UNIVERSIDAD POLITÉCNICA DE MADRID

Proyecto de Fin de Grado
Grado en Ingeniería de Software

Parallel and Distributed Maximum Ant Colony Optimizer

Author: Pablo San José Villar

Supervisors:
Sandra Gómez Canaval - David Moreno Navas

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

Resumen

Este Proyecto de Fin de Grado propone una solución al problema de asignación de recursos generalizado (GAP) usando una meta-heurística bioinspirada en el contexto de la computación paralela y distribuida. En particular, la meta-heurística que se ha utilizado es Ant Colony Optimization (ACO) la cual se inspira en el comportamiento de las hormigas y cómo son capaces de optimizar el camino mínimo hasta la comida por medio del refuerzo de feromonas.

El problema concreto que trataremos de resolver es una variante del GAP aplicada a un caso de uso real como es la asignación de recursos a tareas en redes ferroviarias. La idea es optimizar al máximo, en primer lugar el número de tareas cubiertas y en segundo lugar, el número de recursos utilizados para cubrir esas tareas. La modelización del problema se ha hecho utilizando Programación por Restricciones siguiendo una definición previa implementada de forma secuencial, la cual fue introducida en el Proyecto de Fin de Máster "Maximum Ant Colony Optimizer" de David Moreno Navas, presentado en la E.T.S. de Ingenieros Informáticos de la Universidad Politécnica de Madrid.

La solución propuesta en este Proyecto está desarrollada para ser desplegada en la plataforma Apache Spark para conseguir la paralelización de las tareas del algoritmo ACO, mediante el uso del modelo *Map Reduce*. Los resultados obtenidos con la ejecución paralela del algoritmo propuesto aporta una mejora crucial en el tiempo de ejecución con respecto al mismo algoritmo en su versión secuencial. Además, los resultados permiten observar que la solución paralela no se ve afectada en el tiempo de ejecución cuando los datos o los parámetros de entrada crecen. Por tanto, se puede afirmar que la solución propuesta en este Proyecto se comporta de forma similar con el aumento de la complejidad del problema lo cual tiene correspondencia directa con la escalabilidad del sistema.

Este documento se estructura de la siguiente manera. En primer lugar, se introduce y estudia el problema del GAP y sus variantes. En particular, se trata de dar una visión comprensiva y analítica del reto que supone enfrentarse a este problema, así como de

las técnicas usadas para resolverlo. Adicionalmente, se introducen los conceptos relacionados con la Programación por Restricciones (CP) y las meta-heurísticas bioinspiradas. A continuación se hace un análisis breve de cómo estas han influido en la resolución de problemas similares y se analizan algunas propuestas de paralelización de este tipo de algoritmos usando estas técnicas. Posteriormente, se introduce la modelización del problema para el escenario real de la asignación de recursos a tareas en una red ferroviaria, así como también el algoritmo paralelo propuesto. A continuación se explica cómo se ha desplegado el algoritmo en la plataforma Apache Spark. Se introduce la batería de experimentos diseñada para probar el funcionamiento y ejecución del algoritmo, analizando los resultados obtenidos para finalmente, presentar las conclusiones obtenidas a partir de ellos así como tambiń introducir aspectos a ser considerados en un trabajo futuro.

Abstract

This Grade Final Project proposes a solution to the Generalized Assigment Problem (GAP) using a bioinspired meta-heuristic in the context of parallel and distributed computing. In particular, the meta-heuristic that has been used is Ant Colony Optimization (ACO) which is inspired by the behavior of the ants and how they are able to optimize the minimum path to the food by the reinforcement of pheromones.

The specific problem that we will try to solve is a variant of GAP applied to a real use case such as the assignment of resources to tasks in railway networks. The idea is to optimize firstly the number of tasks covered and secondly, the number of resources used to cover those tasks. The modeling of the problem has been done using Constraint Programming following the previous sequential definition, which was introduced in the David Moreno Navas “Maximum Ant Colony Optimizer” Master Project, presented in the E.T.S. Of Computer Engineering of the Polytechnic University of Madrid.

The solution proposed in this project is developed to be deployed in the Apache Spark platform to achieve the parallelization of the tasks of the ACO algorithm, using the Spark’s *Map Reduce* framework. The results obtained with the parallel execution of the proposed algorithm provide a crucial improvement in the execution time compare to the same algorithm in its sequential version. In addition, the results show that the parallel solution is not affected at runtime when data or input parameters grow. Therefore, it can be affirmed that the solution proposed in this Project behaves similar even with the increase of the complexity of the problem. This has direct correspondence with the scalability of the system.

This document is structured as follows. First, the problem of GAP and its variants is introduced and studied. In particular, we give a comprehensive and analytical vision about the challenge of dealing with this problem, as well as the techniques used to solve it. In addition, concepts related to Constraint Programming (CP) and bio-inspired metaheuristics are introduced.

The following is a brief analysis of how they have influenced the resolution of similar problems. Some proposals for parallelization of this type of algorithm are analyzed using these techniques. Subsequently, the modeling of the problem is introduced for the real scenario of the assignment of resources to tasks in a railway network, as well as the proposed parallel algorithm. Here's how the algorithm was deployed on the Apache Spark platform. We introduce the battery of experiments designed to test the operation and execution of the algorithm, analyzing the results obtained to finally present the conclusions drawn from them. Finally we introduce aspects to be considered in a future work.

Index

Index of Tables

Index of Figures

Chapter 1

Introduction

The new information era in which we live has given the privilege of having enormous amounts of data. But it is our responsibility to be able to manage this data in an useful and efficient way. One of the consequences of having such amounts of data is that the processing capability of individual computers is now insufficient. So, when we have a particular working solution for a concrete problem, it is necessary to come across new approaches to computing the solution for that problem in order to work with a massive data set.

There is a big tendency to develop algorithms in distributed platforms. The characteristics of them, make the parallel platforms very good not only to work with problems with massive data sets, but also to work with problems which solution set is incredibly big and/or complex to find. In other words, we can take advantage of the distributed solutions to work with hard problems, named, those problems requiring intensive computations in a big spaces. There is a very high interest in create new parallel and distributed versions on traditional algorithms which solves hard problems in order to improve the execution performance. This is one of the main justifications for propose a distributed solution.

In this project, we are going to develop a highly parallel and distributed solution for a problem related with the optimal assignment of resources to tasks. This problem can be stated as a branch of the Generalized Assignment Problem (GAP, in short). Our solution is going to be based in the one proposed by David Moreno in his Master

Final Project (MFP) which was developed using bio-inspired metaheuristics and Integer Programming [Moreno Navas, 2015].

1.1. Goals

As we said before, our goal is to propose a highly parallel and distributed solution to the resources assignment to tasks or more generally, GAP problem or similar. In David Moreno work [Moreno Navas, 2015], it is presented a very smart and complete sequential solution for the GAP problem. This solution includes constraint programming and bio-inspired meta-heuristics. We are going to take his solution as our base algorithm for the parallelization. First of all, we need to analyze the algorithm and take it to an ultra-scalable platform. We want to compare our distributed solution with the sequential one, at least in an informal way.

1.1.1. Specific Goals

We defined the following specific goals:

1. Analyze the sequential solution proposed in [Moreno Navas, 2015].
2. Identify which parts of this algorithm can be directly transformed into parallel ones.
3. Design the parallel algorithm.
4. Implement the proposed algorithm in an ultra-scalable architecture like Apache Spark.
5. Perform a comparative study between the results from both versions of the algorithm: the sequential version and our parallel solution, and discuss them.

1.2. Motivation and justification

In today's society there are a series of problems that need to be solved in an efficient way, with high quality results. The magnitude of these problems and the need for society to have fast and accurate solutions makes each day seek new alternatives for the dismantling and implementation of new solutions that meet these requirements.

Some of these problems are the allocation of resources to tasks, allocation of distribution routes, optimum cutting of materials in the industry, assignments of aircraft crews, among others. All these problems require solutions that are optimal results or close to the optimal method efficiently at reasonable times.

These problems are practical applications of classical problems of computer theory, specifically of computational complexity theory. Some of these problems are known and characterized as problems. The problem of the problem of satisfaction or the problem of generalized allocation (GAP). The theory of computational complexity shows that this type of problem is among the most difficult to solve, becoming intractable when the input data of the same begins to have a certain amount of elements.

The need for efficient (time and resource) computational solutions to these problems makes it continually to look for new algorithms and implementations that can improve the results of the implementations currently used, the salaries are usually designed to be deployed in traditional computing environments .

Nowadays, the emergence of parallel and distributed computing hardware and software platforms, among which are found in the management guidelines Large data, open a new line of interest for the development of tools ultra capable of executing complex algorithms. In particular, these platforms are able to efficiently handle massive computations in data with a great facility to scale physical resources and parallelize computations.

It is essential to take advantage of the computing power offered by new computing architectures in the offers and that are allowing to offer solutions to the areas of Computer Science that until now were not technologically possible (such as Big Data, Machine Learning, Scientific Computing , Bio-Informatics, Operations Research, etc.).

Since one of the disadvantages of finding solutions to these problems in the generation and evaluation of the search space for candidate solutions, there is the possibility of accelerating or generating this space in a reasonable way using these new platforms. It is possible to destabilize the real computing capacity that allows flexibility to deal with the spaces of search for huge solutions. This is due to the fact that it can scale the processing in several units of a concrete computational architecture so as to allow the size of the problem a solver falls into a certain range of the one that controls that can be solved in a reasonable time.

In this context, this End-of-Grade Project proposes a highly parallel and distributed solution to the resources assignment to tasks problem or more generally, GAP problem or similar, using an bio-inspired meta-heuristic and a Integer Programming approach following the sequential solution proposed in [Moreno Navas, 2015] . Up to our knowledge, there aren't many works able to attack these problems with these approaches using new highly parallel and distributed platforms.

Chapter 2

Background

In this chapter we would like to introduce the main concepts needed to understand the problem we try to solve.

2.1. Computational and Algorithmic Complexity

The algorithmic complexity is the efficiency measure for the algorithms. Concretely, the complexity in time give us the magnitude order of the number of elementary instructions executed to finish the algorithm. The complexity of a problem is defined by the complexity of the algorithms needed to solve it [Arora and Barak, 2007]. To represent the algorithmic complexity we use the O notation. Let $O(f(n))$ the temporal complexity, where n represents the input size of the algorithm. This function is an upper bound which allow us to understand the time growth from the input growth. To classify the complexity of a decision problem (A problem with a yes or no answer) there are several complexity classes.

P is a complexity class that represents the set of all decision problems that can be solved in polynomial time. That is, given an instance of the problem, the answer yes or no can be decided in polynomial time. This is $O(nk)$ where n is the input size and k a constant independent of the input.

NP is a complexity class that represents the set of all decision problems for which the instances where the answer is “yes” have proofs that can be verified in polynomial time. This means that if someone gives us an instance of the problem and a certificate (sometimes called a witness) to the answer being yes, we can check that it is correct in polynomial time. Also the NP class is defined by the set of decision problems solvable in polynomial time by a theoretical non-deterministic Turing machine. This is the basis for the abbreviation NP, which stands for "non-deterministic polynomial time".

NP-Complete is a complexity class which represents the set of all problems X in NP for which it is possible to reduce any other NP problem Y to X in polynomial time. Intuitively this means that we can solve Y quickly if we know how to solve X quickly. Precisely, Y is reducible to X, if there is a polynomial time algorithm f to transform instances y of Y to instances $x = f(y)$ of X in polynomial time, with the property that the answer to y is yes, if and only if the answer to $f(y)$ is yes.

NP-Hard are the problems that are at least as hard as the *NP-Complete* problems. Note that*NP-Hard* problems do not have to be in NP, and they do not have to be decision problems. The precise definition here is that a problem X is $NP-hard$, if there is an *NP-Complete* problem Y, such that Y is reducible to X in polynomial time. But since any *NP-Complete* problem can be reduced to any other *NP-Complete* problem in polynomial time, all *NP-Complete* problems can be reduced to any *NP-Hard* problem in polynomial time. Then, if there is a solution to one *NP-Hard* problem in polynomial time, there is a solution to all NP problems in polynomial time.

The P and NP classes have a relationship. Concretely $P \subseteq NP$. Every decision problem which can be solve by a polynomial deterministic algorithm can be solve by a polynomial no deterministic algorithm too. So if a decision problem is P class it is also NP. This one is the most famous problem in computer science, and one of the most important outstanding questions in the mathematical sciences. In fact, the Clay Institute is offering one million dollars for a solution to the problem [Institute, 2000]. Creo que esta referencia no esta bien puesta

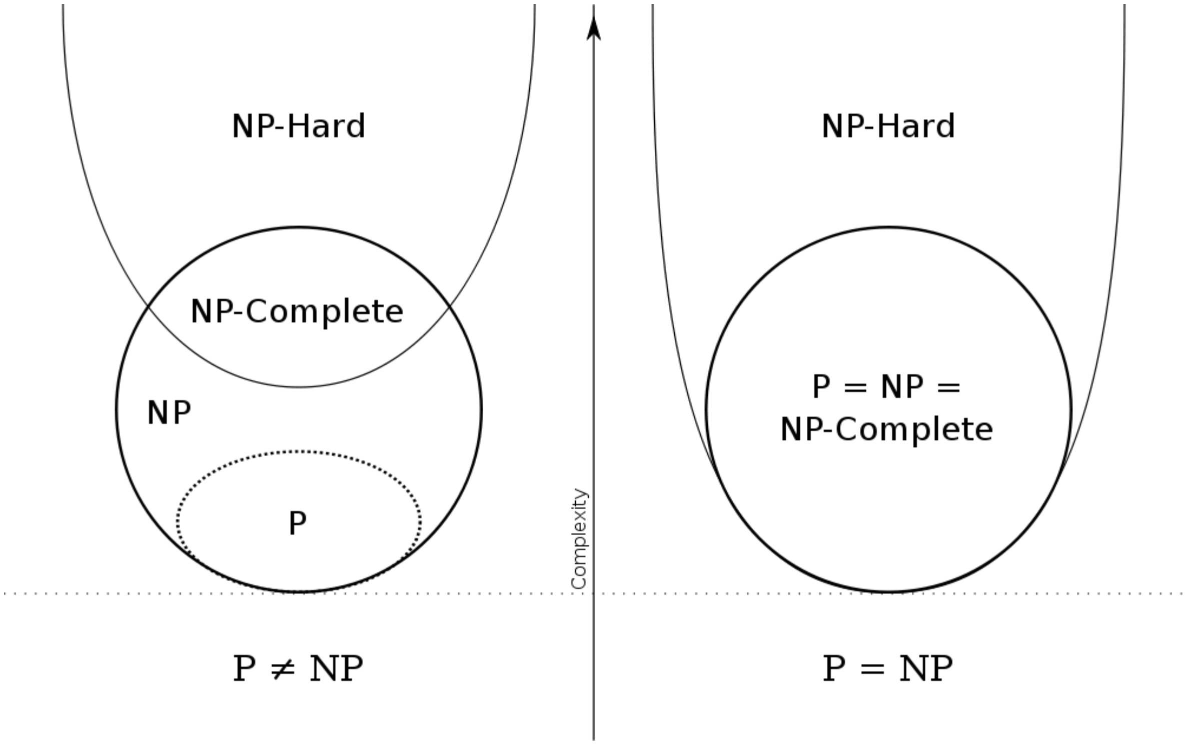

Figure 2.1: Visual representation of P and NP sets

For more details about this section we refer to the reader to [Pérez and Caparrini, 2003].

2.2. Constraint and Integer Programming Concepts

Integer programming (IP in short) is a thriving area of optimization, which is applied nowadays to a multitude of human endeavors, thanks to high quality software. It was developed over several decades and is still evolving rapidly. It is a programming paradigm where the relationships between the variables are in restriction forms. These restrictions represent the properties of the solutions to find. These solutions have to be as optimal or close to optimal as possible.

To better understand the excitement that is generated today by this area of mathematics, it is helpful to provide a historical perspective. Babylonian tablets show that mathematicians were already solving systems of linear equations over 3,000 years ago. The eighth book of the Chinese Nine Books of Arithmetic, written over 2,000 years ago, describes what is now known as the Gaussian elimination method. In 1809, Gauss used this method in his work, stating that it was a "standard technique". The method was subsequently named after him [Conforti et al., 2014].

A major breakthrough occurred when mathematicians started analyzing systems of linear inequalities. This is a fertile ground for beautiful theories. In 1826 Fourier gave an algorithm for solving such systems by eliminating variables one at a time. Systems of linear inequalities define polyhedra and it is natural to optimize a linear function over them. This is the topic of linear programming, arguably one of the greatest successes of computational mathematics in the twentieth century. The simplex method, developed by Dantzig in 1951, is currently used to solve large-scale problems in all sorts of application areas. It is often desirable to find integer solutions to linear programs.

