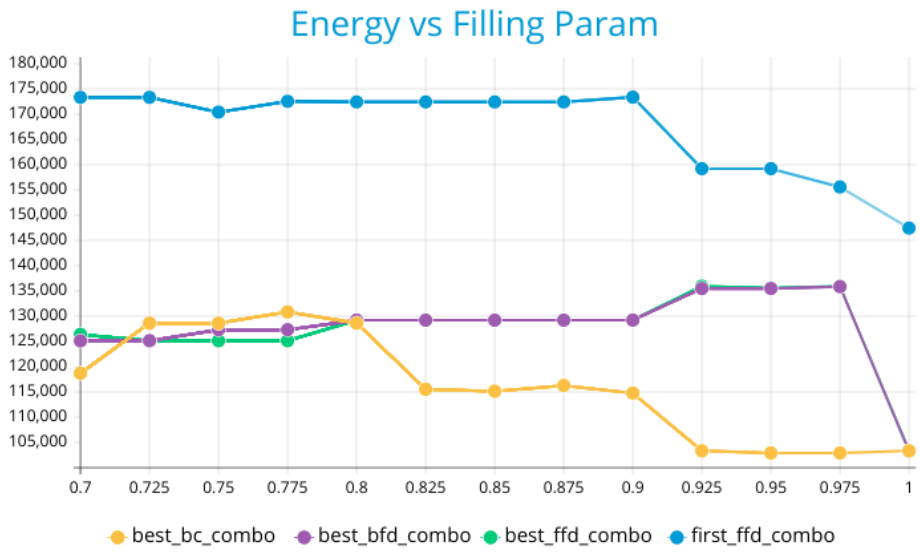

Figure 20: Energy cost with *filling* variations, for the four tested packing strategies.

As it was expected, the First *FFD* Combo algorithm presents the worse performance in all cases. The simplicity of the algorithm, as already explained, does not ensure the return of the best possible combo. This leads to more irrigation time, which corresponds to a greater energy cost. Also, not using the best combos makes it more probable to perform watering procedures outside cheap intervals, in order for some sectors to survive. In general, energy performance increases with the tested parameter.

As for the other three algorithms, best performance heavily depends on the *filling* parameter. When this parameter is 0.7, Best *BC* (Bin Completion) Combo presents the best performance, followed by Best *BFD* Combo and then Best *FFD* Combo. But when $filling = 0.775$, Best *FFD* Combo is more efficient, followed by Best *BFD* Combo and Best *BC* Combo, respectively. In general, Best *FFD* Combo and Best *BFD* Combo tend to have very similar results, even equal ones for greater values of *filling* parameter, and its performance mainly decreases with the increase of the tested value. The Best *BC* Combo presents the best energy performance for the majority of cases, and its performance improves from $filling = 0.775$ to $filling = 1$. Interestingly enough, these three algorithms present the same performance when *filling* is 1, corresponding to the lowest energy cost among the tests.

As a final conclusion, for the tested package strategies, the lowest energy cost is obtained with $filling = 1$. In general, the Best *BC* Combo achieved the best results.

Following this, the next figure displays the performance of each of the strategies regarding water balance cost:

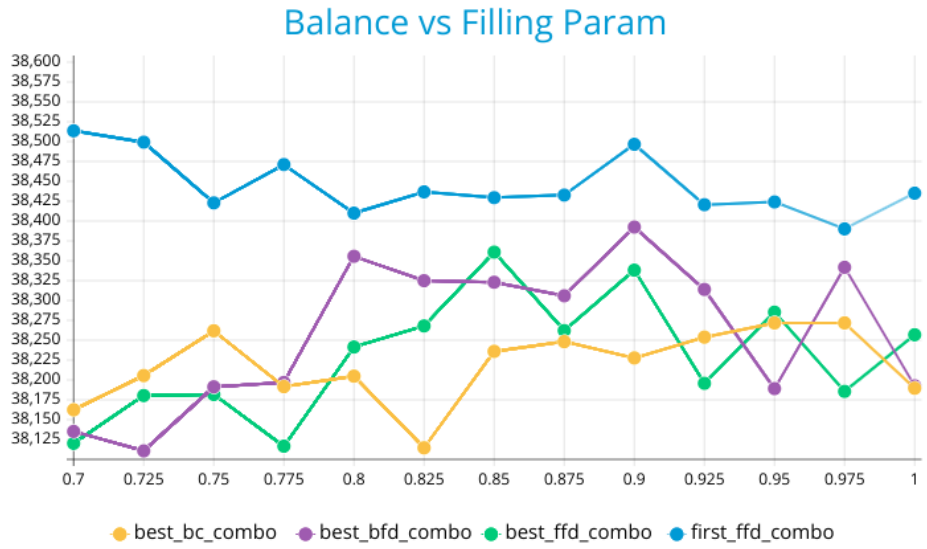

Figure 21: Water Balance cost with *filling* variations, for the four tested packing strategies.

Once again, the First *FFD* Combo packing strategy provides greatest cost throughout the *filling* range. The Best *BFD* Combo algorithm was able to achieve the global best performance when $filling = 0.725$, although in most cases it has greater balance cost than Best *FFD* Combo and Best *BC* Combo algorithms. With $filling = 1$, both Best *BFD* Combo and Best *BC* Combo provide the best balance performance.

Overall, there seems to be no correlation between *filling* parameter and balance cost for all tested packing strategy.

The following graph displays the performance of each of the strategies regarding pump cost, i.e. pump efficiency:

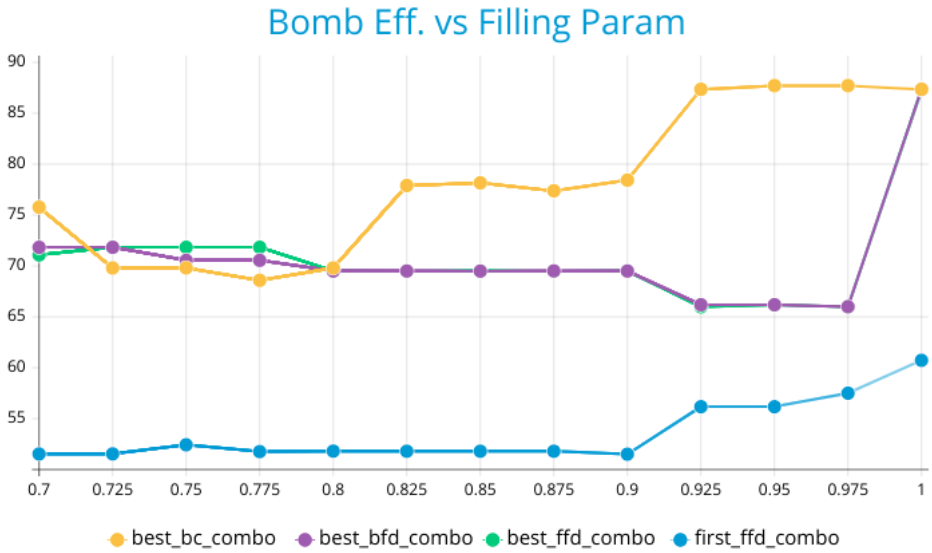

Figure 22: pump cost with *filling* variations, for the four tested packing strategies.

As expected, the First *FFD* Combo strategy achieved the lowest performance, i.e. the lowest mean pump efficiency, in all tests, though performance increased with *filling* parameter. The behaviour of the other three packing strategies was similar here to the one presented in the energy cost graph. Indeed,

Best *FFD* Combo and Best *BFD* Combo have very similar performance, Best *BC* Combo algorithm roughly achieves the greatest pump efficiencies, and the three algorithms present the same performance when $filling = 1$.