When considering algorithmic questions, a fundamental issue is the increase in computing time when the size of the problem instance increases. Edmonds was one of the pioneers in stressing the importance of polynomial-time algorithms. The existence of a polynomial-time algorithm for linear programming remained a challenge for many years. This question was resolved positively by Khachiyan in 1979, and later by Karmarkar using a totally different algorithm. Both algorithms were (and still are) very influential, each in its own way. In integer programming, Lenstra found a polynomial-time algorithm when the number of variables is fixed. So, although integer programming is NP-hard in general, the polyhedral approach has proven successful in practice.

Integer and constraint programming are paradigms broadly used into Operations Research area. Next, we introduce briefly, some concepts about Operation research (OR).

Operation Research

The operations research (in short OR) is a scientific approximation to problem analysis and decision making. The main goal is achieve optimal solutions or, at least, very close to the optimum. The methods used by the Operations Research are varied and interdisciplinary. Some of them are simulation, optimization, queue theory, Markov chain, economic methods, data analysis, statistics, neural networks, experts systems, and decision analysis [Moreno Navas, 2015].

Stages in a operation research study

The operation research is a very complex math field. It of course has a scientific part based in all the techniques mentioned above, but it also has some kind of artistic element based in the experience and creativity of the researches. Because of that, it is difficult to set a guide for an OR study.

The following steps can be a systematic approximation to address OR problems:

1. **Problem definition**: The objective is to define the problem's decision elements. This is: the description of the decision alternatives, the definition of the goals within the study and the specification of the limitations for the system.

2. **Model Construction**: Translate the problem definition into mathematical terms. So, if the model fits any of the standard mathematical models, it will be used. Otherwise, if the problem is too complex, heuristics and simulations will be used. Sometimes a combination of both will be the best option.

3. **Model Solution**: This is the simplest part because it uses optimization algorithms very well defined. One important aspect is the sensitivity analysis when the model parameters can't be accurately estimated.

4. **Model Validity**: Check if the model achieves the expected behavior. The model is valid if for similar starting conditions, it behaves similarly.

5. **Implementation of the solution in a validated model**: Translate the results into instructions for the system users.

Some problems in Operations Research

There are many and different problems in the Operation Research (OR). From communication network optimization, to finding optimal prices to many fields and many applications. We would like to enumerate briefly some problems closely related to the one which we will discuss in this project: assignments problems, planning for routes and similar ones.

Assignment Problem (AP): This problem tries to assign in an optimal way n people to n tasks, assuming that each person has a numerical value for the performances of each task. An optimal assignment is that maximize the sum of each values for the people assigned to each task. There are $n!$ possible solutions and there can be many optimal solutions. Find each optimal solution is almost impossible unless n is really small. The main goal for this problem is to find an reasonably efficient algorithm which can find just one optimal solution or close to it.

Generalized assignment Problem (GAP): This problem, along with AP problem, constitute the main problem we trying to solve in our project. We will explain them in the next sections. For we briefly introduction, this problem is an extension of the classic assignment problem. There are as well n agents and n tasks, and each agent can be assign to every task having a tuple of cost and benefits that will depend of the agent-task assignment. So the goal is to find one assignment which maximize the benefit.

Nurse assignment Problem (NSP): The problem consist in generate schemes for the nurse coverage. In each schema it is specified the number and identity of every nurse which work each day in the planning period. Because we have the nurses' identity, we generate the scheme in an individual level. The problem is to generate that nurse pattern to cover the hospital requirements and the nurse preference.

Air crew assignment Problem (ACAP): This problem tries to generate rotations between different crews by a defined scheme, trying to minimize costs. The problem has to be solve considering the different and complex agreements, the pairings costs, and having to return the crew to their bases.

Some methods for solving OR problems

There are a lot of mathematical methods and tools for solving these Operations Research problems. The **Simplex method** is one of the most famous. It works it linear programs in standard form. This method tries to maximize an objective function subject to a series of restrictions. When we got a large linear program, another popular method is the **Column Generator**. The overarching idea is that if your linear programs are too large then you won't consider all the variables explicitly. Since most of the variables will be non-basic and assume a value of zero in the optimal solution, only a subset of variables need to be considered in theory when solving the problem. Column generation leverages this idea to generate only the variables which have the potential to improve the objective function.

2.3. Generalized Assignment Problem - GAP

The Generalized Assignment Problem (GAP) can be described, using the terminology of knapsack problems [Martello and Toth, 1990], as follows. Given n items and m knapsacks, with:

$$p_{ij} = \text{profit of item } j \text{ if assigned to knapsack } i \tag{2.1}$$

$$w_{ij} = \text{weight of item } j \text{ if assigned to knapsack } i \tag{2.2}$$

$$c_i = \text{capacity of knapsack } i \tag{2.3}$$

assign each item to exactly one knapsack so as to maximize the total profit assigned, without assigning to any knapsack a total weight greater than its capacity, i.e

$$\text{maximize } z = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} \tag{2.4}$$

$$\text{subject to } \sum_{j=1}^{n} w_{ij} x_{ij} \leq c_i, i \in M = \{1, ..., m\}, \tag{2.5}$$

$$\sum_{i=1}^{m} x_{ij}, j \in N = \{1, ..., n\}, \tag{2.6}$$

$$x_{ij} = 0 \text{ or } 1, i \in M, j \in N, \tag{2.7}$$

The restriction 2.6 is formally defined as

$$f(x) = \begin{cases} 1, & \text{if the item } j \text{ is assigned to knapsack } i \\ 0, & \text{otherwise} \end{cases} \tag{2.8}$$

This problem is frequently described in the literature as the optimal assignment of n tasks to m processors (n jobs to m agents, and so on), given the profit p_{ij} and the amount of resource w_{ij} corresponding to the assignment of task j to processor i, and the total resource c_i, available for each processor i.

The minimization version of the problem can also be stated by defining c_{ij} as the *cost* required to assign item j to knapsack i, such that MINGAP is

$$\text{minimize } v = \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.9}$$

$$\text{subject to } \sum_{j=1}^{n} w_{ij} x_{ij} \leq c_i, i \in M = \{1, ..., m\}, \tag{2.10}$$

$$\sum_{i=1}^{m} x_{ij}, j \in N = \{1, ..., n\}, \tag{2.11}$$

$$x_{ij} = 0 \text{ or } 1, i \in M, j \in N, \tag{2.12}$$

GAP and MINGAP are equivalent. Setting $p_{ij} = -c_{ij}$ (or $c_{ij} = -p_{ij}$) for all $i \in M$ and $j \in N$ immediately transforms one version into the other. If the numerical data are restricted to positive integers (as frequently occurs), the transformation can be obtained as follows. Given an instance of MINGAP, define any integer value t such that

$$t > max_{i \in M,\ j \in N}\{c_{ij}\} \tag{2.13}$$

And set

$$p_{ij} = t - c_{ij} \text{ for } i \in M, j \in N \tag{2.14}$$

From 2.9 then we have

$$v = t \sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij} - \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} x_{ij} \tag{2.15}$$

From such 2.6, the first term is independent of x_{ij}. Hence the solution (x_{ij}) of GAP also solves MINGAP. The same method transforms any instance of GAP into an equivalent instance of MINGAP (by setting $c_{ij} = t - p_{ij}$ for $i \in M, j \in N$ with $t > max_{i \in M,\ j \in N}\{p_{ij}\}$).

Due to constraints, an instance of the generalized assignment problem does not necessarily have feasible solutions. Moreover, even the feasibility question is NP-*complete*.

2.4. Meta-heuristics for GAP problems

Heuristic methods have proven to be a comprehensive tool to solve hard optimization problems. They bring a balance of “good” solutions (relatively close to global optimum) and affordable time and cost [Parejo et al., 2012]. However, heuristics are usually based on specific characteristics of the problem at hand, which makes their design and development a complex task. In order to solve this drawback, meta-heuristics appear as a significant advance [Glover, 1977]. They are problem-agnostic algorithms that can be adapted to incorporate the problem-specific knowledge. Meta-heuristics have been remarkably developed in recent decades [Voß, 2001], becoming popular and being applied to many problems in diverse areas.

As a solution, object-oriented paradigm has become a successful mechanism used to ease the burden of application development and particularly, on adapting a given meta-heuristic to the specific problem to solve. Based on this paradigm, there are a number of proposals which jointly offer support for the most widespread techniques, platforms and languages.

In practice, combinatorial optimization problems are often NP-hard, CPU time-consuming, and evolve over time. Unlike exact methods, meta-heuristics allow to tackle large-size problems instances by delivering satisfactory solutions in a reasonable time. Meta-heuristics are general-purpose heuristics that split in two categories: evolutionary algorithms (EA) and local search methods (LS). Although serial meta-heuristics have a polynomial temporal complexity, they remain unsatisfactory for industrial problems. Parallel and distributed computing is a powerful way to deal with the performance issue of these problems.

Ant Colony Optimization

We would like to introduce briefly the Ant Colony Optimization algorithm (ACO), as it is going to be one of the main issues of this project. This metaheuristic is inspired in the ant behavior and their capability to find quickly the shortest paths to a certain point. The ants leave pheromones in the path they have walked. The best paths are the ones with more pheromones. There is a random factor that let try out new paths.

A more detailed explanation of this technique can be found in this chapter 3.2.1.

Chapter 3

State of the art

The continuous increase in the volume and detail of data captured by organizations in the last years, such as the rise of social media, Internet of Things (IoT), and multimedia, has produced an overwhelming flow of data in either structured or unstructured format. Data creation is occurring at a record rate, referred to herein as big data, and has emerged as a widely recognized trend [Hashem et al., 2015].

This high volume of data is known as Big Data. Initially, Big Data was defined using the 3Vs model (Volume, Velocity, and Variety) n [Laney, 2001]. After that, other studies pointed out that this definition was insufficient. Then, other characteristics such as veracity, and value wad added to the 3Vs definition [Apache software, 2009].

Nowadays, there are many computing platforms for Big Data scenarios. A possible classification can be given for the type of scalability that they can offer. The scalability is the capacity of the a computing system to adapt it to continuous and increasing demand related with the data processing [Singh and Reddy, 2014].

The types od scalability are:

1. **Scale out**: (horizontal scalability): This type of scalability refers to expand computational resources such that servers, physical devices or computers (usually within a cluster of computers) in order to improve the processing capacity. This scalability implies the processing load distribution between the machines such that each one has its own operative system.

2. **Scale up** (vertical scalability): This type of scalability refers to expand compu-

tational resources such that processors memory and/or other advanced hardware units within one machine or server. This scalability implies only one instance of an operative system.

Computing platforms offering scale out are Giraph, Power Graph, Spark and so on. By the other hand, computing platforms offering scale up are frequently hardware devices such Graphical Processing Units (GPUs).

In this chapter we introduce the current context about the OR methods and Meta-Heuristic approaches within Big Date age but firstly, we introduce in a detailed way the applications for the GAP problem in OR.

3.1. GAP Applications for Operations Research

Integer-programming models arise in practically every area of application of mathematical programming. To develop a preliminary appreciation for the importance of these models, we introduce, in this section, three areas where integer programming has played an important role in supporting managerial decisions.

3.2. Algorithms and Bioinspired Meta-heuristics solutions for GAP

In the following section we explain the concepts for ACO and the main implementations for a single objective and multi-objective search as well as it applications. The concepts related with ACO and CP with some implementations will be introduced too.

3.2.1. Ant Colony Optimization

The use of ant based algorithms was proposed to solve hard combinatorial problems [Dorigo et al., 1996]. The ants are eusocial animals with autorganizational capabilities. There is not just one ant which controls the rest of the colony. One of the most interesting parts of their behavior is the stigmergy, a communication form based in pheromones. With this mechanism, the ants achieve the best paths to the food. If there is two possible routes to the food, one longer than the other, the ant which has gone by the shortest one returns before the other ant. The rest of the ants now are prone to go by that path and reinforce it with their own pheromones.

The ant based algorithms take advantage of the stigmergy communication as an indirect form of communication which modifies the environment and can only be interpreted by the communication agents. This feature is easily adapted to artificial agents in order to solve problems with different states and state variables giving access to these variables to the ants.

Two aspects used by the artificial ants are the autocatalysis (positive feedback) and the implicit solution evaluation, which are defined as follows:

1. Autocatalysis quickly reinforces the best solutions prematurely and this can be a problem because the solutions can be within local minimal. In order to avoid it, it is used the evaporation of the pheromones and the stochastic choice of states.

2. The implicit evaluation is due to their ability to quickly find the shortest pathways and are therefore reinforced by faster pheromones.

Other features from the artificial ants are:

1. Due to the discrete nature of the problems that wish to be solved with this meta-heuristic the world of the artificial ants is discreet.

2. Ants can keep an internal state that represents past actions.

3. The way in which the pheromones are deposited do not have to be equal to the real because of the type of problem that is being solved. This deposition can be done once a solution has been found.

4. It can be added characteristics of other meta-heuristics, such as local search and as in the case we want to expose, use constraint programming to always ensure correct states.

On the other hand, the features from the natural ants are:

1. Is a cooperative individuals colony.

2. Pheromones paths and stigmergy.

3. The search for the shortest path and the use of local movements for that.

4. Stochastic state transitions.

Specifically, Ant Colony Optimization (ACO) takes features of the natural ants and other features of the artificial ants.

ACO Meta-heuristic Characterization

We can see in the algorithms 1, 2 and 3 a ACO Meta-heuristic specification as it is proposed by [Dorigo et al., 1996].

Algorithm 1 ACO-Meta-heuristic

```
while finalizationCriteriaNotAccomplish do
    GenerateActivitiesAndAnts();
    pheromoneEvaporation();
    daemonActions(); //optional
end while
```

Algorithm 2 GenerateActivitiesAndAnts()

```
while enoughResources do
    planNewAntCreation();
    NewActiveAnt();
end while
```

Algorithm 3 NewActiveAnt() (Ant Lifecycle)

```
initializeAnt();
$M$ = updateAntMemory();
while currentState ! = objectiveState do
    $A$ = getAntRoutingTable();
    $P$ = computeTransitionProbabilities(A, M, problemsRestrictions);
    nextState =applyAntDecisionPolicy(P, problemsRestrictions);
    moveNextState(nextState);
    if pheromoneUpdateStepByStepOnline then
        putPheromoneInVisitedEdge();
        updateAntRoutingTable();
    end if
    $M$ = updateInternalState();
end while
if pheromoneUpdateDelayedOnline then
    evaluateSolutionn();
    putPheromoneInAllVisitedEdges();
    updateAntRoutingTable();
end if
die();
```

ACO will be executed while the desire objective is not reached. This objective will depend on the problem to solve or till a certain number of iterations is reached. In each iteration all the ant's activities that the problem requires will be created. Each ant saves an internal state to take the decisions. In each step, while the objective state is not reached, an algorithm takes the decision of the next state, considering the restrictions of the problem and the probability of the transactions. In the next state the pheromone of the edge between the previous and the current state will be updated as the algorithm requires it. Also the ant's routing table will be updated. When the objective state is reached, if the algorithm specifies it, the solution evaluation will be made and with the help of it, all the pheromones of the visited edges will be updated.

Then the ant's routing table will be updated.

After using all the ants, the next step is the pheromone evaporation form all the edges in the problem. So every less visited path will be weakened. We can use a daemon which allows us to have a global vision of the results, because the ants just give us a local knowledge. One option is deposit more pheromones in the best paths found. In that situation we also help to the knowledge to the system and we can applied the best heuristics for the problem.

All this steps are adapted to the problem to solve, using or suppressing some of them.

ACO Algorithms

Ant System (AS)

AS was the first ACO algorithm [Dorigo et al., 1996]. This algorithm was implemented to solve TSP (Traveling Salesman Problem) .

In AS, the artificial ants generate paths, moving by the nodes of the graph. Through a probabilistic transition rule, the ants go from one node to another. The chosen artists are added to the path. Each ant has limited number of steps to be taken by iteration. The number of iterations is limited.

The evaporation takes place before the ants begin to deposit pheromones. There are no daemon actions.

The pheromone trail $\tau_{ij}(t)$ associated with an arc $arc(i,\, j)$ represents how desirable is to pass to the node j being in the i node.This pheromone trail is changing as is learning about the problem that is being solved. The evaporation of the pheromones avoids stagnation in finding solutions.

When TSP is solved, the ants have a memory where they store nodes that have already traveled to be able to traverse the remaining ones.