Finally, the following figure displays the time the full algorithm took using each of the four given packing strategies:

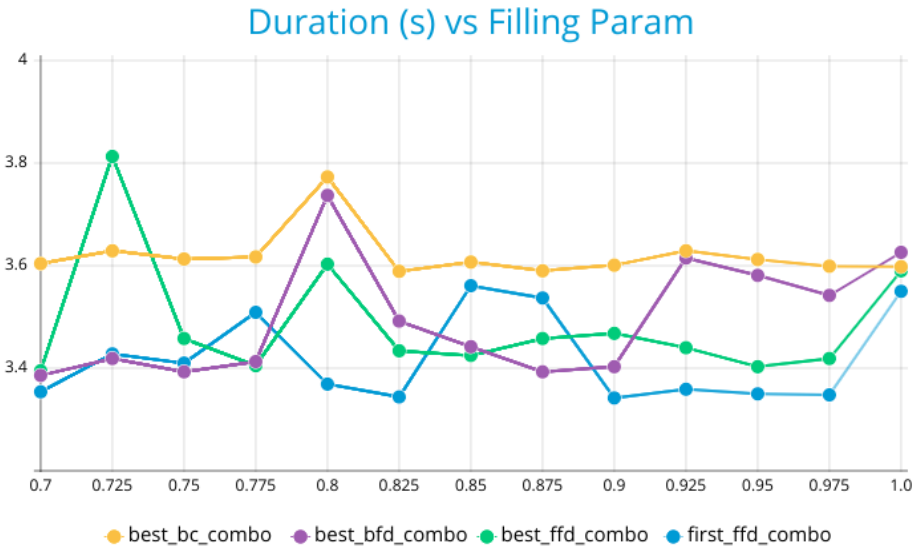

Figure 23: Full algorithm duration with *filling* variations, running each of the four tested packing strategies.

Given First *FFD* Combo packing strategy's simplicity, it provides the overall algorithm with the smallest duration in most tests. The Best *BC* Combo algorithm presents the greatest running duration, except when $filling = 0.725$ (it is exceeded by Best *FFD* Combo) and when $filling = 1$ (it is exceeded by Best *BFD* Combo). It is also worth noting that its running duration is roughly constant throughout *filling* parameter variation, much more than in the other ones. The Best *FFD* Combo and the Best *BFD* Combo algorithms present a more less random behaviour regarding *filling* variation. Overall, the differences in running duration are not much relevant: minimum duration and maximum duration of all tests only differ about 0.5 seconds.

Weighting all results, the Best *BC* Combo packing strategy is the one chosen for the algorithm, with $filling = 1$.

4.3.5.6 Crop's moisture level

For the test setup in question, the water balance level of some sectors' cultures will be visually represented, as they tell much about the output of the algorithm.

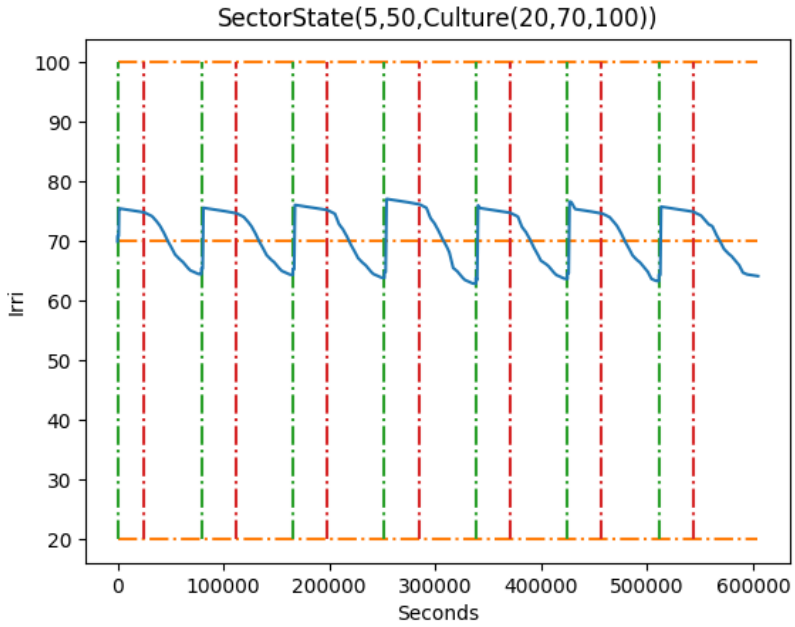

Figure 24: Water balance of sector 5's culture.

The graph displays lots of information:

- Energy tariffs: green vertical lines signal the start of a cheap energy interval, while red vertical lines correspond to the end of that cheap interval and the consequential start of an expensive energy interval.
- Water balance boundaries: from top to bottom, the orange lines represent the maximum water level, the ideal water balance point and the minimum water level for crop survival.
- Crop's water balance: the blue line represents the crop's expected water level, in *millimeters*, for each second of the coming week, which is the resulting product of the algorithm generated irrigation plan and evapotranspiration.
- Sector and culture's characteristics: the graph's title is in the following format: *SectorState(ID, Debit, Culture(Minimum Water Level, Ideal Water Level, Maximum Water Level))*.

Specifically regarding this sector, various details can be examined:

- Crop's irrigation is taking place only in the beginning of cheap intervals.
- Although maximum and minimum levels have values 100 and 20, respectively, the water level is never over 80mm or below 60mm, noticing the expected behaviour of the water level balancing around the ideal line, similar to a sinusoidal or a pulse wave. This ensures a constant approximation to the ideal level.
- At each irrigation moment, even though more water could still be given to the sector, it is not so and the network only provides enough water for the culture to balance around the ideal point until the next cheap interval comes. Indeed, the algorithm predicts when the water balance level will be symmetric around the ideal line from a cheap interval to the other one. Thus, water is saved without compromising crop.

Another interesting example is the one provided by sector 11:

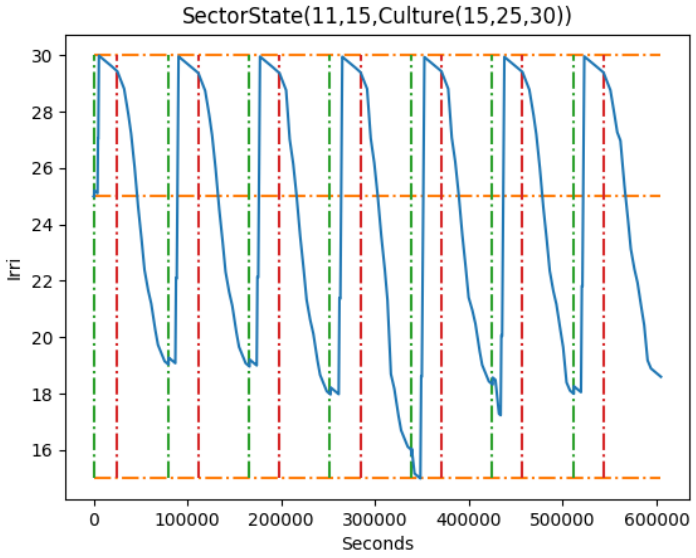

Figure 25: Water balance of sector 11's culture.

Noticeably, contrasting with sector 5 which had a possible balance interval of 80mm (and a resulting water balance distribution changing in an interval of about 10mm), sector 11 only has a possible water level interval of 15mm. Furthermore, the ideal and maximum levels only distance 5mm, and sector's debit is $15m^3/h$, less than a third of sector 5's debit, $50m^3/h$. This means that a sinusoidal balance around the ideal line could only have a range of 10mm. Due to these characteristics, we notice some stress in the crop, i.e. moments where water level is close to or even reaching the minimum balance level.

In most of the days, the water level drops to about 18 or 19. As the cheap energy time interval begins, it can be seen a small irrigation right at the start of it. This corresponds to the sector's fertilization moment. As the algorithm entails, such moments always happen in the beginning of the cheap intervals.

After that small initial fertirrigation, crop receives enough water to reach the maximum water level. This happens because the algorithm knows that in the next cheap interval the culture will have less water than the desired one for balance around the ideal line. The most critical situation happens while passing from the fourth to the fifth day, when the sector's crop drops its water level to the minimum balance level. Even though the culture almost drops below the minimum level, resulting in its death, the algorithm succeeded on assuring not only the crop's survival but also the exclusive irrigation in cheap intervals.