The ants decision table: $\mathcal{A}_i = [a_{ij}(t)]_{\text{—}\mathcal{N}_i\text{—}}$ in the node i is obtained by composing the value of the pheromone trace with the local heuristic as follows:

$$a_{ij} = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in \mathcal{N}_i}[\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} \forall\, j \in \mathcal{N}_i \tag{3.1}$$

such that:

- $\tau_{ij}(t)$ is the amount of pheromone in the edge $arc(i,\, j)$ at the time t;
- $\eta_{ij} = \frac{1}{d_{ij}}$ is the heuristic value to move from the node i to the node j;
- $\mathcal{N}_i$ are the set of neighbours of the node i;
- α and β are parameters that control the relative weight of the pheromone trace and the heuristic.

The probability that an ant k choses to go from the node i to the node $j \in \mathcal{N}_i^k$ while its path is been building in the iteration t is:

$$p_{ij}^k(t) = \frac{a_{ij(t)}}{\sum_{l \in \mathcal{N}_i^k} a_{il}(t)} \tag{3.2}$$

Such that $\mathcal{N}_i^k \subseteq \mathcal{N}_i$ is the set of all neighbours to the node i which the ant k hasn't visited yet.

After all the ants have completed their course, the pheromones evaporate and, afterwards, each ant k deposits an amount of pheromone $\Delta\tau_{ij}^k(t)$ as follows:

$$\Delta\tau_{ij}^k(t) = \begin{cases} 1/L^k(t) & \text{if } (i,j) \in T^k(t) \\ 0 & \text{if } (i,j) \notin T^k(t) \end{cases} \tag{3.3}$$

such that $T^k(t)$ is the path followed by the ant in the iteration t and $L^k(t)$ is its length.

The way in which the deposit of the pheromones is made through evaporation is:

$$\tau_{ij}(t) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{3.4}$$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \tag{3.5}$$

such that m is the number of the ants and $\rho \in (0, 1]$ is the evaporating coefficient of the pheromones.

At the time when this algorithm was developed, the best known solutions could not be reached, due to the limitations of the machines at that time. However, the rapid convergence towards good solutions is achieved.

MaxMin AS ($\mathcal{MM}$AS)

MaxMin AS ($\mathcal{MM}$AS) [Stützle and Hoos, 2000] is based on AS but has the following differences:

- The pheromone trail is updated by the daemon, giving an extra pheromone to the edges of the best solution.
- The value of the pheromone trail is limited to an interval and the trail is initialized to the maximum allowable value.
- The updating of the pheromones is done proportionally with what the difference between traces is diminishing and in this way it favors the exploration of new ways.

Ant Colony System (ACS)

ACS is based on AS but has the following differences:

- At the end of an iteration, the daemon adds pheromones to the edges of the best path found. The rule for updating pheromones is the following, with $\rho \in (0, 1]$ y L^+ the best path length T^+ from the beginning of the iteration:

$$\tau_{ij}(t) = (1 - \rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t) \tag{3.6}$$

$$\Delta\tau_{ij}(t) = 1/L^+(t) \tag{3.7}$$

- Ants use a different decision rule, known as the *pseudo-random-proportional* rule. The decision table remains the same as in AS. The new rule works as follows. Having a random q, uniformly distributed between 0 and, q_0 between 0 and 1, being a configurable parameter.

If $q \leq q_0$ then

$$p_{ij}^k(t) = \begin{cases} 1 & \text{if } j = arg\,max\,a_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

If $q > q_0$

$$p_{ij}^k(t) = \frac{a_{ij(t)}}{\sum_{l \in \mathcal{N}_i^k} a_{il}(t)} \tag{3.9}$$

- The ants are updating in each step the pheromones with the objective of avoiding the stagnation and that can arise new ways. The pheromone update rule, with $\varphi \in (0, 1]$ and τ_0, where τ_0 is the initial value of the pheromone trace, is as follows:

$$\tau_{ij}(t) = (1 - \varphi)\tau_{ij}(t) + \varphi\tau_0 \tag{3.10}$$

- It uses a list of candidates with more heuristic information about which nodes are preferred to those that go from another node.

3.2.2. CP y ACO

The idea of putting together ACO and Constrait Programming tries to generalize the different ACO algorithms for different problems [Moreno Navas, 2015]. CP allows to create a model with all the restrictions for each problem. This model could be solve by general purpose or specific heuristics for the problem. The classic procedure for ACO can be used as an heuristic for the CSP to solve, being a general purpose part in the problem. ACO can introduces the specific aspects for each problem with its heuristic information matrix, and the configurable parameters such as the number of iterations, the population size, values of α β y p. It is possible to generalize it so multicriteria function could be accepted. However, aspects like this need a more in deep investigation.

From the ACO perspective, using CP gives the advantage that every constructed solutions are feasible. We can see algorithm develop introduced in [Moreno Navas, 2015] which combines CP and ACO in the algorithms 10. The algorithm is divided in two core parts for the CP. First, the algorithm decide in which order are going to be assigned the variables calling the method in 5. Then, the assignation for each value are made using the method in 7. The method completes an assignation for a variable and then moves to the next one.

Algorithm 4 antRailroadAssignment()

```
if solveRestrictionProblem() then
    saveAntInfo();
end if
```

Algorithm 5 solveRestrictionProblem()

```
solve(variablesSelector, valuesSelector);
```

Algorithm 6 variablesSelector() RandomACO Version

```
variable = emptyVariable;
numberOfTries = 0;
while numberOfResourcesAssigned < maximunResorcesAssignables and !allVariablesFixed and numberOfTries < totalResources do
    random = getPseudoramdomValue(0, totalResources);
    resource = getResource(resourceArray[random]);
    numberOfTries = numberOfTries + 1;
    if stepNumberCurrentResource < maximunSteps(resource) and !variableIsInstancied(resource, stepNumberCurrentResource) then
        stepNumberCurrentResource = stepNumberCurrentResource + 1;
        variable = getAssignationToVariable(resource, stepNumberCurrentResource);
    else
        if assignedToTask(resource) then
            numberOfResourceAssignedToTask + 1;
        end if
        stepNumberCurrentResource = 0;
        variable = variablesSelector();
    end if
end while
return variable;
```

Algorithm 7 valuesSelector()

```
selectedTask = emptyTask;
random = getPseudoramdomValue(range(0, 1));
if random $\leq$ scanThreshold then
    selectedTask = selectRandom();
else
    selectedTask = selectMoreProbableTask();
end if
return selectedTask;
```

Algorithm 8 selectRandom()

```
denominator = 0;
random = getPseudoramdomValue(range(0, 1));
task = emptyTask;
while valuesInDomain == True do
    denominator = denominator + computeProbability(currentTask);
end while
while valuesInDomain == True and pheromone ¡= random do
    if currentTask != emptyTask then
        pheromone = computeProbability(currentTask)/denominator * scanThreshold;
        if random ¡pheromone then
            task = emptyTask;
        end if
    end if
end while
return task;
```

Algorithm 9 selectMoreProbableTask()

```
task = emptyTask;
probability = 0.0;
while valuesInDomain do
    if currentTask != emptyTask then
        evaluation = computeProbability(currentTask);
        if evaluation ¿probability then
            task = currentTask;
        end if
    end if
end while
return task;
```

Algorithm 10 computeProbability()

```
distance = pow(getDistanceMatrixValue(previousTask, currentTask),$\beta$);
pheromone = pow(getPheromoneMatrixValue(idPrev, value),$\alpha$);
return distance * pheromone;
```

Algorithm 11 computeDistanceMatrix()

```
distanceMatrix;
row = 0;
while therIsTaks do
    column = 0;
    while therIsTaks do
        distanceMatrix[row][column] = computeDistance(row, column);
    end while
end while
```

Algorithm 12 computeDistance(currentTask, nextTask)

```
value = initialPheromone ;
if firstStep and !emptyTak then
    if isPassengerTask(currentTaks) then
        value = initialPheromone / numerOfTaks;
    else
        value = initialPheromone / sqrt(numerOfTaks);
    end if
else
    if noAccesibleNextTaks or emptyTak == currentTask or emptyTak == nextTask
    then
        value = 0.0;
    end if
else
    if isPassengerTask(currentTask) or isPassengerTask(nextTask) then
        value = computePheromonePassengerTaks(currentTask, nextTask);
    end if
else
    value = computeMaximumPheromone(currentTask, nextTask);
end if
```

Algorithm 13 computePheromonePassengerTaks(currentTask, nextTask)

```
value = initialPheromone;
if isPassengerTask(nextTask) then
    if isPassengerTask(currentTask) and sameTrain(currentTask, nextTask) then
        value = initialPheromone / numerOfTaks;
    else
        value = initialPheromone / (2 * numerOfTaks);
    end if
else
    computeMaximumPheromone(currentTask, nextTask);
end if
return value
```

Algorithm 14 computeMaximumPheromone(currentTask, nextTask)

```
value = initialPheromone;
if thereIsNextStep then
    value = initialPheromone;
else
    value = initialPheromone/$sqrt$((initialTime(nextTask) - finalTime(currentTask));
end if
return value
```

3.3. Computing strategies and approaches for Operations research

There are not so many developments in distributed meta-heuristics for operation research. In this section we are going to analyze a couple of interesting solutions in this field.

3.3.1. Parallel and distributed meta-heuristics

We want to provide some solutions in parallel and distributed meta-heuristics. These meta-heuristics are not focused in solving the GAP, but we can obtain a clear view of what we can expect of a distributed meta-heuristic [Cahon et al., 2004].

Parallel distributed evolutionary algorithms

Evolutionary Algorithms [Goldberg and Holland, 1988] (EA) are based on the iterative improvement of a population of solutions. At each step, individuals are selected, paired and recombined in order to generate new solutions that replace other ones, and so on. As the algorithm converges, the population is mainly composed of individuals well adapted to the environment, for instance the problem. The main features that characterize EA are the way the population is initialized, the selection strategy (deterministic/stochastic) by fostering good solutions, the replacement strategy that discards individuals, and the continuation/stopping criterion to decide whether the evolution should go on or not. Basically, three major parallel and distributed models for EA can been distinguished: the island (a)synchronous cooperative model, the parallel evaluation of the population, and the distributed evaluation of a single solution.

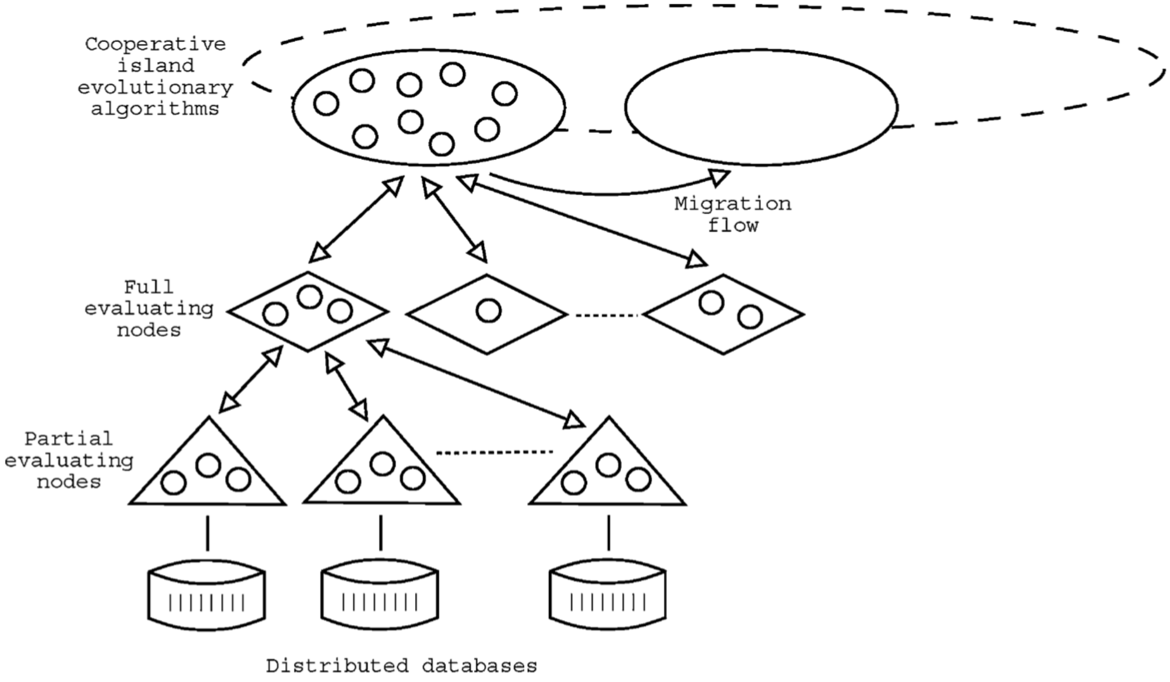

Figure 3.1: Three major parallel distributed models for EA [Cahon et al., 2004]

1. ***Island (a)synchronous cooperative model:*** Different EA are simultaneously deployed to cooperate for computing better and robust solutions. They exchange in an asynchronous way genetic stuff to diversify the search. The objective is to allow to delay the global convergence, especially when the EA are heterogeneous regarding the variation operators. The migration of individuals follows a policy defined by few parameters: the migration decision criterion, the exchange topology, the number of emigrants, the emigrants selection policy, and the replacement/integration policy.

2. ***Parallel evaluation of the population:*** It is required as it is in general the most time- consuming. The parallel evaluation follows the centralized model. The farmer applies the following operations: selection, transformation and replacement as they require a global management of the population. At each generation, it distributes the set of new solutions between different workers. These evaluate and return back the solutions and their quality values. An efficient execution is often

obtained particularly when the evaluation of each solution is costly. The two main advantages of an asynchronous model over the synchronous model are:

a) The fault tolerance of the asynchronous model

b) The robustness in case the fitness computation can take very different computation times (e.g. for nonlinear numerical optimization).

Whereas some time-out detection can be used to address the former issue, the latter one can be partially overcome if the grain is set to very small values, as individuals will be sent out for evaluations upon request of the workers.

3. ***Distributed evaluation of a single solution:*** The quality of each solution is evaluated in a parallel centralized way. That model is particularly interesting when the evaluation function can be itself parallelized as it is CPU time-consuming and/or IO intensive. In that case, the function can be viewed as an aggregation of a certain number of partial functions. The partial functions could also be identical if for example the problem to deal with is a data mining one. The evaluation is thus data parallel and the accesses to data base are performed in parallel. Furthermore, a reduction operation is performed on the results returned by the partial functions. As a summary, for this model the user has to indicate a set of partial functions and an aggregation operator of these.

Parallel distributed local searches

We will first present the main existing local search methods and their working principles. Afterward, we describe the main existing parallel/distributed models of their design and implementation.

Local searches: All meta-heuristics dedicated to the improvement of a single solution are based on the concept of neighborhood. They start from a solution randomly generated or obtained from another optimization algorithm, and update it, step by step, by replacing the current solution by one of its neighboring candidates. Some criterion have been identified to differentiate such searches: the heuristic internal memory, the choice of the initial solution, the candidate solutions generator, and the selection strategy of candidate moves. Three main algorithms of local search stand out: Hill Climbing (HC) [Papadimitriou, 1976], Simulated Annealing (SA) [Kirkpatrick et al., 1983] and Tabu Search (TS) [Glover, 1989].

Hill Climbing (HC): [Papadimitriou, 1976] is likely the oldest and simplest optimization method. At each step, the heuristic only replaces the current solution by the neighboring one that improves the objective function. The search stops when all candidate neighbors are worse than the current solution, meaning a local optima is reached. Variants of HC may be distinguished according to the order in which the neighboring solutions are generated (deterministic/stochastic), and according to the selection strategy (choice of the best neighboring solution, the first best, etc.)

Simulated Annealing (SA): [Kirkpatrick et al., 1983] is sort of stochastic HC. In addition to the current solution, the best solution found since the beginning of the execution is stored. Moreover, few parameters control the progress of the search, which are the temperature and the iteration number. SA enables under some conditions the decrease of a solution. At each step, the neighborhood consists of only one candidate move randomly selected, that replaces the current solution if it is better. Otherwise, the new solution could be accepted with a given probability.

Tabu Search (TS): [Glover, 1989] is similar to the steepest deterministic HC. Besides, it manages a memory of the solutions or moves recently applied, which is called the tabu list. When a local optima is reached, the search carries on by selecting a candidate worse than the current solution. To avoid the previous solution to be chosen again, and so to avoid cycles, TS discards the neighboring candidates that have been previously applied. Yet, if a candidate is proved to be very good, it could be accepted. This mechanism is called the aspiration criterion.

Parallel local searches: Two parallel distributed models are commonly used in the literature: the parallel distributed exploration of neighboring candidate solutions model and the multi-start model.

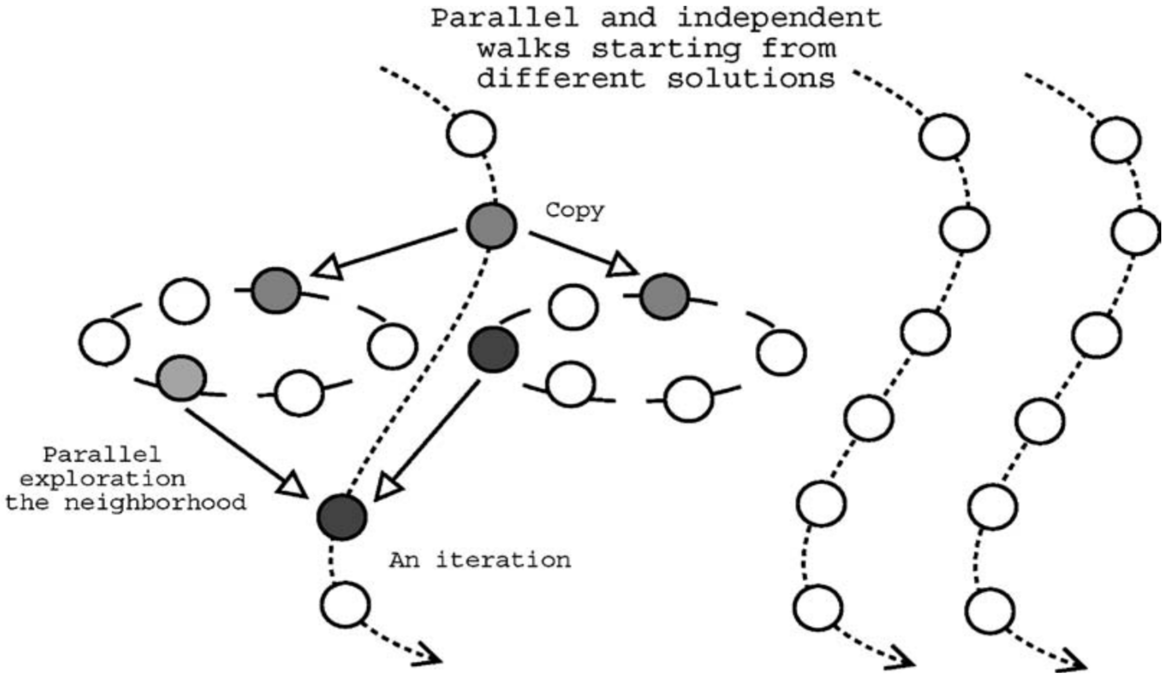
Parallel and independent walks starting from different solutions
Copy
Parallel exploration the neighborhood
An iteration

Figure 3.2: Two parallel models for local search [Cahon et al., 2004].

Parallel exploration of neighboring candidates: It is a low-level Farmer-Worker model that does not alter the behavior of the heuristic. A sequential search computes the same results slower. At the beginning of each iteration, the farmer duplicates the current solution between distributed nodes. Each one manages some candidates and the results are returned to the farmer. The model is efficient if the evaluation of a each solution is time-consuming and/or there are a great deal of candidate neighbors to evaluate. This is obviously not applicable to SA since only one candidate is evaluated at each iteration. Likewise, the efficiency of the model for HC is not always guaranteed as the number of neighboring solutions to process before finding one that improves the current objective function may be highly variable.

Multi-start model: It consists in simultaneously launching several local searches. They may be heterogeneous, but no information is exchanged between them. The results would be identical as if the algorithms were sequentially run. Very often deterministic algorithms differ by the supplied initial solution and/or some other parameters. This

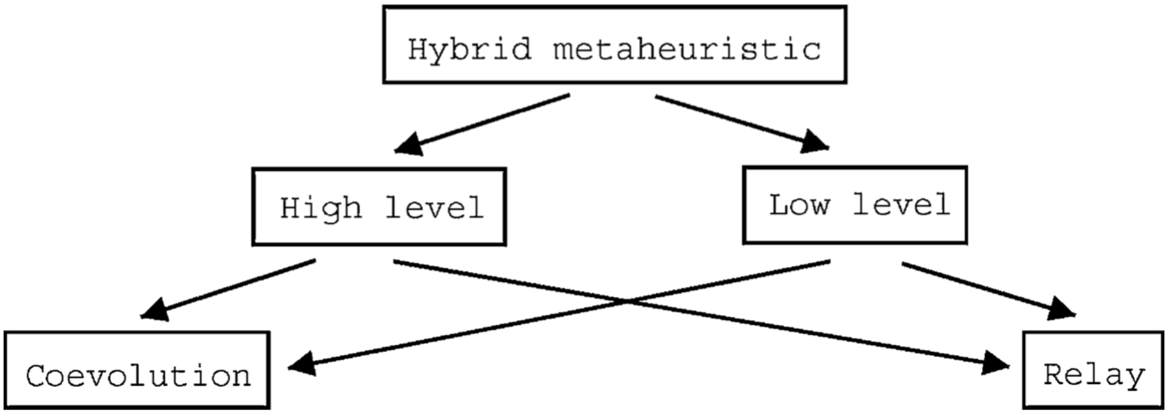

Figure 3.3: Hierarchical taxonomy of hybrid Meta-heuristics [Cahon et al., 2004].

trivial model is convenient for low-speed networks of workstations.

Hybridization

Recently, hybrid meta-heuristics have gained a considerable interest [Talbi, 2002]. For many practical or academic optimization problems, the best found solutions are obtained by hybrid algorithms. Combinations of different meta-heuristics have provided very powerful search methods. In [Talbi, 2002], E.-G. Talbi has distinguished two levels and two modes of hybridization (see Figure 3.3): Low and High levels, and Relay and Cooperative modes. The low-level hybridization addresses the functional composition of a single optimization method. A function of a given meta-heuristic is replaced by another meta-heuristic. On the contrary, for high-level hybrid algorithms the different meta-heuristics are self-containing, meaning no direct relationship to their internal working is considered. On the other hand, relay hybridization means a set of meta-heuristics is applied in a pipeline way. The output of a meta-heuristic (except the last) is the input of the following one (except the first). Conversely, co-evolutionist hybridization is a cooperative optimization model. Each meta- heuristic performs a search in a solution space, and exchange solutions with others.

3.3.2. Parallel ACO for the Traveling Salesman Problem

We can find in [Manfrin et al., 2006] a parallel solution for this problem. The MaxMin version of the ACO algorithm is used in this parallelization. One of the main results of this paper is that the simplest way of parallelizing the ACO algorithms, based on parallel independent runs, is surprisingly effective.

Traveling Salesman Problem (TSP): Over a certain amount of nodes, all of them connected and with a defined distance, we want to find the shortest Hamiltonian path. In other words, the shortest path in which every node is visited only once and returns to the origin.

In order to have a version that was easy to parallelize, the occasional pheromone re-initialization was removed from the MMAS implementation and only a best-so-far pheromone update was used. The topologies we studied are:

Fully-connected In this parallel model, k colonies communicate with each other and cooperate to find good solutions. One colony acts as a master and collects the values of the best-so-far solutions found by the other $k-1$ colonies. The master then broadcasts to all colonies the identifier of the colony that owns the best solution among all k colonies so that everybody can get a copy of this solution. A synchronous and an asynchronous implementation of this model is considered, identified by **SFC** and **AFC**, respectively, in the following.

Replace-worst This parallel model is similar to the fully-connected, with the exception that the master identifies also the colony that owns the worst solution among the k colonies. Instead of broadcasting the identity of the best colony, the master sends only one message to the best colony, containing the identity of the worst colony, and the best colony sends its best-so-far solution only to the worst colony. We consider a synchronous and an asynchronous implementation of this model identified by **SRW** and **ARW**, respectively, in the following.

Hypercube In this model, k colonies are connected according to the hypercube topology (see [14] for a detailed explanation of this topology). Practically, each colony is located on a vertex i of the hypercube and can communicate only with the colonies that are located in the vertex that are directly connected to i. Each colony sends to each of its neighbors its best-so-far solution. We consider a synchronous and an asynchronous implementation of this model respectively **SH** and **AH** in the following.

Ring Here, k colonies are connected in such a way that they create a ring. We have implemented a unidirectional ring, so that colony i sends his best-so-far solution only to colony $(i + 1) \text{ mód } k$, and receives only the best-so-far solution from colony $(i - 1 + k) \text{ mód } k$. We consider a synchronous and an asynchronous implementation of this model, called **SR** and **AR** in the following.

Parallel independent runs In this model, k copies of the same sequential MMAS algorithm are simultaneously and independently executed using different random seeds. The final result is the best solution among all the k runs. Using parallel independent runs is appealing as basically no communication overhead is involved and nearly no additional implementation effort is necessary. In the following, we identify the implementation of this model with the acronym **PIR**.

These topologies allows to consider decreasing communication volumes, moving from more global communication, as in fully-connected, to more local communication, as in ring, to basically no communication, as in parallel independent runs.

On average, all the parallel models, except SFC, seem able to do better than SEQ and SEQ2, but that the best performing approach is PIR. The differences in performance of all the parallel models with information exchange from those of PIR are statistically significant. This confirms that PIR is the best performing approach under the tested conditions.

On the other hand, the impact of communication on performance seems negative. An apparent problem of the communication scheme is that the communication is too frequent. The reduced frequency in communication has indeed a positive impact on the performance of the two parallel algorithms SRW2 and SR2. To achieve better results than PIR a more sophisticate communication scheme has to be develop, that is dependent not only on the instance-size, but also on the run time.

As we can see these seems promising. However, this solution is not a parallelization in the algorithm implementation. It is a parallel executions of several sequential instances of the same algorithm. This is the first approach in parallelization for these types of meta-heuristics.

3.3.3. A parallel bi-objective hybrid meta-heuristic for energy-aware scheduling for cloud computing systems

We have found an interesting solution for the scheduling problem parallel meta-heuristic. In [Mezmaz et al., 2011] was made a research about the problem of scheduling precedence-constrained with parallel applications on heterogeneous computing systems (HCSs) like cloud computing infrastructures. This kind of application was studied and used in many following research works. Most of these works propose algorithms to minimize the completion time without paying much attention to energy consumption.

The authors proposed a new parallel bi-objective hybrid genetic algorithm that takes into account energy consumption as well as makespan. They particularly focus on the island parallel model and the multi-start parallel model, both of them we have explained it in previous sections. Their method is based on dynamic voltage scaling (DVS) to minimize energy consumption.

In terms of energy consumption, the obtained results show that this approach outperforms previous scheduling methods by a significant margin. In terms of completion time, the obtained schedules are also shorter than those of other algorithms. However, the real impact of this project was the energy efficiency.

Their approach has been evaluated with the Fast Fourier Transformation task graph. Experiments show that the bi-objective meta-heuristic improves on average the results obtained in the literature particularly in energy saving. The energy consumption is reduced by 47.5 % and the completion time by 12 %. The experiments of the insular approach also show that the more islands used, the better the results will be. Furthermore, the multi-start approach is on average 13 times faster than the island approach using 21 cores.

However, it is observed that the hybrid approach consumes more resources than ECS, and the insular approach consumes more resources than the hybrid approach. In the insular approach, experiments show that the more islands used, the more the resources are needed. A resource can be a processor, a network bandwidth, etc. The energy consumed by an approach increases when the used resources increase, which makes sense.

3.3.4. Choco Solver: A constraint programming library

Choco [Prud'homme et al., 2016] is a Free and Open-Source Java library dedicated to Constraint Programming. It aims at describing hard combinatorial problems in the form of Constraint Satisfaction Problems and solving them with Constraint Programming techniques.

The first version of Choco was written in 1999 by Francois Laburthe and Narendra Jussien in Claire. The main objective was to provide an open source library for constraint programming dedicated to teaching and research.

Choco is one of the more mature free open source Java library for constraint programming. In a nutshell:

- Three types of variables: integer, boolean and set.
- More than 70 constraints : the most useful and state-of-the-art implementations.
- PLM framework allows configurable searches and most wide-spread search strategies.
- Deal with satisfaction and optimization (mono and multi) problems, multi-thread resolution.

3.4. Big Data computing platforms and Operations Research

The OR field is rich in complex problems and sophisticated algorithms that can take advantage of parallelization offered by Big Data computing platforms. However, two circumstances can be occurring 1) there are only a few parallel algorithms and parallel implementations of them and 2) the traditional algorithms in the literature do not fit to be adapted to one parallel version in a direct and not complicated way.

In the context of the situation 1) many of the algorithms and implementations in ultra - scalable platforms founded in the literature were deployed in platforms with scale up via GPU computing as can be consulted in [Boy, 2017]. In the context of Big Data platforms offering scale in, we can found Giraph, PowerGraph, Spark, etc. In the next section, we introduce Apache Spark and how it is used in the Operations Research field.

3.4.1. Apache Spark

Apache Spark is a cluster computing platform designed to be fast and general purpose [Karau et al., 2015]. It is an open source project that has been built and is maintained by a diverse community of developers. It started in 2009 as a research project in the UC Berkeley RAD Lab. The researchers in the lab had previously been working on Hadoop MapReduce, and observed that MapReduce was inefficient for iterative and interactive computing jobs. Thus, from the beginning, Spark was designed to be fast for interactive queries and iterative algorithms, bringing in ideas like support for in-memory storage and efficient fault recovery.

Speed is important in processing large datasets, as it means the difference between exploring data interactively and waiting minutes or hours. One of the main features Spark offers for speed is the ability to run computations in memory, but the system is also more efficient than MapReduce for complex applications running on disk.

On the generality side, Spark is designed to cover a wide range of workloads that previously required separate distributed systems, including batch applications, iterative algorithms, interactive queries, and streaming. By supporting these workloads in the same engine, Spark makes it easy and inexpensive to combine different processing types, which is often necessary in production data analysis pipelines. In addition, it reduces the management burden of maintaining separate tools.

Spark is designed to be highly accessible, offering simple APIs in Python, Java, Scala, and SQL, and rich built-in libraries. It also integrates closely with other Big

Data tools. In particular, Spark can run in Hadoop clusters and access any Hadoop data source, including Apache Cassandra [Apache software, 2000].

The Spark Stack

The Spark project contains multiple closely integrated components. At its core, Spark is a "computational engine" that is responsible for **scheduling, distributing, and monitoring** applications consisting of many computational tasks across many worker machines, or a computing cluster. Because the core engine of Spark is both fast and general-purpose, it powers multiple higher-level components specialized for various workloads. These components are designed to inter-operate closely, letting you combine them like libraries in a software project.

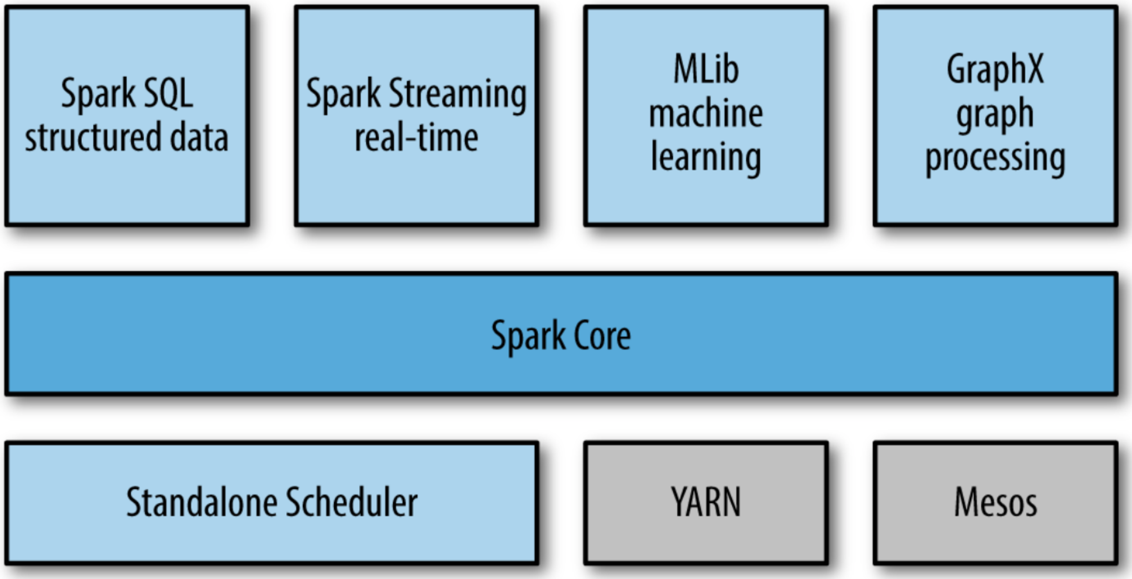

Figure 3.4: The Spark Stack [Karau et al., 2015]

We are going to explain briefly the main elements in the Spark Stack.