The following graph displays the water balance of sector 12's culture:

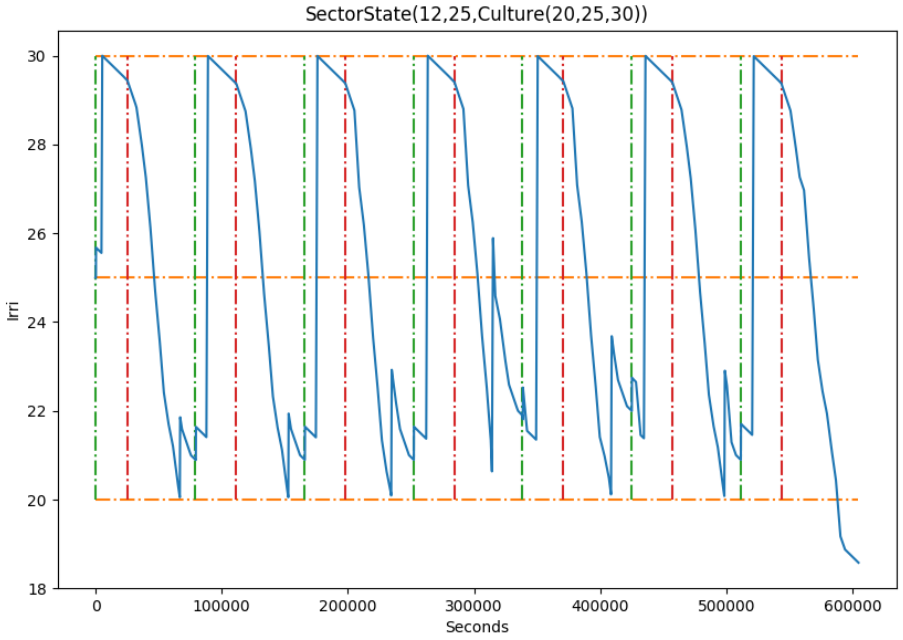

Figure 26: Water balance of sector 12's culture.

Sector 12 demonstrates how the algorithm behaves when dealing with the most critical crops. The ideal water level is right between minimum and maximum levels, which provides a favourable situation for balancing around the ideal line. However, the possible water balance interval is only 10mm. It can be seen that the culture is not able to survive without irrigation in expensive intervals. In the moments where the crop reaches the lowest water levels, it can usually be seen two spikes which precede the irrigation moment to the maximum level: one corresponds to the urgent watering of the culture in an expensive interval in order to save it from death; the other one is the fertilization moment of the crop. Most importantly, the algorithm does not allow the death of the culture.

Finally, it is worth looking at the water level of sector 7 across the week:

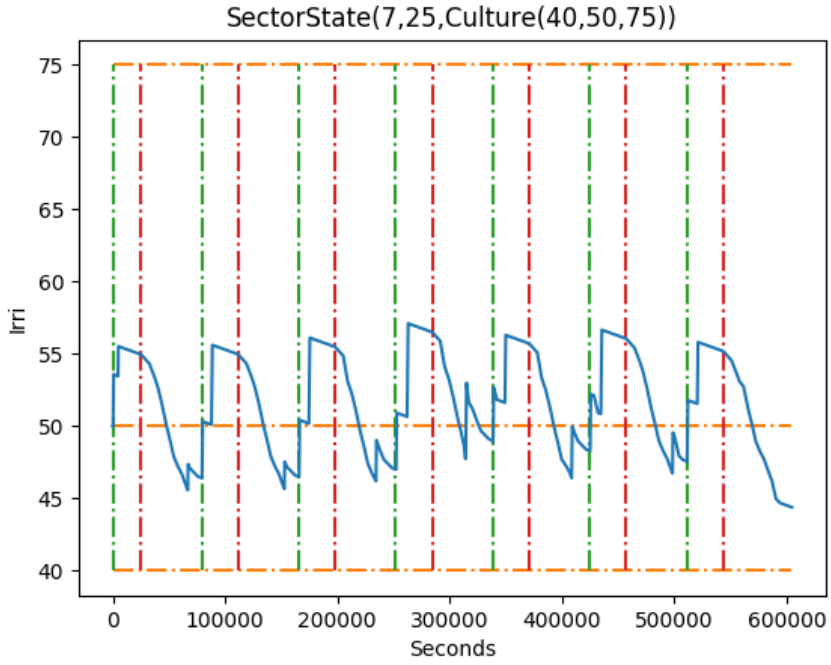

Figure 27: Water balance of sector 7's culture.

In this peculiar case, we see that there seems to be some unnecessary irrigation moments in the expensive intervals. Indeed, the sector is not at all in danger, and even if it were to be close to death, there is a lot of space for cheap irrigation until the maximum level is reached. However, the expensive irrigation moments coincide precisely with the expensive irrigation moments of sector 12, which was previously presented. In fact, both sector 7 and 12 have a debit of $25m^3/h$, which combined maximize pump efficiency in this specific case. This is a demonstration of how the *Bin Packing for Survival* phase works. Indeed, due to the fact that sector 12 is in danger of going below the minimum water balance level, the water pump will have to work during an expensive interval. So, if that is unavoidable, at least the pump will try to work in its maximum efficiency and irrigate some other sectors (in this case, sector 7), resulting in an overall water saving and energy consumption minimization.

4.3.6 General Testing

In the following section, the algorithm, using the Best *BC* Combo packing strategy and *filling = 1*, will be tested in various conditions, in order to evaluate its time complexity.

4.3.6.1 Time behaviour

The algorithm went through 156 tests. Each of those tests used the same fixed 15 irrigation sectors, and the amount of randomly generated sectors varied. With the variation of sectors, the size of the water pump and injection pump also varied, as to accommodate the system's needs. Cheap intervals were always the same, as well as the values of evapotranspiration.

The following graph shows the variation of running duration with the number of sectors (without accounting for the variation of any other variable):

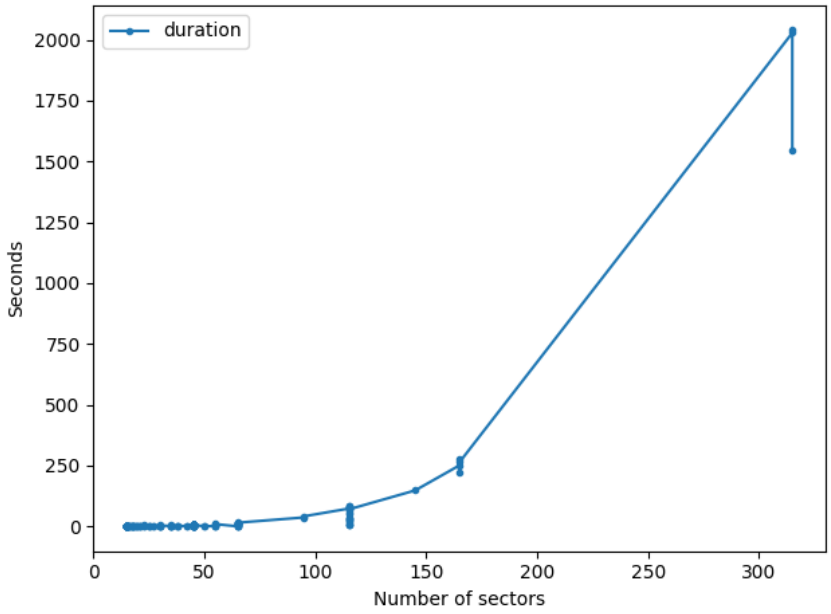

Figure 28: Running duration variation with the number of sectors.

Noteworthy, it seems the growth of running duration is exponential with the increase of number of sectors. For more than 300 sectors, one can see that the algorithm took more than 2000 seconds in one of the tests, which is more than 30 minutes of running time. Ignoring tests with more than 300 sectors, one can see more clearly the algorithm's time complexity:

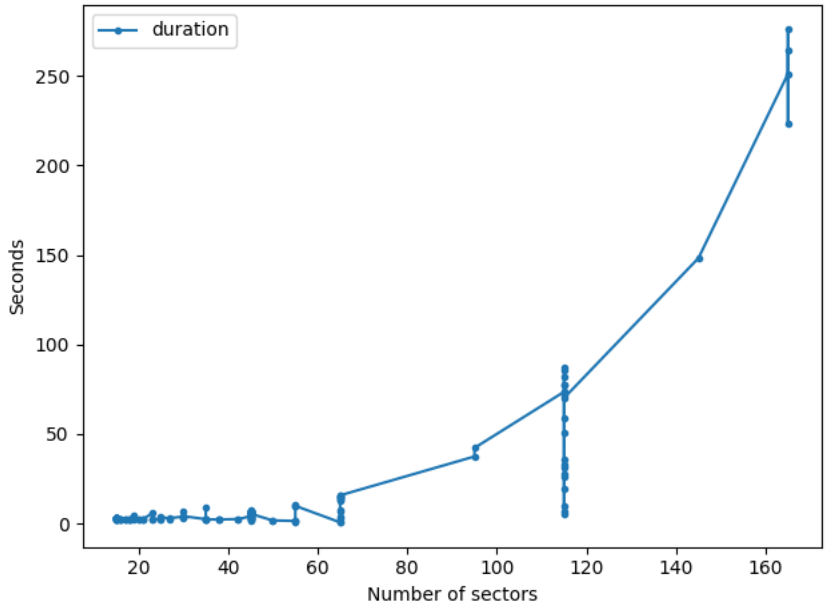

Figure 29: Running duration variation with the number of sectors, for tests with a number of sectors below 300.

Indeed, the increase of duration entails an exponential time complexity with the increase of number of sectors. Some values of number of sectors went through more testing. The duration variation in these cases is mostly due to the variation of the water pump debit, which turns out to be rather important for the time complexity. Indeed, for 115 sectors one sees that tests' running duration varied almost 100

seconds, which is quite significant.

The effect of water pump maximum debit variation on the algorithm's time complexity is assessed in the following images. The first graph shows various tests for different values of water pump debit in a system of 15 sectors:

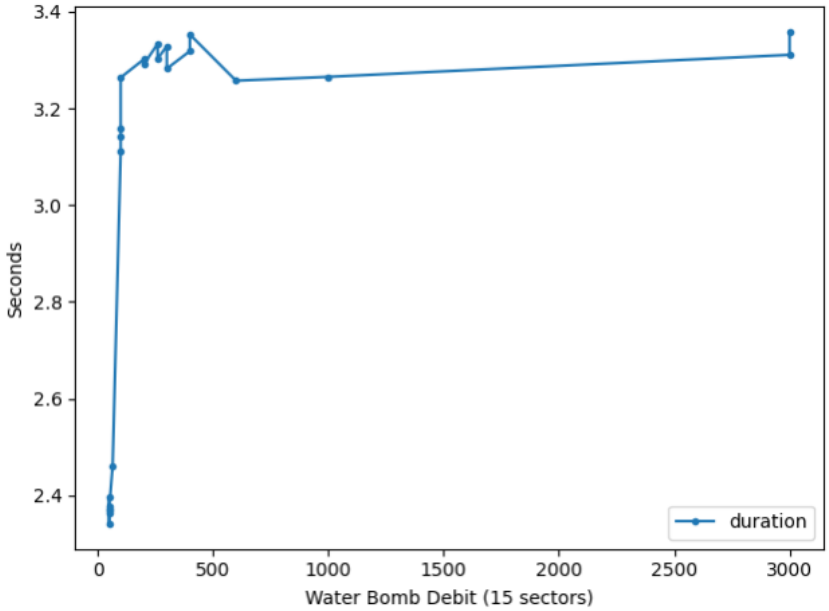

Figure 30: Running duration variation with the debit of the water pump, for setups of 15 sectors.

Running duration increases rapidly (about 1 second) for values of water pump debit increasing from 50 to 200. However, it seems to reach some plateau and time variation becomes irrelevant. One can see the same behaviour in tests with other number of sectors, such as the tests of the 45 sectors system:

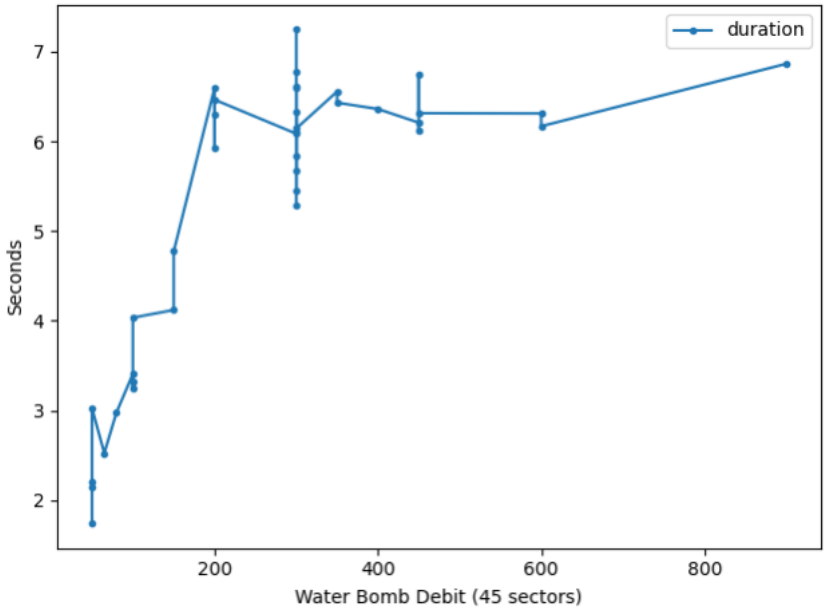

Figure 31: Running duration variation with the debit of the water pump, for setups of 45 sectors.

The plateau of the 45 sectors test seems to be around 7 seconds, twice that of the previous graph.

This graph also entails that the running duration variation between tests with the same conditions but different randomly generated sectors can also be quite substantial, since in this setup the variation can go up to 2 seconds. The next image shows the duration variation for a system of 115 sectors:

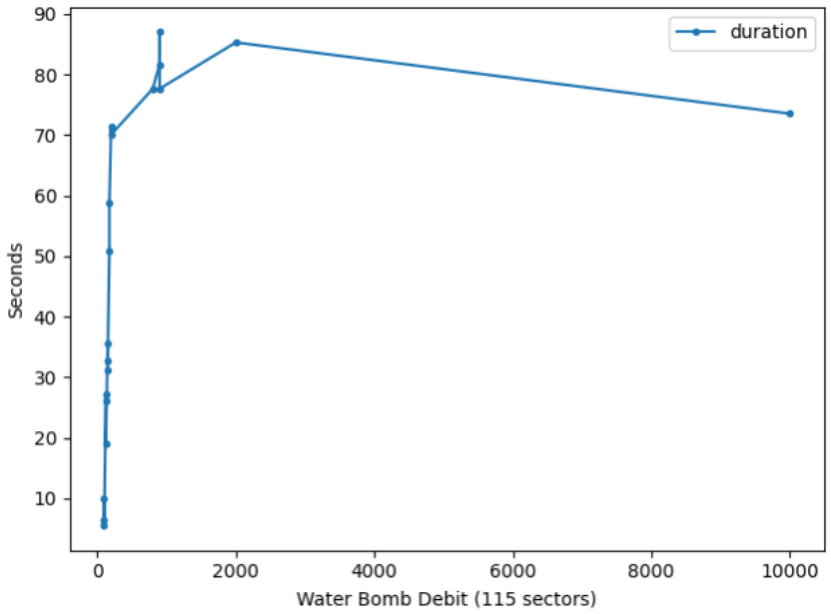

Figure 32: Running duration variation with the debit of the water pump, for setups of 115 sectors.

As expected, this graph shows the same plateau behaviour as the previous ones. Test duration variations seem now to reach more than 10 seconds. If compared with the plateau level, this graph shows a variation no bigger than 15%, while the previous graph showed a variation of almost 30% compared to its maximum running duration.