1. **Spark Core:** contains the basic functionality of Spark, including components for task scheduling, memory management, fault recovery, interacting with storage

systems, and more. Spark Core is also home to the API that defines resilient distributed datasets (RDDs), which are Spark?s main programming abstraction

2. **Spark SQL:** is Spark's package for working with structured data. It allows querying data via SQL

3. **Spark Streaming** is a Spark component that enables processing of live streams of data. Examples of data streams include logfiles generated by production web servers, or queues of messages containing status updates posted by users of a web service

4. **MLlib:** contains common machine learning functionality such as classification, regression, clustering, and collaborative filtering, as well as supporting functionality such as model evaluation and data import.

5. **GraphX:** is a library for manipulating graphs (e.g., a social network's friend graph) and performing graph-parallel computations

6. Spark also offers a variety of **Cluster Managers**, including Hadoop YARN, Apache Mesos, and a simple cluster manager included in Spark itself called the Standalone Scheduler. This allow Apache Spark to efficiently scale up from one to many thousands of compute nodes.

3.4.2. Apache Spark in OR

There is not a lot of tools out there developed for Operations Research in Spark. We analyze some of these few tools in order to find an starting point which can help us in the developing process. We would like to introduce some of the most interesting ones we came across.

Spark OR: (Spark Operational Research) is build on Apache Spark in Scala which aims to solve optimization problems. It uses Spark 1.6.1 and Mllib 1.6.1 to distribute calculations. This package allows us to solve linear systems [Petit et al., 2016]. The general way to solve a optimization problem using Spark OR is quite simple. Here are the key steps:

1. Define the parameters of your problem using one of the provided Problem classes (for instance, *LinearOptimizationProblem*).

2. Create a new solver for your problem. If you do not not which solver is the most appropriate for your problem, you can use the *GenericSolver* class. This solver will use the default solver for your problem type.

3. (Optional) Tune the parameters of your solver.

4. (Optional) Set up your solving callbacks.

5. Call the *solve()* method on your solver

6. Get the score of your solution by calling the *getScore()* method on your solver.

A diagram showing the Spark OR architecture can be seen in the Figure 3.5.

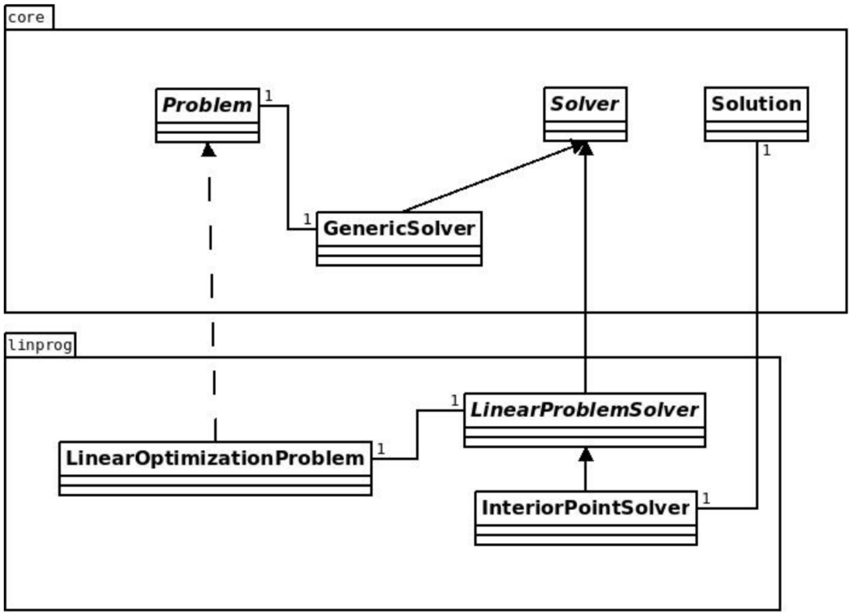

Figure 3.5: The Spark OR architecture [Petit et al., 2016]

ACO-pants: Pants is a Python3 implementation of the Ant Colony Optimization Meta-Heuristic. It is not meant to be distributed, but, because it is implemented in

Python, it can be blended with Spark easily. That's the reason we included it at this point. Pants provides a tool for solving the Traveling Salesman Problem

3.4.3. Ant colony optimization algorithm based on Spark

There have been previous approximations to a parallel solutions of the ACO algorithm. In [Wang et al., 2015] it is described an implementation of the ACO algorithm in Spark for the TSP. The approximation here is based in the ability of Spark to convert any kind of object list in an instance of an RDD (Resilient Distributed Dataset). In the section 3.4.1 we give a brief introduction of how Spark works and operates with this RDD.

The steps for reproducing the architecture in this paper are:

1. Initialize the pheromone and distance matrix.

2. Encapsulate the functionality of a single ant in a separate class. This class will have the methods which each ant should do in every iteration, such as: choose the next node based on its likeness, compute the distance between each node, keep the information of the path followed, etc.

3. Make an ant colony as an array of this ant class and transform this array in a RDD with the *parallelize()* method in Spark.

4. In each iteration, we call the Spark's *map()* function over the ant colony RDD to start the ants task in parallel. The output should be another RDD wich contains the information about the followed paths. This RDD should be ordered from the best to the worst solution

5. Then we use the *reduce()* function to get the best solution so far and we update the pheromone matrix based in the followed paths.

This design is very intuitive because is really similar to a multi-threading solution. The key difference here is the use of Spark and its outstanding ability to start and kill task in parallel thanks to its RDD.

Chapter 4

Parallel and Distributed Maximum Ant Colony Optimizer: design and development

As was introduced in the Chapter 1, the goal of this Project is to propose a highly parallel and distributed solution to the resources assignment to tasks (an extension of the classical GAP problem) from the work proposed in [Moreno Navas, 2015] named "Maximun Ant Colony Optimizer" (MACO in short). MACO has proposed a very smart and complete sequential solution for the resources assignment to tasks problem from constraint programming and bio-inspired meta-heuristics. MACO was applied to a use scenario related with railway network using a public dataset.

In this chapter we introduce the parallel version of MACO. Specifically, we present the new parallel algorithm using IP and ACO approaches and introduce a brief discussion about its design; after we will analyze the algorithm and focus it within a use scenario corresponding to a railway network.

4.1. Problem Analysis

In this section, we introduce the scenario corresponding with a generic railway network in which it is desired to solve solution the assignment of resources to tasks.

Informally the scenario can be described by:

Given a railway network, we want to perform the strategic planning of it. This network is defined by its stations, the connections between them, the train lines and the arrival times at each station for each train of the lines. Each train line should be covered by a series of resources that comply with the training required by it, be drivers of diesel engines, be vigilant, etc. The resources can go on trains as passengers to get to the stations and trains where they have been assigned. Each resource has a base station from which they begin their working day. For each resource, a daily work limit of 8 hours is defined.

Within this scenario the problem is to achieve the maximum possible coverage. This means, the lowest number of unassigned tasks, with the minimum number of possible resources. Due to the restrictions imposed in the executions in time or number of possible iterations, the solutions found will tried to be as close as possible to the optimum. In this scenario, the optimal result is known therefore we will able to compare the obtained solutions as the expected results.

4.1.1. Constraint Programming Proposed Model

The entire Constraint Programming (CP) model is based on a dynamic step approach. Each resource has an initial step to which will be assigned new steps whose limit is the day of the resource. A task is assigned at each step. Each step knows its previous and next task.

Hereafter, we introduce the the data entities and the restrictions to model the CP instance to the railway network using IP.

Data entities

We are going to enumerate the data entities which define the model of the problem:

- S Stations. Stations with a stop.

$$S = [0..s] \tag{4.1}$$

- CP Checkpoints. Stations with arrival times.

$$CP = \{\{s, t\}\} \, \forall s \in S, t \in \mathbb{N} \tag{4.2}$$

- R Resources. Human Resources to cover the needs of the problem.

$$R = [0..r] \tag{4.3}$$

- Tr Trains. Train lines which need to be covered. There are a list of checkpoints sorted by time where a checkpoint follow another if the arrival time of the first is less than the second one.

$$Tr = \{cp\} \, cp \in CP, \forall i.j : j = i + 1 \rightarrow t_i < t_j \tag{4.4}$$

- W_r Workday for the resource r.

- B_r Base station for the resource. It represent where the resource is going to start his workday.

$$B_r = \{\{r, s\}\} \, r \in R, s \in S \tag{4.5}$$

- Ta Tasks. Each task is represented with a initial checkpoint and a final checkpoint of a train.

$$Ta = \{cp_i, cp_j\} \; t_i < t_j \tag{4.6}$$

- $Ta_{\oslash}$ Empty task.

- M_r Worked minutes for the resource r.

- A Assignation variable between tasks and resources

$$A = [0..n], \, n = |Ta| \tag{4.7}$$

- St_r Resource steps. Each step is made by a variable to reference the previous step P_p, a reference to the next step P_s and the accumulated worked minutes for the resource M_r.

$$S_r = \{A, S_a, S_s, M_r\} \tag{4.8}$$

Constraints

The constraints applied for the instance problem to model related with the the assignation variable A and M_r are:

- The resources mustn't work more minutes than their workday.

$$M_r \leqslant W_r \tag{4.9}$$

- The only tasks that can be assigned in the first step are the empty task and those which have the initial checkpoint in the base station of the resource.

$$St_r^0 \in [Ta_{\oslash}, Ta_{B_r}] \tag{4.10}$$

- In two successive steps such as:

$$St_r^k, St_r^l,\ k < l,\ k \geq 0,\ l \geq 1 \tag{4.11}$$

$$A_r^k \equiv Ta_r^k \tag{4.12}$$

$$A_r^l \equiv Ta_r^l \tag{4.13}$$

$$Ta_r^k = \{cp_0^k, cp_f^k, c\} \tag{4.14}$$

$$Ta_r^l = \{cp_0^l, cp_f^l, c\} \tag{4.15}$$

- In two successive steps, the assigned initial task and the next one must share the final station of the first one and the initial station of the second.

$$s_f^k = s_0^l \tag{4.16}$$

- In two successive steps, the assigned initial task and the next one must be sorted in time. So the arrival time to the destination checkpoint of the first task must be before the arrival time to the initial checkpoint of the second task.

$$t_f^k < t_0^l \tag{4.17}$$

- A resource can be assigned to an already assigned task as a passenger.

- A task can be only covered once.

$$0 \leq |A_r^k \equiv Ta_r^k| \leq 1 \tag{4.18}$$

4.1.2. Problem modelling: proposals and adaptations

MACO uses the combination of CP and ACO to reach optimal solutions. We take the same idea for the constraint model described in the previous section but applying some simplifications in the instance problem formulation itself in order to keep the scope of this Final Project Dissertation.

The great big change with regard to the original model was the removal of the training restrictions in the trains. This change makes the whole system easier to model because none of the tasks need a specific type of resource to be covered. To cover a tasks, we just need a single resource. In other words, we will just assign the drivers to the trains between two stations.

Another big important part in order to combine ACO + CP is the use of a Constraint Programming library or motor. The one used in [Moreno Navas, 2015] was Choco Solver. Choco Solver is a powerful open source CP library for Java, allow us to construct a search method for the solution to the CP problem model with it. So Choco Solver is the one in charge to ensure that all the restrictions for the problem are satisfied, and we can make the ACO algorithm as a custom search to find solutions with Choco.

In a first iteration, we try to use Choco in the Spark architecture. This was a really big problem because a Java class must implements the Serializable interface in order to work in Spark inside a RDD. Unfortunately, none of the classes in the Choco library implements that interface. This means that we need to modify the library to work with Spark, and change the Choco's classes to implement the Serializable interface. Although we were able to change the whole Choco library to work in Spark, the nature of the custom search in Choco with ACO made impossible to implement the distributed ACO algorithm developed with Spark and Choco at the same time. So we decided to focus in the development of the distributed version in Spark, and implement the restriction checking ourselves with another solutions.

In the following section, we will explain how we ensure the restrictions of the problem are feasible.

4.2. Development of the Proposed Solution

In this section we present the model for the ACO + CP proposed in the previous section adapted into a parallel version in order to be deployed in Spark.

4.2.1. Components and classes

We defined the following components for the problem:

- Trains: Represents the train lines. They will have the number of stops, the stations of the line in chronological order, and the schedule for each stop.

- Checkpoint: It has the name of the station and its arrival time.

- Task: represent the task to be covered. It has an initial checkpoint and a final checkpoint.

- Step: reference to the task that is covered in a step for a resource.

- Resource: represent a resource. It will hace a list with all the steps in chronological order and the worked minutes. It will have a function to add an step, and to check that a step can be added, in other words, a function to check if the constraint model is feasible if we add a certain step.

- DataModel: is the main data structure of the project. It will hold all the tasks, the resources, the stations and the tasks covered in total.

- Ant: is the responsible of building the solutions and the assignations. Each ant will have a base DataModel built from the data. When the ant construct the solution, it will return a new instance of data model wich will include all the assignations for the resources

4.2.2. Algorithm design

In order to implement the algorithm for ACO + CP in a parallel version to be deployed into Spark, we first take as base the implemented algorithm given in [Moreno Navas, 2015] which was introduced in 3.2.2.

The approach taken to parallelize the algorithm was inspired by the work in made in [Wang et al., 2015]. The main idea of the algorithm is to take the advantages of the Map Reduce framework in Spark. Following this framework, we can make all the ants in the colony to operate in a parallel way. We can achieve this converting the list of objects representing ants in the colony into a Spark RDD. The parallel proposed algorithm for ACO + CP to be deployed in Spark is shown in the Algorithm 15.

Algorithm 15 ParallelACO() (ACO with Spark Characterization)

```
readProblemData();
pheromoneMatrix = initializePheromoneMatrix();
distanceMatrix = initializeDistanceMatrix();
ants = Array[Ant](numberOfAnts);
parallelAnts = sparkContext.parallelize(ants);
bestSolution = null;
bestScore = 0;
unImprovementCount = 0;
for numberOfIterations do
    solutions = parallelAnts.map(ant =¿ant.findSolution()).sortBy(solution =¿.getAssignationScore, ascending = false)
    iterationBestSolution = solutions.first();
    if iterationBestSolution.getAssignationScore ¿bestScore then
        bestScore = iterationBestSolution.getAssignationScore();
        bestSolution = iterationBestSolution;
        unImprovementCount = 0;
    else
        UnImprovementCount += 1;
    end if
    if unImprovementCount ¿25 then
        resetPheromonematrix();
        unImprovementCount = 0;
    else
        collectedSolutions = solutions.collect();
        updatePheromoneMatrix(collectedSolutions);
    end if
end for
```

We need to define two functions:

- *solve()* which find a solution for the assignation in the railway network and returns it. This function is in the Ant class. It returns an object of the type DataModel. This function uses the same algorithm as it is specified in [Moreno Navas, 2015].

- *getAssignationScore()* which returns the score for the assignation. This function is in the class DataModel. The score for each solution is based in the number of tasks covered and the number of resources used. So first, we want get the maximum number of tasks covered. If all the tasks can be covered, we prioritize those solutions which use the minimum number of resources. We calculate a integer, and return it.

With the Spark's *map()* function, *solve()* is called for each ant in the colony RDD. This operation returns another RDD with a list of DataModel objects. Then, sort this RDD by the individual scores of the solution objects. To do so we call the Spark's *sort()* function. This function will call the *getAssignationScore()* function for each solution object and sort them descending order.

We the function *take(1)* we get the best solution found by the ants in this iteration. We stored the best solution and update it if we found another better solution in the next iterations. Then we continue updating the pheromone matrix.
In the figure 4.1 we can see a visual representation of the adapted MACO algorithm.

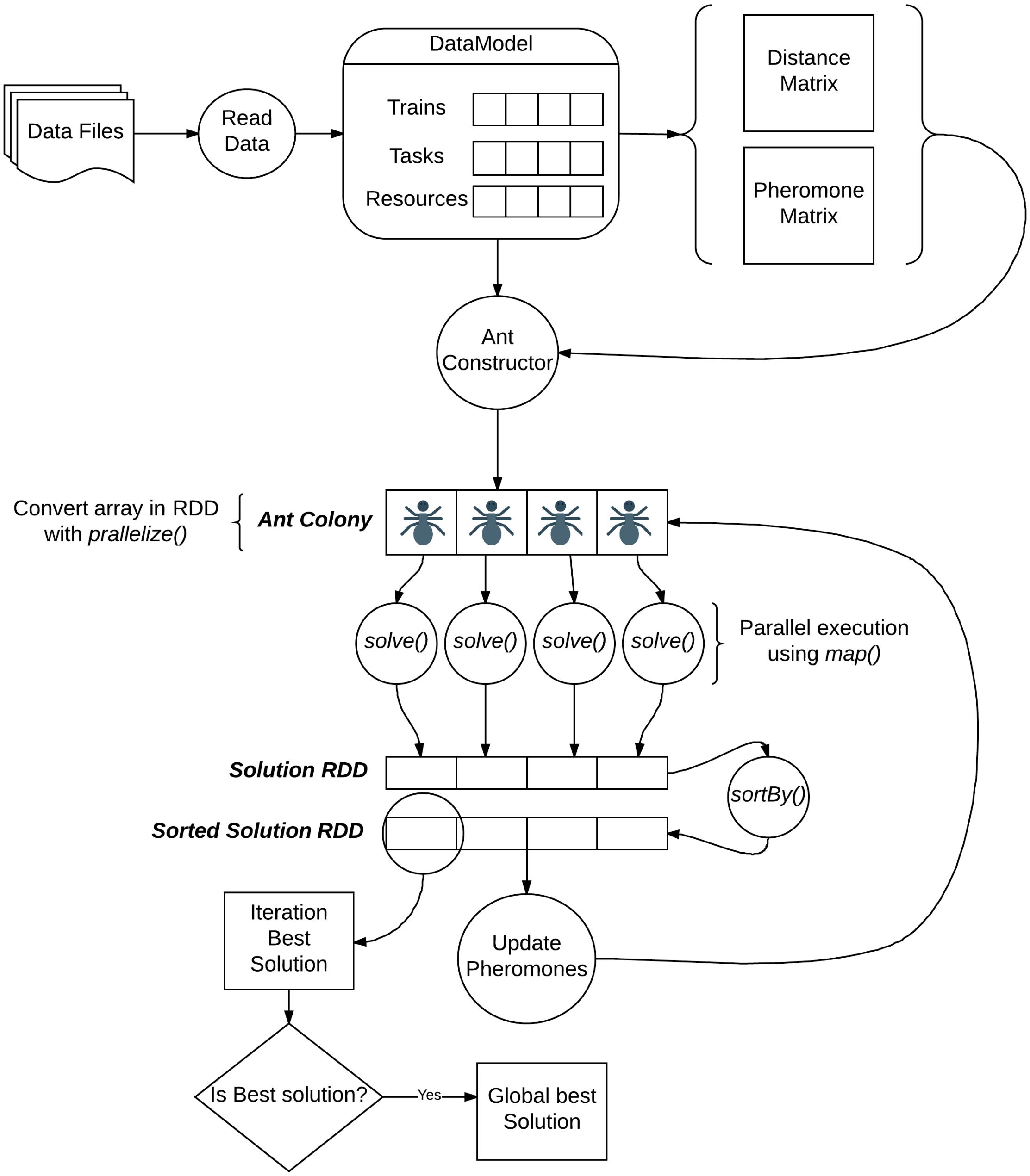

Figure 4.1: Visual representation of the distributed MACO algorithm

4.2.3. Pheromone Matrix

The pheromone matrix is the main concept on the ACO algorithms. In our proposal, it is a matrix with column and row size of the number of tasks created. All the matrix is initialized to the same pheromone.

The pheromone deposit is based in the number of tasks that have been created. The more not covered tasks, the less is the deposit. We follow the next equation to reinforce the pheromone paths, where Q is the reward and H is the number of not covered tasks:

$$\begin{cases} f(i,j) = f(i,j) + Q/H & H > 0 \\ f(i,j) = f(i,j) + Q & H = 0 \end{cases}$$

In the case of the deposit for each ant iteration, we have $Q = 10/A$ where A is the number of ants. In the case of the *daemon* deposit, this value is $Q = 2/A$.

4.2.4. Distance Matrix

The ACO algorithm uses a distance matrix in order to calculate the probability of traverse from one node to another in addition to the pheromone matrix. This matrix have the same size. So, our distance matrix have so many columns and rows as the number of tasks created. Each value of the distance matrix represents a metaphoric distance between nodes. The main idea for our problem is that this matrix also have the purpose to check if an assignation is possible if we have a certain previous tasks assigned. For that reason, in our problem, the distance matrix will give more value to the tasks closed in time, and will set the tasks which are impossible to traverse from one to the other a negative value. So, when we compute the probability to one tasks to another, we can check with one single cmnprobation if the nodes are accessible.

4.2.5. ACO Hyperparameters

As the ACO algorithm has a very defined constants an hyperparameters, we develop our program with the following ones:

- It use the *daemon* actions. The pheromones of the best ant is reinforced for each iteration and for the best global solution.
- α is equal to 1
- β is equal to 2
- ρ is equal to 0.05
- The exploration probability is equal to 0.75
- The initial pheromone value is equal to 0.5
- Each 25 iterations the pheromone matrix is restarted if the assignation hasn't improved.

The *daemon* reinforcement is focused firstly in the coverage of the maximum number of tasks. If all the solutions founded equals the coverage, it is choose the one with the minimum number of employees.

Chapter 5

Parallel Maximum ACO implementation

5.1. Implementation of the algorithm

In the following section we are going to present how was the algorithm implemented in Spark.

5.1.1. Implementation

In order to develop an algorithm to work on Apache Spark, we have to choose between 3 different programming languages: Scala, Java or Python. Our first option was Python. Is a compact and concise language who can give us speed in the development. However, in the first iteration of the project, we wanted to use Choco Solver as it was used in [Moreno Navas, 2015]. This library is written in Java. Because of that, we decided to do our development in Java.

The main drawback with this programming language is that their nature is not related with the functional programming paradigm. Java was develop as an object oriented language. Despite of the new lambda functions included in the language in Java 8, it is still a very verbose and complex to use functionality, than a more functional programming language as Scala. So, we decided to implement all the classes and components for the ACO algorithm in Java, and all the components which interact with Spark in

Scala. Thanks to the interoperability of both languages, it is trivial to combine them in a project.

We can see in the figure 5.1 a UML graph of the classes and dependencies of our project

5.2. Libraries used

We have the following libraries and dependencies for our project:

- Apache Spark - Spark Core: Version 1.6.1
- Scala: Version 2.11.8
- Java: Version 1.8

The main library in our project is the Spark Core, which provide us all the API interface to interact with Spark.

We manage and compile all these dependencies using a tool called **sbt (Simple Build Tool)**. It is an open source build tool for Scala and Java projects, similar to Java's Maven. It provides dependency management using Ivy (which supports Maven-format repositories) and continuous compilation, testing, and deployment.

5.3. Setup of the platform

We setup our system to run our program in Spark using an IDE in a single machine. We use IntelliJ IDEA, a IDE for Java, Scala, Kotlin, Groovy, and Android; to compile and execute our project. IntelliJ allow us to set up our project using *sbt* and execute it in our local machine using Spark.

In the table 5.1 we can see the system specifications of our local machine.

5.4. Dataset used

We will use the same dataset used in [Moreno Navas, 2015] in order to compare the quality of the results with the original work. In previous works with ACO and CP,

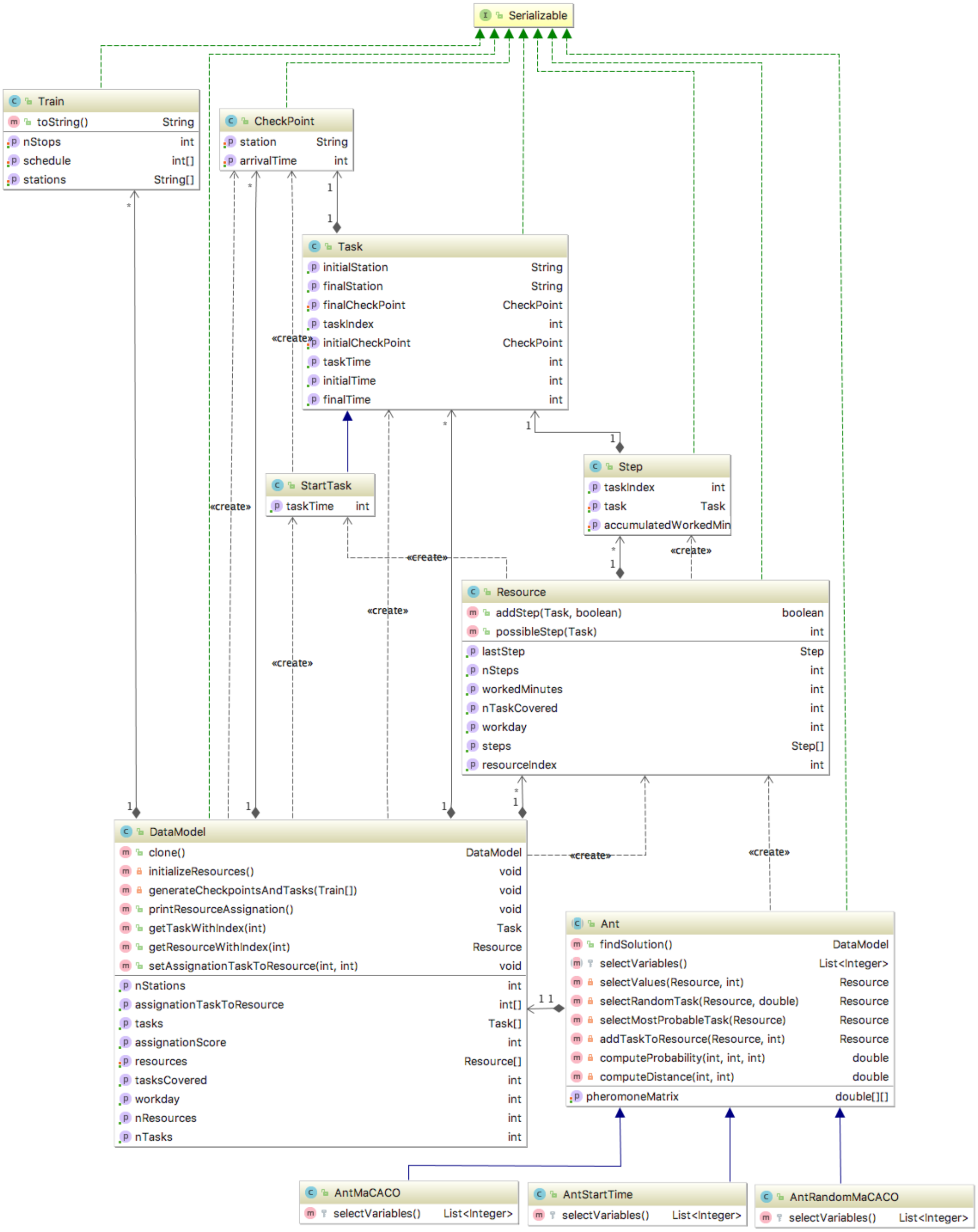
Serializable
Train
toString() String
nStops int
schedule int[]
stations String[]
CheckPoint
station String
arrivalTime int
Task
initialStation String
finalStation String
finalCheckPoint CheckPoint
taskIndex int
initialCheckPoint CheckPoint
taskTime int
initialTime int
finalTime int
StartTask
taskTime int
Step
taskIndex int
task Task
accumulatedWorkedMin
Resource
addStep(Task, boolean) boolean
possibleStep(Task) int
lastStep Step
nSteps int
workedMinutes int
nTaskCovered int
workday int
steps Step[]
resourceIndex int
DataModel
clone() DataModel
initializeResources() void
generateCheckpointsAndTasks(Train[]) void
printResourceAssignation() void
getTaskWithIndex(int) Task
getResourceWithIndex(int) Resource
setAssignationTaskToResource(int, int) void
nStations int
assignationTaskToResource int[]
tasks Task[]
assignationScore int
resources Resource[]
tasksCovered int
workday int
nResources int
nTasks int
Ant
findSolution() DataModel
selectVariables() List<Integer>
selectValues(Resource, int) Resource
selectRandomTask(Resource, double) Resource
selectMostProbableTask(Resource) Resource
addTaskToResource(Resource, int) Resource
computeProbability(int, int, int) double
computeDistance(int, int) double
pheromoneMatrix double[][]
AntMaCACO
selectVariables() List<Integer>
AntStartTime
selectVariables() List<Integer>
AntRandomMaCACO
selectVariables() List<Integer>
«create»
«create»
«create»
«create»
«create»
«create»
«create»
«create»
1
1
1
1
1
1
1
1
1
1 1
*
*
*
*
*

Figure 5.1: UML Class Diagram

CPU	Intel(R) Core(TM) i7-479k CPU @ 4.00 GHz
RAM	32.00 GB
Operating System	Windows 10, 64 bits, CPU x64

Table 5.1: System specifications

the minimum resources used to get the maximum coverage is 30 resources using the RandomMaCACO algorithm for the variables selector. The description of this dataset can be found in Chapter 7.3 (Annex). The number of resources available is 400 and the number of tasks to be covered are 539.

5.5. Verification tests

In the following section, we will present how we verify our algorithm. We got to prove that the algorithm is performing correctly and that Spark is using all the resources available in our machine.

5.5.1. Constraint fulfillment verification

In order to verify that our algorithm is performing correctly in Spark, we set up simple test cases. Both of them uses the dataset included in Chapter 7.3 but with modifications. The dataset is simplified in order to let us check what is happening.

The first test dataset uses a single train line and a single resource in the first station of the train line. In this case we checked than the assignation of the resource is performing correctly and the worked time for the resource is right with each new task assigned. We also check if the pheromone matrix and the distance matrix are build correctly and if the updates in the pheromone matrix work for each iteration.

The second test dataset uses two new train lines and 10 resources. The total number of tasks in 30. With this dataset we check the same functionality than the previous test case but with a larger scale of data. We check that the resources are not assigned to tasks that do not fulfill the constraints.

5.5.2. Spark distribution testing

The main strength of the algorithm is the distribution of the tasks with Spark. We have to verify that Spark is using all the cores in our machine and distribute the load between them. To do so we execute our algorithm and open the resources monitor in Windows. We can see in the figure 5.2, the CPU load during the execution of the parallel algorithm. We can see how the load is evenly distributed between the CPU cores in our machine, meaning that Spark can allocate the workers in our CPU.

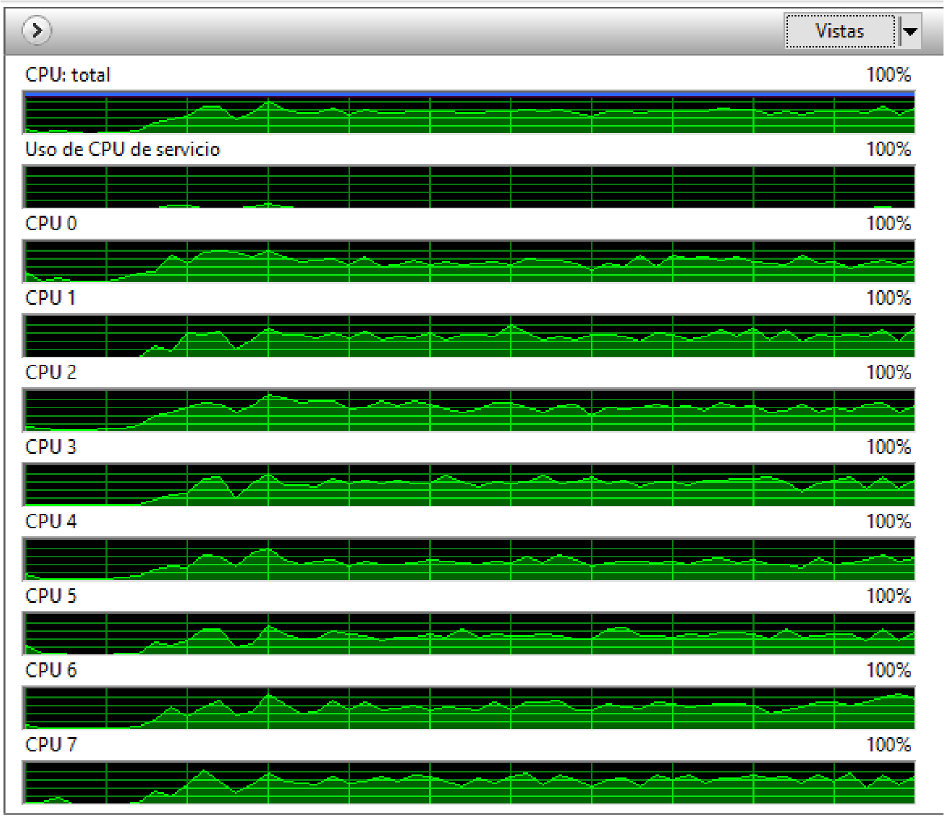

Figure 5.2: CPU consumption in the ACO+Spark algorithm

In the other hand, in the Figure 5.3 we can see the execution of the non parallel version of our algorithm. In this version, we can see that the load of CPU is randomly distributed.