The final image displays the results of just 3 tests for a system with 315 sectors:

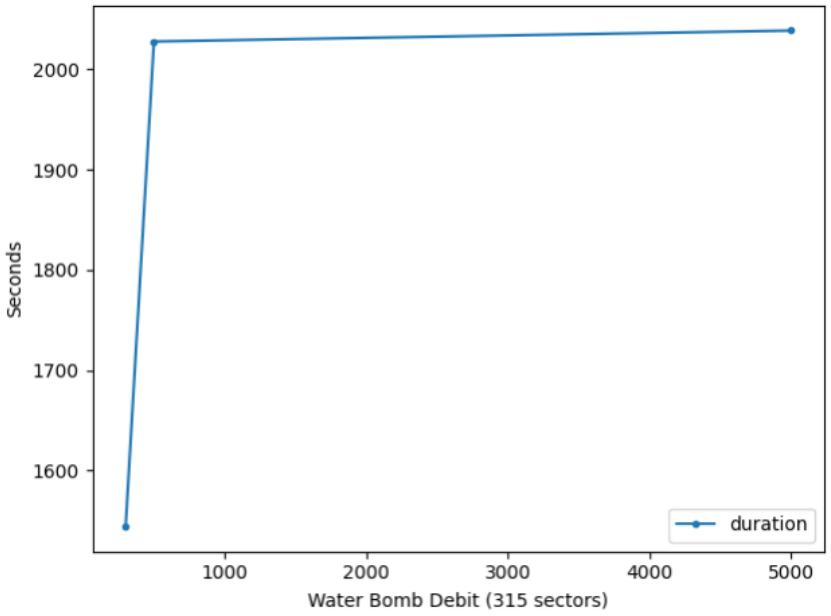

Figure 33: Running duration variation with the debit of the water pump, for setups of 315 sectors.

Although there is not much information for a full corroboration of the expected behaviour, it still seems to follow the same typical pattern. One can also see the magnitude of the running duration for such a system, reaching time registrations of more than 2000 seconds, which is more than half an hour.

Lastly, the following graph will shed a little more light on the time variation in tests with the same conditions but different sectors:

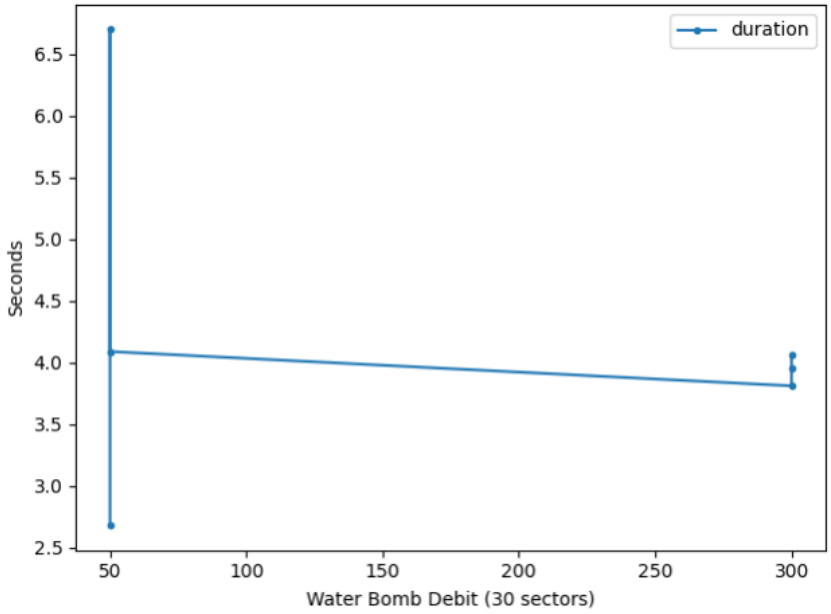

Figure 34: Running duration variation with the debit of the water pump, for setups of 30 sectors.

Indeed, and as in many other systems in testing, the duration variation in tests with the same number of sectors seems to decrease with the increase of water pump debit.

4.3.6.2 Packing time complexity

Given that we seem to have an exponential growth of time duration with the increase on the number of sectors in the system, more detailed testing was done as to understand the reason for this time complexity. The selected packing algorithm was divided into three different parts, and time was recorded in three different checkpoints. Each test corresponds to the event of creating one single combination of sectors derived from the entire system, i.e. the best combination.

The first tests aimed at analyzing the time variation with the increase of bin capacity (which corresponds to water pump debit in the overall problem), for a fixed number of items. The following images show the time variation in a system with 490 sectors and are representative of the behaviour presented in the other tested systems. The first image corresponds to the first checkpoint:

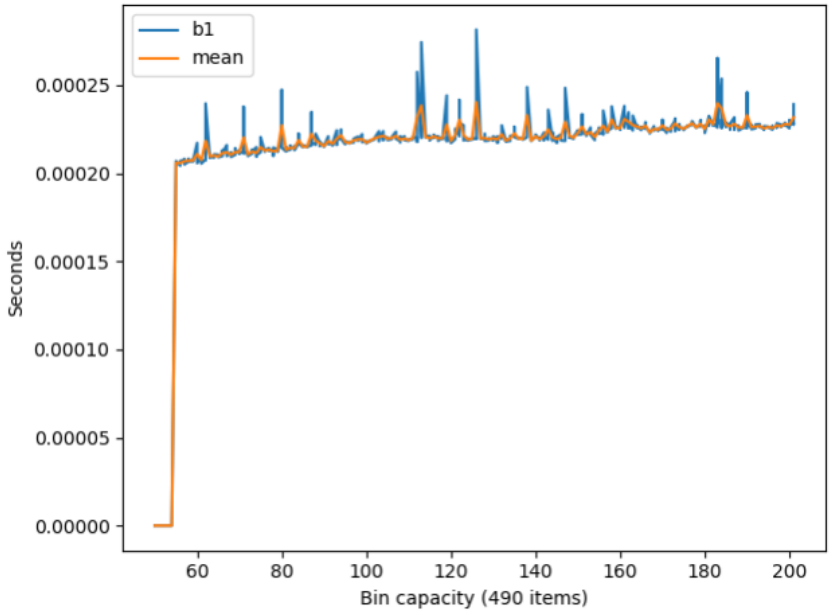

Figure 35: First checkpoint duration variation with bin capacity, for setups of 490 sectors.

The first checkpoint marks the point where the Best BC Combo strategy sorts sectors by decreasing order of debit, filters the ones which cannot fit the bin, and pops the sector with biggest debit. Naturally, since all sectors pass filtering in these tests, there is no relation between the steps before the first checkpoint and the value of bin capacity. Therefore, the almost constant distribution of duration and the small amount of wasted time was expected, and it is conformed by the previous graph.

The next image corresponds to the second checkpoint of the packing algorithm:

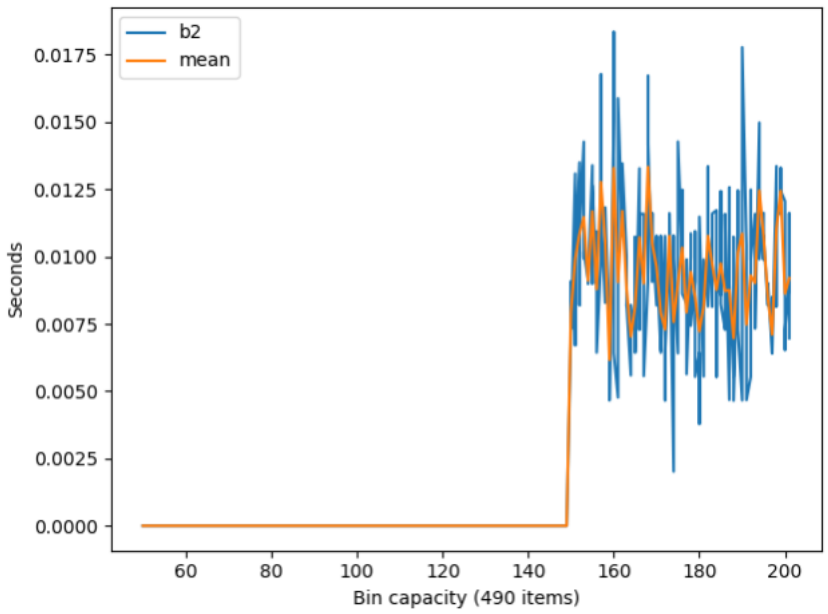

Figure 36: Second checkpoint duration variation with bin capacity, for setups of 490 sectors.