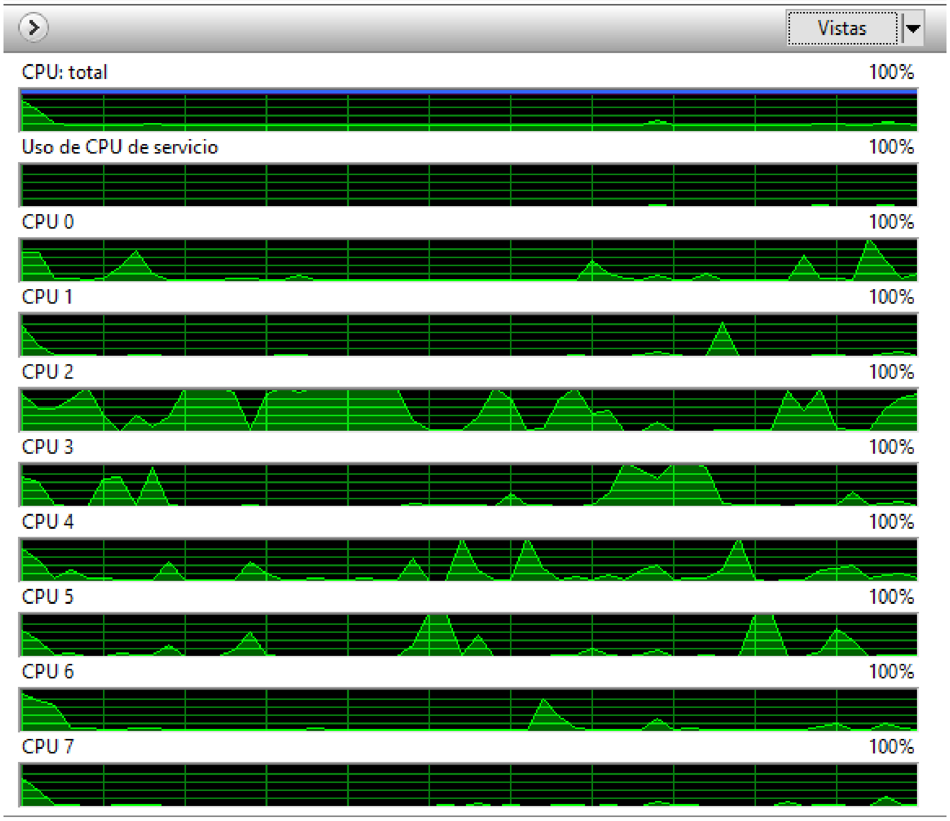

Figure 5.3: CPU consumption in the ACO sequential algorithm version

The rest of exhaustive testing proofs are not considered in the scope of this document. The testing plan, the case test design and the results about them, were firmly executed. Both, faults and expected results were registered and, in the case of the faults obtained, they were satisfactorily solved in the implementation phase of our solution.

Chapter 6

Experiments and results

The present work intends to explore the capacities of the ACO meta-heuristic in the context of a highly parallel platform as Spark to solve multiobjective problems to reach optimum, or near optimum solutions. The reason of distributed computing in this context is provide a more time-efficient algorithm when we face hard and big problems as GAP. We take advantage of the Spark platform and its Map-Reduce framework to do so.

6.1. Experiments set up

We want to compare the difference between an iterative classic implementation of the ACO algorithm versus a parallel and distributed one, to observe the possible benefits of a distributed meta-heuristic solution for the GAP problem. These benefits could be a incremental quality of the solutions, or an improvement in the execution times. In the Table 6.1, the experiments parameters are defined. We will test the experiments in different conditions for these constants. We will increment the number of iterations as well as the number of ants, and we try the combinations between them.

Variable	Value
Maximum number of Iterations	100
Maximum execution time	15 m
Number of ants in each iteration	10, 25, 50
Number of execution	50

Table 6.1: Algorithms execution set up

In the Table 6.2 we defined the following cases depending of the number of ants.

Case number	Number of Ants
Case 1	10
Case 2	25
Case 3	50

Table 6.2: Experiments scenarios in base of the number of ants and iterations

Each one of the following paragraphs define the information shown for each experiment:

1. A table with the mean time required for all the executions.
2. A table with the maximum and minimum values for the number of resources used in the executions. The percentage of the executions that have minimum and maximum values as well as the time required to reach this results. The same values are shown for maximum and minimum tasks covered.
3. A table with general statistics which includes theirs means for each case.
4. A bar graphic which represents the number of executions that reach the best results in a certain iteration.
5. A graph which shows the mean number of employees used in the assignation for each iteration.

When we mention the best value for a certain iteration, we are focusing in the best value between all the ants in the iteration.

All the times shown in the tables are in milliseconds.

The specifications of the system the experiments where run on can be found in the Table 5.1.

6.2. Experiments results

In the section we are going to present the results for the experiments introduced in the previous section. First, we are going to analyze the sequential version of the algorithm developed in Java and then we are going to present the results of the parallel version in Spark. Our goal with these experiments is to observe the impact of parallelization in the algorithm.

6.2.1. Sequential version results

In the following tables and graphics, we can see the results of the sequential version of the algorithm. The sequential version operates almost the same as the parallel one. The difference is in the execution of the tasks for the ants. The algorithm call the function *solve*() for each ant in a sequential way. Also, in each *solve*(), the ant has to clone its data model in order to not modify the original one for future iterations. This step was not necessary in the Spark version because the inherent immutability of the Spark's RDD.

The results presented in the Table 6.3 are the mean time for the executions. The time increases as we increase the number of ants proportionally. The experiments took 1.2 minutes for the case 1, 3.17 minutes for the case 2, and 6.3 minutes for the case 3.

	Case 1	Case 2	Case 3
Mean Time for the executions	75641	190217	379722

Table 6.3: Mean time for all the executions

The Table 6.4 shows the maximum and minimum values for the resources used and the unassigned tasks. As we can see, there is a slightly improvement in the minimum and maximum number of resources used as we increase the number of ants. The number of uncovered tasks in all cases is 0, reaching the maximum coverage in all the experiments for all cases. The time difference between the minimum and maximum number of resources used is very similar for each case, being maximum number of resources assigned slightly faster.

	Case 1	Case 2	Case 3
Minimum resources used	57	58	57
Minimum resources used Time	77153	188769	378740
Minimum resources used Percentage	2 %	10 %	4 %
Maximum resources used	69	66	65
Maximum resources used Time	75730	189828	380073
Maximum resources used Percentage	2 %	6 %	2 %
Maximum tasks uncovered	0	0	0
Maximum tasks uncovered Time	73416	186026	374851
Maximum tasks uncovered Percentage	100 %	100 %	100 %
Minimum tasks uncovered	0	0	0
Minimum tasks uncovered Time	73416	186026	374851
Minimum tasks uncovered Percentage	100 %	100 %	100 %

Table 6.4: Maximum and minimum values for resources and uncovered tasks in each case

In the Table 6.5, we can see some general statistics about the executions for each case. The mean number of resources used in each case decrease with the increase of the number of ants. Also, all the tasks are covered in all experiments and assigned in the first iteration.

The mayor increase is in the mean time for each iteration of the algorithm which increases proportionally with the increase of the ants. Between the case 1 and case 2, there are an increase of 250 % in the number of ants as well as a 250 % increment in time for each execution. The same situation happens between case 1 and 3, with an increment of 500 % in ants and time. The iteration which reaches the minimum number of resources increases slightly with the number of ants too.

	Case 1	Case 2	Case 3
Mean number of unassigned tasks	0	0	0
Mean number of resources used	64	62	61
Iterations to reach the minimum number of uncovered tasks	0	0	0
Iterations to reach the minimum number of uncovered tasks and resources	55	59	58
Mean time for iteration	756	1902	3797

Table 6.5: General Statistics

In the Figure 6.1 we can see the distribution of the number of executions to reach the best result in a certain iteration. We see that this distribution is not very well defined but it tends to reach the best result between the fortieth and sixty fifth iteration.

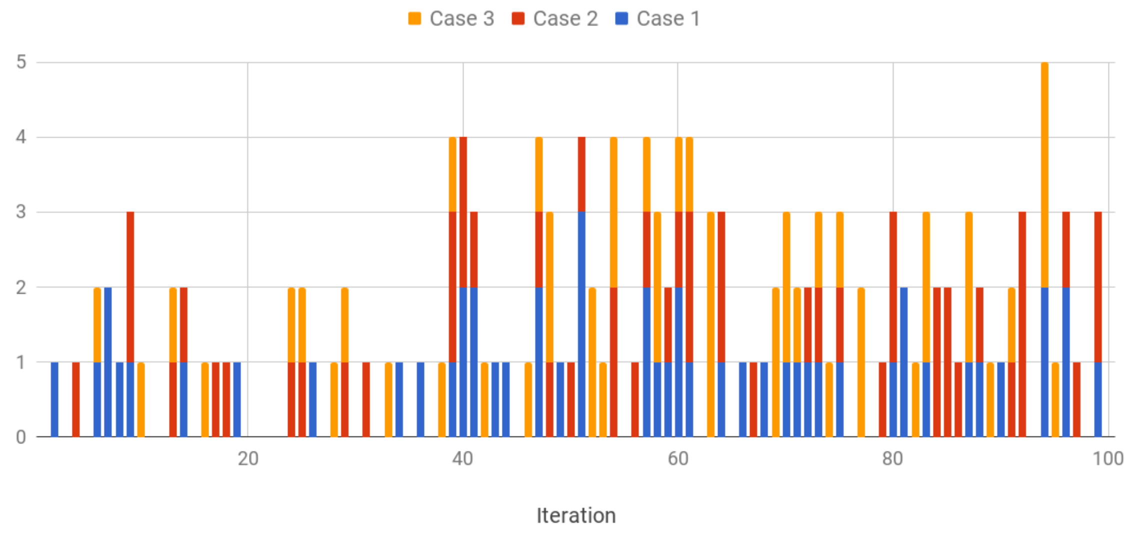

Figure 6.1: Number of executions to reach the optimal assignment in a certain iteration

We can see in the Figure 6.2, how the number of resources used decrease with each iteration. As we said before, we can see how the number of resources used decrease as we increase the number of ants. Also, with more amount of ants, the algorithm tends to converge faster, finding best results in early iterations.

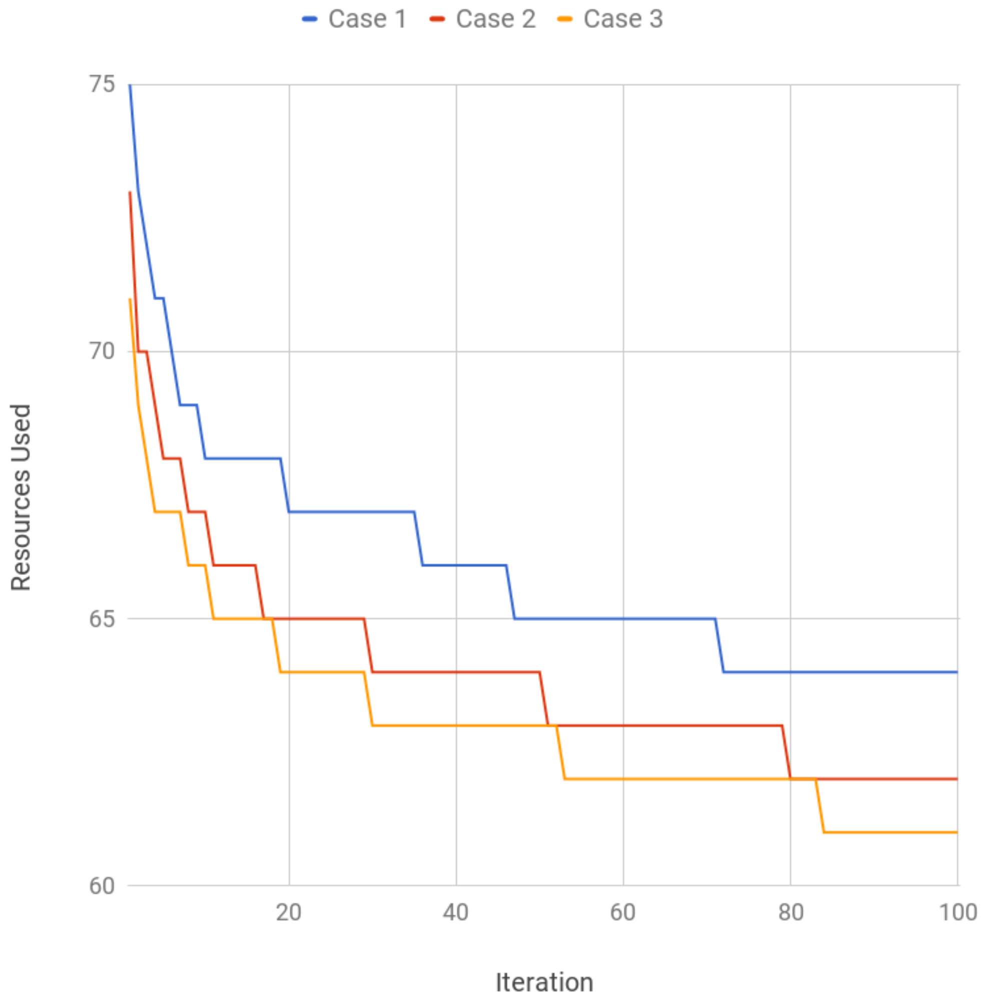

Figure 6.2: Mean number of resources used in each iteration

6.2.2. Parallel version results

In the following tables and figures, we can see the results of the parallel version of the algorithm developed in Spark and Scala with Java.

	Case 1	Case 2	Case 3
Mean Time for all the executions	29429	50995	78874

Table 6.6: Mean time for all the executions

The next results presented in the Table 6.6, are the mean times for the executions. We can see how the time increases as we increase the number of ants. The experiments took 0.49 minutes for the case 1, 0.84 minutes for the case 2, and 1.3 minutes for the case 3.

	Case 1	Case 2	Case 3
Minimum resources used	40	42	41
Minimum resources used Time	29648	51663	81064
Minimum resources used Percentage	2 %	10 %	6 %
Maximum resources used	50	47	46
Maximum resources used Time	28859	50920	78783
Maximum resources used Percentage	2 %	16 %	6 %
Maximum tasks uncovered	0	0	0
Maximum tasks uncovered Time	28859	50196	77503
Maximum tasks uncovered Percentage	100 %	100 %	100 %
Minimum tasks uncovered	0	0	0
Minimum tasks uncovered Time	28859	50196	77503
Minimum tasks uncovered Percentage	100 %	100 %	100 %

Figure 6.3: Maximum and minimum values for resources and uncovered tasks in each case

The Table 6.3 shows the maximum and minimum values for the resources used and the unassigned tasks. There is a slightly improvement in the maximum number of resources used as we increase the number of ants, but for the minimum there is no improvement. The number of uncovered tasks in all cases is 0, reaching the maximum coverage in all the experiments for all cases. The time difference between the minimum and maximum number of resources used is very similar for each case, being maximum number of resources assigned slightly faster.

	Case 1	Case 2	Case 3
Mean number of unassigned tasks	0	0	0
Mean number of resources used	45	45	43
Iterations to reach the minimum number of uncovered tasks	0	0	0
Iterations to reach the minimum number of uncovered tasks and resources	65	53	45
Mean mean time for iteration	294	510	788

Table 6.7: General Statistics

In the Table 6.7 we can see some general statistics about the executions for each case. The mean number of resources used in each case decreases slightly with the increase of the number of ants. Also, all the tasks are covered in all experiments and assigned in the first iteration.

The time for each iteration increase as we increase the number of ants. However, for an increment in the number of ants in 250 %, we get just a 175 % increment in time (between case 1 and 2). Also, with a 500 % number of ants increment, we get just a 260 % time increment (between case 1 and 3). The iteration to reach the minimum number of resources decreases considerably with the number of ants too.