The second checkpoint marks the point where the packing strategy aims at filling the sector combination, which just has one sector given from the first checkpoint, with another two sectors. This relates

to the size of the bin, insofar as the number of possible combinations is related to the remaining space in the bin. The reason for the constant line in the zero until bin capacity reaches a value around 150 is the fact that only then can the algorithm pass the second checkpoint. Indeed, if an optimal combination of one, two or three items is reached, the algorithm will not reach the time checkpoint. After that, as expected, given that the algorithm now has enough space in the bin to try all combinations, the time duration is roughly constant.

Finally, this next figure corresponds to the third checkpoint:

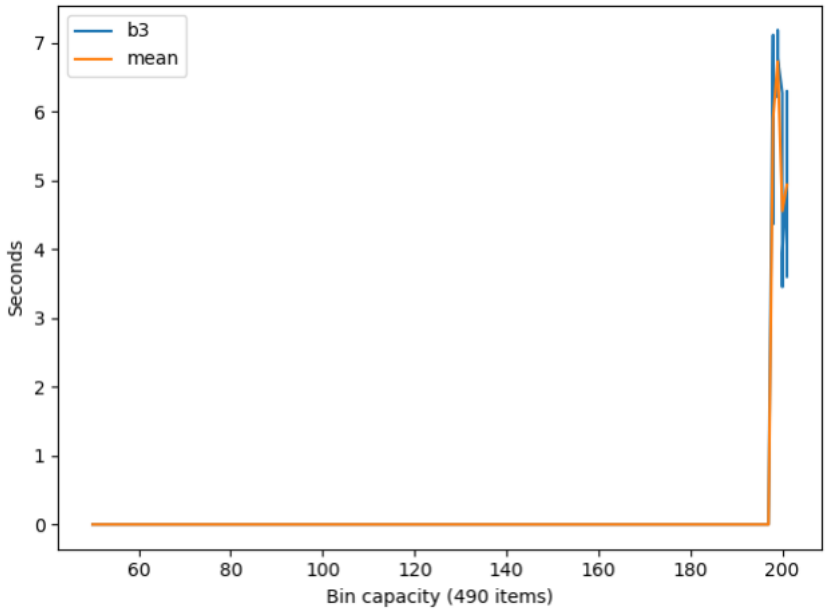

Figure 37: Third checkpoint duration variation with bin capacity, for setups of 490 sectors.

The third checkpoint is reached if the entire algorithm is passed. Given that that means to pass the test of 4 sector combination, it is natural results only appear in at around 200. Time continues to present itself constant with the bin capacity increase. The conclusion here is that bin capacity does not significantly impact the strategy's time complexity.

In the following three graphs, the same checkpoint times are analyzed, but this time it is the number of items which varies across tests, while bin capacity remains at 201. This value is not random, as it enables the algorithm to pass the three checkpoints, considering the conclusions taken above. The first figure corresponds to the first checkpoint:

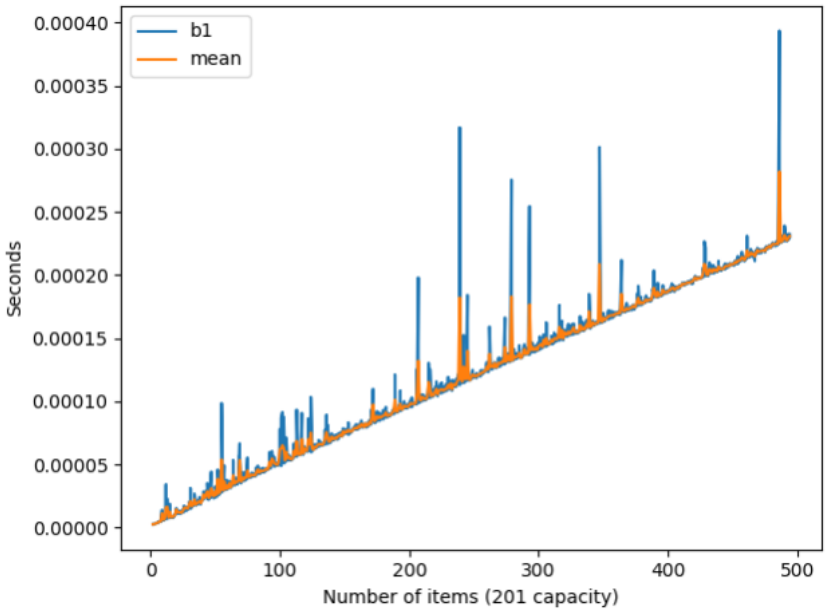

Figure 38: First checkpoint duration variation with number of sectors, for setups with a bin capacity of 201.

The duration variation increases linearly with the number of sectors. This corresponds to the process of sorting and filtering the sector list, which is a task with linear time complexity. The next graph presents the second checkpoint results, which present a more critical behaviour:

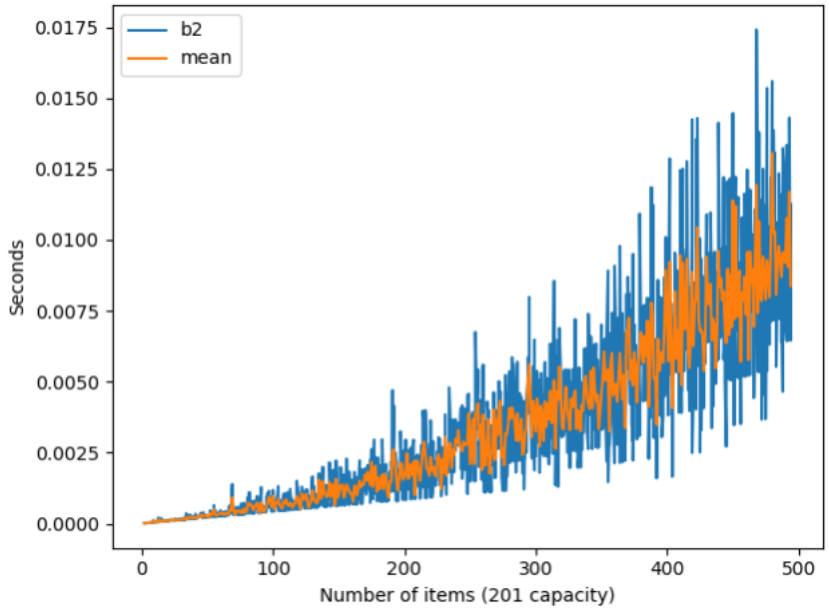

Figure 39: Second checkpoint duration variation with number of sectors, for setups with a bin capacity of 201.

Now the duration seems to take on an apparently exponential complexity. However, a closer look will lead to the conclusion that the graph is actually displaying a quadratic distribution. Indeed, the algorithm tests the various possible combinations of two items which may fit with the biggest item. This means that, in the scenario where there's enough space for 4 items (which is this testing scenario), the

algorithm will test almost $n\_items * n\_items$, which corresponds to the observed quadratic increase. The final graph corresponds to tests done for the third checkpoint:

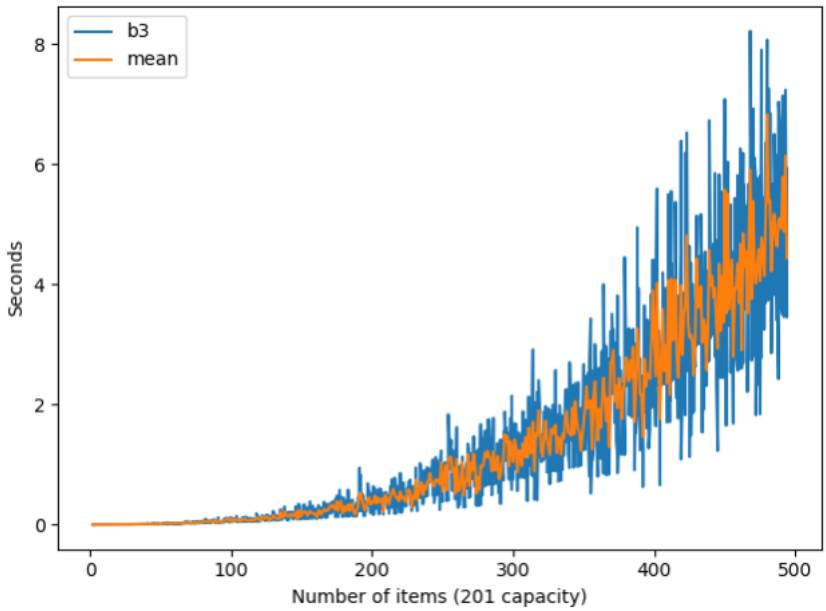

Figure 40: Third checkpoint duration variation with number of sectors, for setups with a bin capacity of 201.

Indeed, now we seem to be reaching a more pronounced increase on time duration. One can also see that time is reaching 8 seconds, which highly contrasts with the second checkpoint maximum of 0.0175 seconds. Now the algorithm will perform three embedded loops on the list of items, which correspond to a cubic distribution. Indeed, if one multiplies the mean second checkpoint duration for 500 items, which is about 0.01 seconds, by 500, one gets 5 seconds, which is roughly the mean third checkpoint duration for 500 items; if one multiplies the maximum second checkpoint duration for 500 items, which is about 0.0175 seconds, by 500, one gets 8.75 seconds, which is even more than the maximum third checkpoint duration. This corroborates the idea of a bin packing strategy with cubic time complexity, and rejects the possibility of having exponential distribution due to the selected packing strategy.

5 Conclusions and Future Possible Improvements

After analyzing all testing results - both to different strategies within the problem and to the problem as a whole - a few conclusions can be taken concerning the performance and usability of the developed algorithm and future improvements should be done both to the testing setup and to the packing strategy itself.

5.1 Algorithm performance

The algorithm manages to output a highly efficient week irrigation plan within a granularity of seconds, leveraging the complexity of expensive time slots of the week, the water pump's capacity and the fertilization needs and its constraints in the system. In most cases, where the system does not come close to having 100 sectors, the algorithm provides a quick answer to the problem. For systems with more sectors, the problem starts to take longer, but its increase is not exponential. Indeed, as results show, the selected packing strategy has a cubic time complexity. Given that this packing strategy has to be done at most to all the existing sectors (assuming duration remains constant, which it does not happen at all - as sectors are irrigated, less sectors need to be allocated in the next packing event), the algorithm would have a time complexity of x^4. Even if it turns out bigger than that, it will still be a polynomial increase and not an exponential one, which is outstandingly different.

The algorithm assures that no crop is going to die - provided there is a possible solution where the crop survives - even if that leads to a higher cost in energy. No cost is greater than loosing the crop. Furthermore, there is no risk of over-fertilizing sectors, which in many cases can lead to the death of the crop as well.

Though it can be seen that energy is saved as much as possible, due to preference of cheap intervals, there is a major problem in the selected packing strategy, which is not present in the remaining strategies, and which was spotted in the final testings. Indeed, the selected packing strategy only allocates at most 4 sectors into the bin, or water pump. This means that, for a system with a pump capable of irrigating the system's 10 sectors with greater debit, the pump will never be more filled than 40% of its capacity. Evidently, for a packing algorithm, this is a significant blunder. Fortunately, possible corrections are quite simple and may not increase time complexity at all. An example would be to proceed with a First Fit Decreasing packing strategy in the end of the selected packing algorithm. This would manage the correct and efficient filling of bins with great capacity and does not increase time complexity, for FFD strategy has linear time complexity. This would resolve the problem of very little filled bins.

The bin packing abstraction proves to be a good fit to the problem in hand. Although the irrigation problem has more complexity than the mere packing problem, the abstraction enabled the bin packing approach to be the core of the algorithm, while the remaining consisted in making the right organization of priorities and the efficient decisions.

The Best Bin Completion Packing Strategy results in a simple add on to the First Fit Decreasing or Best Fit Decreasing approaches, thanks to inspiration drawn from Richard Korf's Bin Completion algorithm.

5.2 Testing setups

More testing would benefit the present thesis with more corroborating information and results regarding not just time performance but also energy savings and water management efficiency. The problem handles lots of different variables simultaneously:

- Energy price slots
- Maximum and minimum water levels of each sector's crop
- Fertilization needs
- Water pump debit and fertilizer injection pump capacity

- Initial water level values of each sector
- Hourly evapotranspiration

For an exhaustive assessment of how the variation of each of these variables impacts the overall performance of the algorithm and efficiency of the solution irrigation plan, the present thesis would have to run tests where each of the variables is isolated and varied while all other variables remain constant, similar to what was done with the relation between water pump debit and the number of sectors in the system. This way, one could assert with precision how the algorithm behaves in handling various different scenarios:

- Highly irregular energy price distributions throughout the week
- Maximum and minimum water levels very close to each other or very set apart (although with the small amount of tests for a given number of sectors, such thing was roughly well analyzed)
- Extreme daily fertilization needs
- Extreme dryness in initial conditions
- Heavy rain or heavy heat, and irregular evapotranspiration distributions throughout the week

It would have been beneficial to use CPU efficient servers which could not only perform parallel testing of a variety of different setups but also run tests during days or even weeks. All presented tests, as previously stated, were performed on a domestic machine with an Intel Core i7 processor of 8th generation (with 12 cores), 12GB of RAM and an Ubuntu 18.04 operating system.

5.3 Linear Programming Model

It would have been interesting to assess a linear programming model of the whole problem, which was not accomplished. Once again, the use of CPU efficient servers running 24 hours a day could provide for this. The used Gurobi solver could be installed in such server and run a relaxed model of the problem. That would be helpful for a further analysis on the performance comparison between the LP algorithm and the Bin Packing one. It could corroborate the fact that linear programming models are either unpractical for real life use or are not able to encompass all the specifications and nuances of the irrigation problem.

References

[1] *Pistachio acreage in Spain up by almost 30% in the last year*. URL: https://www.freshplaza.com/article/9113691/pistachio-acreage-in-spain-up-by-almost-30-in-the-last-year/ (visited on 02/01/2020).

[2] *Informações sobre a árvore do Pistacho - WikiFarmer*. URL: https://wikifarmer.com/pt-br/informacoes-sobre-arvore-de-pistache/ (visited on 02/01/2020).

[3] Charlie Stoltenow and Greg Lardy. "Nitrate Poisoning of Livestock". In: (2015). URL: https://www.ag.ndsu.edu/publications/livestock/nitrate-poisoning-of-livestock (visited on 03/22/2020).

[4] Kenneth Hellevang. "Corn Drying and Storage Tips for 2011". In: (Sept. 2011). URL: https://www.ag.ndsu.edu/graindrying/documents/Corn_Drying_and_Storage_Tips_for_2011.pdf (visited on 03/22/2020).

[5] *Use of water in food and agriculture - Lenntech*. URL: https://www.lenntech.com/water-food-agriculture (visited on 04/15/2020).

[6] *Portuguesa com missão de poupar água ganha prémio de inovação europeu - Público*. URL: https://www.publico.pt/2019/10/15/tecnologia/noticia/portuguesa-missao-poupar-agua-ganha-premio-inovacao-europeu-1890157 (visited on 02/01/2020).

[7] Charles Moss, Grigorios Livanis, and Andrew Schmitz. "The Effect of Increased Energy Prices on Agriculture: A Differential Supply Approach". In: *Journal of Agricultural and Applied Economics* 42 (Nov. 2010). DOI: 10.1017/S1074070800003904.

[8] Toshichika Iizumi and Navin Ramankutty. "How do weather and climate influence cropping area and intensity?" In: *Global Food Security* 4 (2015), pp. 46–50. ISSN: 2211-9124. DOI: https://doi.org/10.1016/j.gfs.2014.11.003.

[9] *FAO Penman-Monteith equation*. URL: http://www.fao.org/3/X0490E/x0490e06.htm (visited on 05/22/2020).

[10] European Soil Data Centre. *Soil pH in Europe*. Tech. rep. Joint Research Centre, 2010.

[11] Randy Schnepf. *Energy Use in Agriculture: Background and Issues*. Tech. rep. Congressional Research Service, 2004.

[12] *How to Read a Pump Curve Series*. URL: http://jmpcoblog.com/how-to-read-a-pump-curve-series (visited on 03/08/2020).

[13] Anand Nayyar and Vikram Puri. "Smart farming: IoT based smart sensors agriculture stick for live temperature and moisture monitoring using Arduino, cloud computing solar technology". In: Nov. 2016, pp. 673–680. DOI: 10.1201/9781315364094-121.

[14] Joaquin Gutierrez et al. "Automated Irrigation System Using a Wireless Sensor Network and GPRS Module". In: *Instrumentation and Measurement, IEEE Transactions on* 63 (Jan. 2014), pp. 166–176. DOI: 10.1109/TIM.2013.2276487.

[15] Diego Ortiz, Alexander G. Litvin, and Maria G. Salas Fernandez. "A cost-effective and customizable automated irrigation system for precise high-throughput phenotyping in drought stress studies". In: *PLOS ONE* 13.6 (June 2018), pp. 1–16. DOI: 10.1371/journal.pone.0198546.

[16] *Introduction to Evapotranspiration - Food and Agriculture Organization of the United Nations*. URL: http://www.fao.org/3/x0490e/x0490e04.htm (visited on 03/09/2020).

[17] Garry Grabow et al. "Evaluation of Evapotranspiration-Based and Soil-Moisture-Based Irrigation Control in Turf". In: May 2008, pp. 1–9. ISBN: 978-0-7844-0976-3. DOI: 10.1061/40976(316)117.

[18] Aziz Benzekri, K. Meghriche, and Larbi Refoufi. "PC-Based Automation of a Multi-Mode Control for an Irrigation System". In: Aug. 2007, pp. 310–315. ISBN: 1-4244-0840-7. DOI: 10.1109/SIES.2007.4297350.

[19] Larry J. Sugarbaker and William J. Carswell. "The 3D elevation program - Precision agriculture and other farm practices". In: (2016). DOI: 10.3133/fs20163088.

[20] *What is NDVI (Normalized Difference Vegetation Index)? - GISGeography*. URL: https://gisgeography.com/ndvi-normalized-difference-vegetation-index/ (visited on 03/13/2020).

[21] U Siddique and Usman Rauf. "Automation of Irrigation System Using ANN based Controller". In: *International Journal of Electrical Computer Sciences IJECS-IJENS* 10 (Jan. 2010).

[22] Yossi Osroosh et al. "Comparison of irrigation automation algorithms for drip-irrigated apple trees". In: *Computers and Electronics in Agriculture* 128 (Oct. 2016), pp. 87–99. DOI: 10.1016/j.compag.2016.08.013.

[23] Richard Koech and Philip Langat. "Improving Irrigation Water Use Efficiency: A Review of Advances, Challenges and Opportunities in the Australian Context". In: *Water* 10 (Dec. 2018), p. 1771. DOI: 10.3390/w10121771.

[24] Bassam Bou-Fakhreddine et al. "Optimal multi-crop planning implemented under deficit irrigation". In: Apr. 2016, pp. 1–6. DOI: 10.1109/MELCON.2016.7495480.

[25] Zia Ul Haq. "Application of genetic algorithm for irrigation water scheduling". In: *Irrigation and Drainage* 53 (Dec. 2004), pp. 397–414. DOI: 10.1002/ird.121.

[26] Jordan Junkermeier. "A Genetic Algorithm for the Bin Packing Problem". In: 2015. URL: https://thejjjunk.ucoz.com/papers/ (visited on 05/25/2020).

[27] John R. Koza and Riccardo Poli. *Genetic Programming - Stanford University and University of Essex*. URL: https://icog-labs.com/wp-content/uploads/2014/07/Genetic-Programing-John-R.Koza_.-Stanford.pdf (visited on 02/04/2020).

[28] Pedro Larrañaga et al. "Learning Bayesian network structures by searching for the best ordering with genetic algorithms". In: vol. 26. 4. Institute of Electrical and Electronics Engineers, July 1996.

[29] Emanuel Falkenauer. "A Hybrid Grouping Genetic Algorithm for Bin Packing". In: vol. 2. 1. Kluwer Academic Publishers, June 1996.

[30] Emanuel Falkenauer and A. Delchambre. "A Genetic Algorithm for Bin Packing and Line Balancing". In: Institute of Electrical and Electronics Engineers, May 1992.

[31] Mohamed Amine Kaaouache and Sadok Bouamama. "Solving bin Packing Problem with a Hybrid Genetic Algorithm for VM Placement in Cloud". In: Procedia Computer Science, 2015.

[32] *Bin Packing Problem - Operations Research Group Bologna*. URL: http://www.or.deis.unibo.it/kp/Chapter8.pdf (visited on 02/05/2020).

[33] Silvano Martello and Paolo Toth. "Lower bounds and reduction procedures for the bin packing problem". In: *Discrete Applied Mathematics* 28 (1990), pp. 59–70.

[34] Richard E. Korf. "A New Algorithm for Optimal Bin Packing". In: Jan. 2002. URL: https://www.aaai.org/Papers/AAAI/2002/AAAI02-110.pdf (visited on 02/05/2020).

[35] Ethan L. Schreiber and Richard E. Korf. "Improved Bin Completion for Optimal Bin Packing and Number Partitioning". In: *IJCAI*. 2013.

[36] Catherine Lewis. "Linear Programming: Theory and Applications". In: (May 2008). URL: https://www.whitman.edu/Documents/Academics/Mathematics/lewis.pdf (visited on 02/05/2020).

[37] *Linear Programming - Brilliant*. URL: https://brilliant.org/wiki/linear-programming/ (visited on 02/05/2020).

[38] *PuLP - Linear Programming modeler in Python*. URL: https://pythonhosted.org/PuLP/main/installing_pulp_at_home.html (visited on 02/05/2020).

[39] LLC Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2019. URL: http://www.gurobi.com (visited on 02/05/2020).

[40] *PySCIPOpt*. URL: http://scip-interfaces.github.io/PySCIPOpt/docs/html/ (visited on 02/05/2020).

[41] Ambros Gleixner et al. *The SCIP Optimization Suite 6.0*. Technical Report. Optimization Online, July 2018. URL: http://www.optimization-online.org/DB_HTML/2018/07/6692.html (visited on 02/05/2020).

[42] *SCIP - Solving Constraint Integer Programs.* URL: https://scip.zib.de/index.php#about (visited on 02/05/2020).

[43] *Why C runs so much faster than Python - HuffPost.* URL: http://huffpost.com/entry/computer-programming-languages-why-c-runs-so-much_b_59af8178e4b0c50640cd632e (visited on 02/15/2020).

[44] *Optimize curve fit documentation - SciPy.org.* URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html (visited on 03/14/2020).

DECLARAÇÃO

FRANCISCO MORAIS DA COSTA MANSO, NIF 201 727 072, com domicilio profissional na Rua Dr. Joaquim Manso 12B 1500-241 LISBOA, na qualidade de representante legal de RIGGER S. A., NIF 514 260 718 e sede em Rua dos Três Lagares, 42, 6230-421 Fundão, tendo tomado inteiro e perfeito conhecimento da dissertação Smart Efficient Decision System, elaborada por Gonçalo Marques Raposo de Magalhães, aluno nº81817 do IST, declara que concorda com o conteúdo da dissertação e autoriza a divulgação da mesma com esse mesmo conteúdo que lhe foi apresentado.

Lisboa, 29 de Julho de 2020

Francisco Morais da Costa Manso