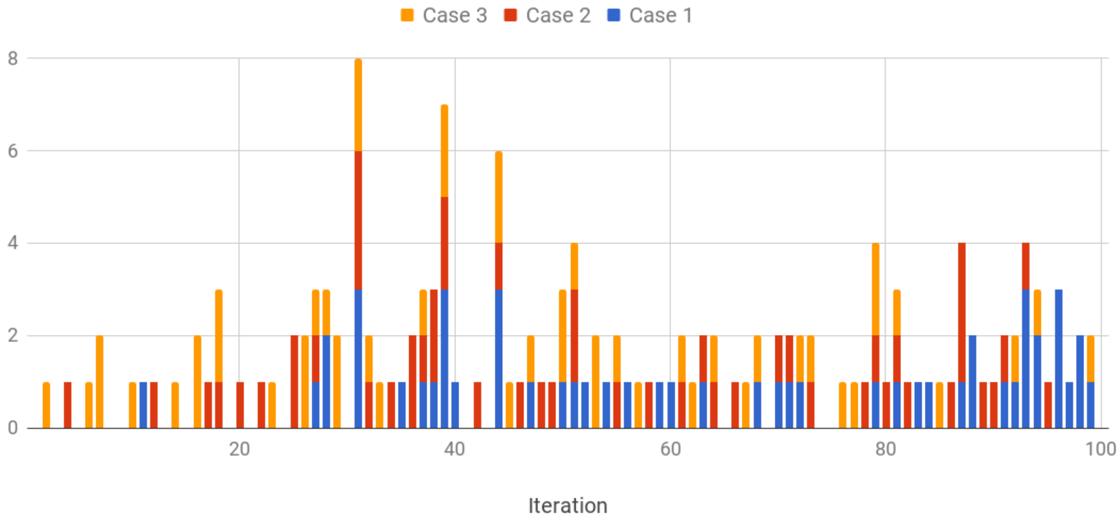

Figure 6.4: Number of executions to reach the optimal assignment in a certain iteration

In the Figure 6.4, we can observe the distribution of the number of executions to reach the best result in a certain iteration. We observe that this distribution is not very well defined but it tends to reach the best result between the twenty fifth and fiftieth iteration. There is a good number of experiments with bests results in the last twenty iterations, too.

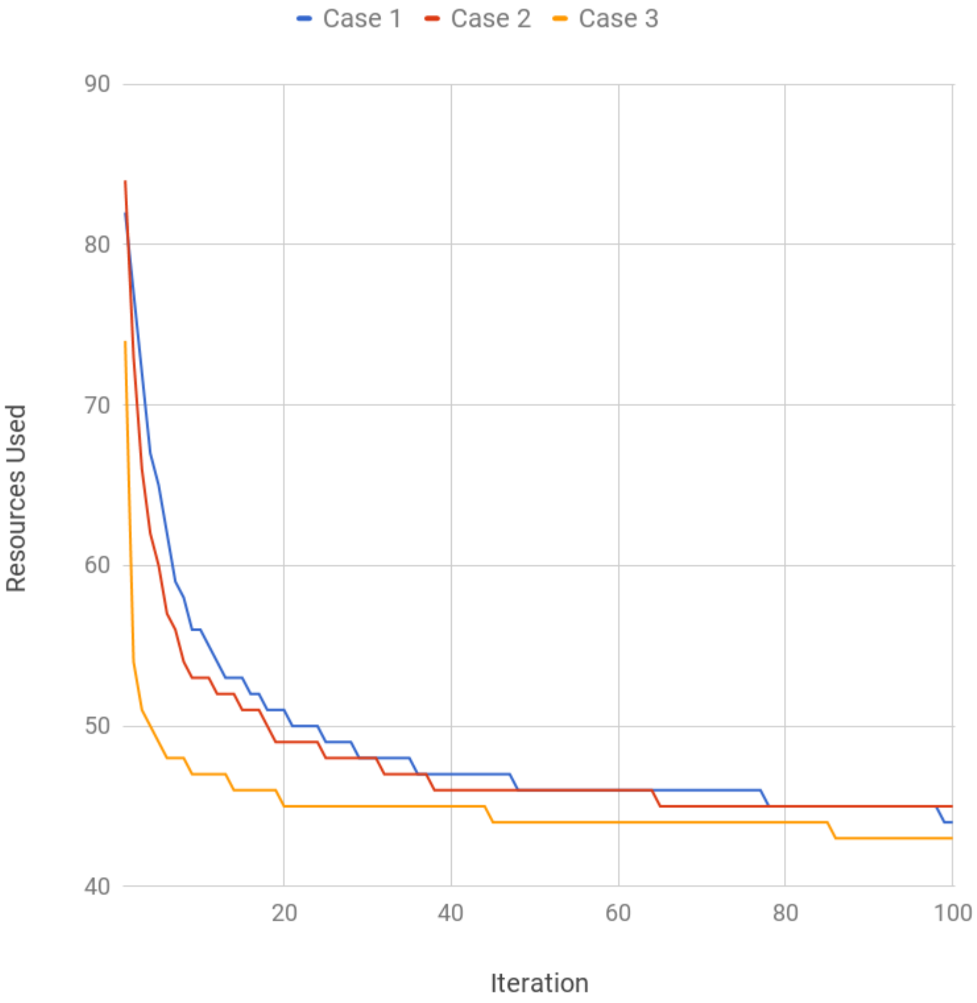

Figure 6.5: Mean number of resources used in each iteration

In the Figure 6.5, we can see how the number of resources used decreases with each iteration. In particular, it is observed how the number of resources used decreases as we increase the number of ants, as we said before. Also, with more amount of ants, the algorithm tends to converge faster, finding best results in early iterations.

6.2.3. Global results and comparative

We are going to compare the obtained results for each version of the algorithm. In the experiments executed, we can see two main differences between the sequential and the parallel version. These differences are: the number of resources used in each experiment and iteration, and the time for iteration and experiment.

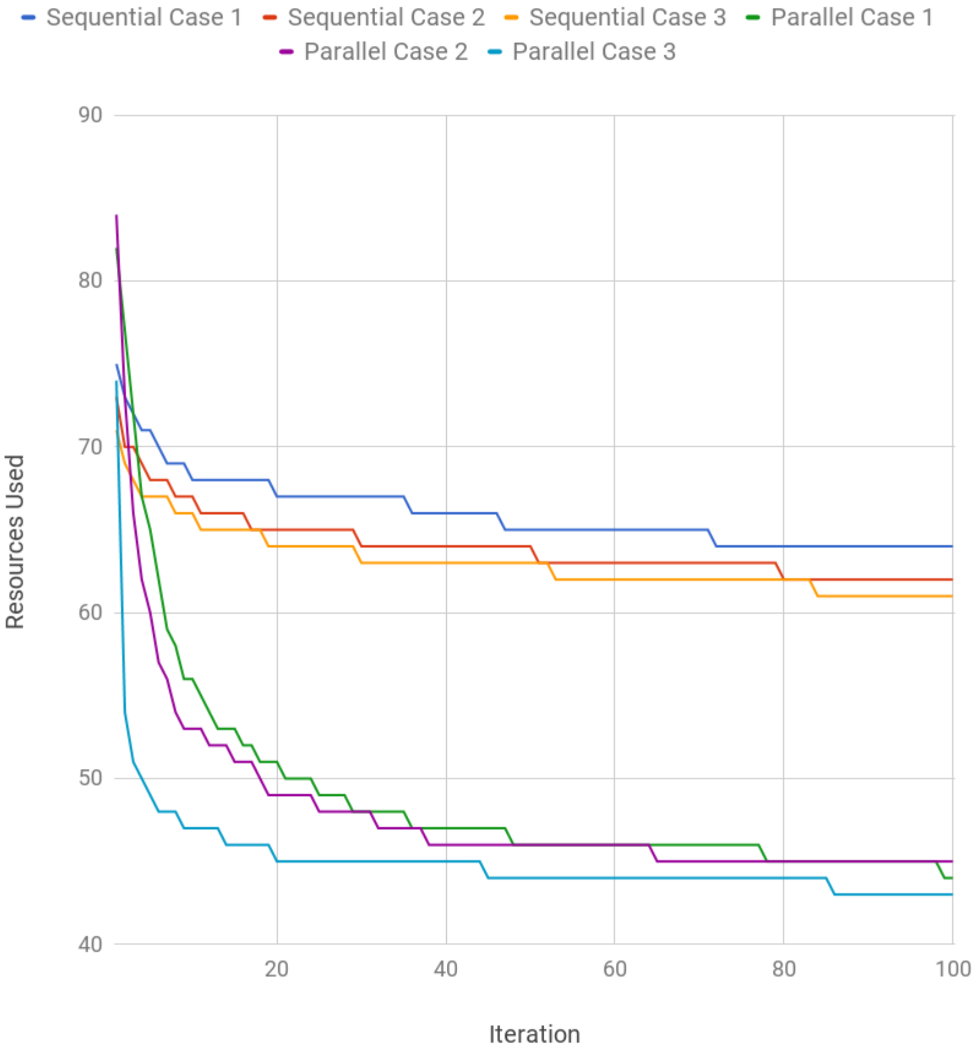

Figure 6.6: Mean number of resources used in each iteration

In the Figure 6.6, we can observe the difference of resources used between the two versions of the algorithm in all cases. In the parallel results, the algorithm finds better solutions in early iterations converging faster and finding better global results too. The best assignation finding by the sequential version in all cases used 57 resources. However, for the parallel version, we find a assignation with just 40.

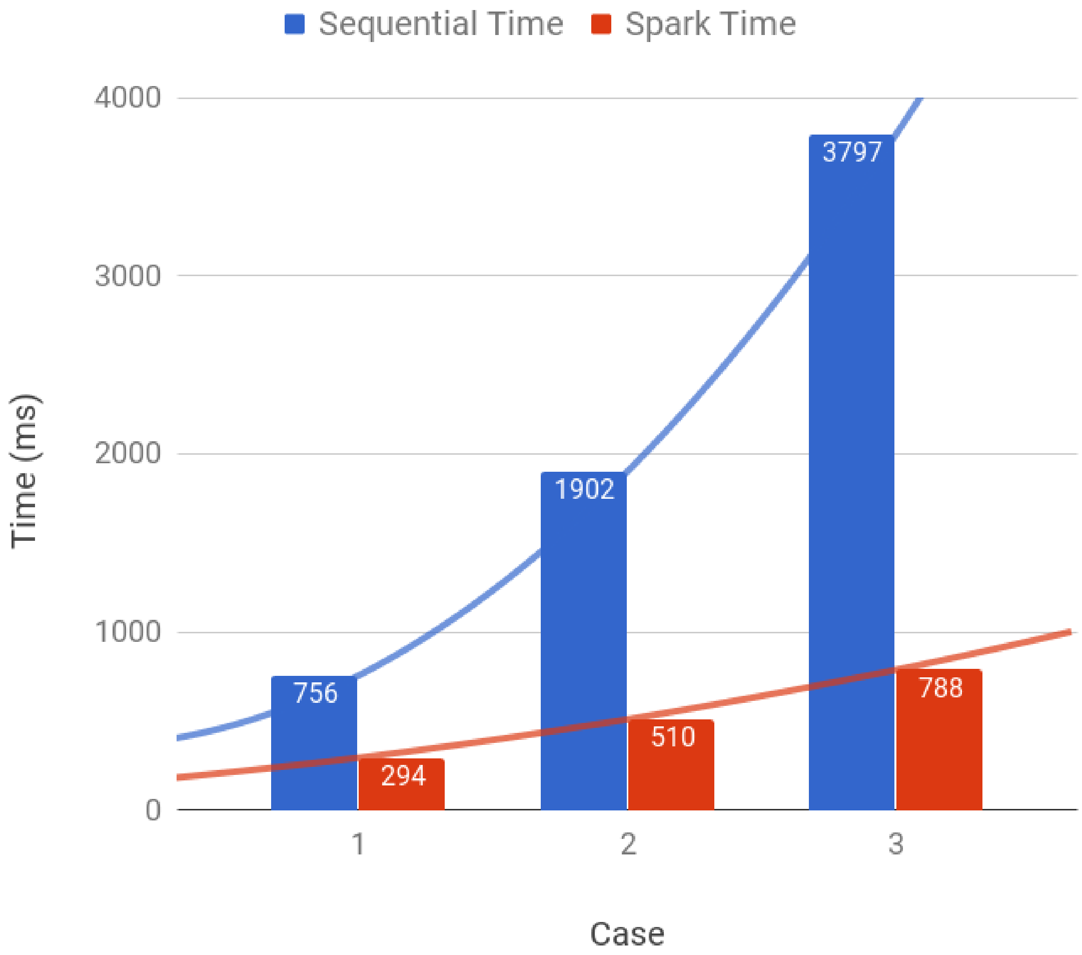

Figure 6.7: Mean Time for the experiments in each case

However, the difference in time between the two versions is even more noticeable. In the Figure 6.7, we can see how the increment of ants affects each version in all the cases. The parallel version is from the beginning faster than the sequential one. Moreover, the increment of ants has a significantly more impact in the sequential version than the parallel one. In addition, in this Figure, we can see that the slope for the function in the parallel version is actually more smoother than the sequential one.

Chapter 7

Conclusions and future work

7.1. Conclusions

We proposed a highly parallel and distributed solution to the resources assignment to task in the context of railway networks. We took the based work done in [Moreno Navas, 2015] and adapted it to work in the Spark parallel platform using its Map Reduce framework.

The work in [Moreno Navas, 2015] uses a combination between the Ant Colony Optimization Algorithm and the Constraint Programming motor Choco Solver. Because of the problems using the Choco Solver with Spark, we adapt our solution to be independent of this library and implement the constraint checking using the distance matrix of the ACO algorithm.

We design our algorithm using the capabilities of the Spark's.Our algorithm executes all the ants $solve()$ method in parallel and then get the best assignments sorting the results. Then we update the pheromone matrix with all the solutions obtained.

7.1.1. Experiments Analysis

The obtained results in the experiments with the parallel algorithm reveal a substantial improvement of time with the sequential version. Moreover, we prove that the behaviour of the parallel algorithm is far more superior than the sequential one when both the input data or parameters grows. In particular, as we increase the number of ants, we can see how the parallel version increases the time to complete an iteration more slowly as the sequential one. Using more ants give in all cases a better results in the assignation but they consume more time. This makes our algorithm more suitable in every situation for this assignment problem, but specially when the size of the problem grows, which affects positively the scalability of the system proposed.

The experiments also reveals that the quality of the solutions found in the parallel version were better than the sequential one. This was an unexpected result. The method of the algorithm to find the solutions is almost the same between each version. The only difference is in how the $solve()$ function for each ant is called: in parallel and distributed it is called simultaneously between the workers in Spark, and sequentially in the other case. But the insights of the $solve()$ function as well as the pheromone update and the distance matrix, is the same in each version. The reasons for these results are not determined. The principal hypothesis is that the exploration which the ants takes randomly was working different in the two versions. We think that the computation of the random values used to work out the probability of exploration is running different over Spark.

7.1.2. Downsides and considerations using Apache Spark

Apache Spark is not an easy platform to manage and could lead to some problems which can influence significantly in the algorithm performance.

Number of partitions

The first problem we came across is the number of partitions of an RDD. This number is the slices in which spark is going to distribute the RDD between workers. Usually Spark works with a big amount of data in the RDD which is distributed between the workers.

When the data in a RDD is small, a big number of partitions makes the workers spend too much time in communication and waiting for others. But if the RDD is big, a small number of partitions overload the workers making them to perform slower. Is important to be aware of the amount of data which is going to be distributed in order to set correctly the number of partitions.

Communication Bottleneck

Another problem in Spark is the bottle neck caused between communication between nodes and workers. In our experiments, we have just a single node (machine) with 8 workers (8 core processor). This setup eliminates the possible communication problems because all the data is encapsulated in a single node. However, a bigger Spark cluster with more nodes, may slow down the performance because the communication between them. This problem is important in our algorithm because the ants RDD has a reference to the pheromone matrix, not a copy. This means that when ant requires the information store in the pheromone matrix it maybe not be stored in its same node.

Energy impact

With using Spark makes the algorithm more efficient in terms of time and assignments results, but there is no efficient in terms of resources used and energy consumption. As we see in the figure 5.2, deploying Spark takes a lot of computing power from the CPU which also translates in a big energy impact. This fact makes Spark more expensive in energy, and therefore in money too.

Developing a distributed application in Spark could bring some benefits to the performance. However we have to be aware that this performance improvement may not be compensated by the energy and money impact of the deployment.

7.1.3. Hyperparameter optimization importance

When we applied algorithms as ACO, other meta-heuristics, or machine learning algorithms, it crucial to spend time in the hyperparameter optimization process. We defined the hyperparameters used for the executions in the Section 4.2.5.

As we performed the experiments, we tried to debug and learn the function of the hyperparameters in the algorithm. Specially the parameters α (the influence of the pheromones), β (the influence of the distance), ρ (the evaporation coefficient) and Q (the pheromone constant) are very important.

The whole algorithm is based in the combination of the pheromones and distances to find optimal solutions. If this values are not set correctly, the solutions are easily stuck in early iterations. Understanding correctly the hyperparametrs and how they interact in the algorithm makes the difference in the quality of the solutions finding by the ants

7.2. Social and Environmental Impact and Ethical and Professional Responsability

As we stated in previous sections, the power consumption of the parallel algorithm is higher than the sequential one. We have to note that deploying this as a commercial application in a big cluster to solve bigger problems could lead in a very demanding power consumption. However, the environmental impact of this Project alone is non-existent because the computing application resulting doesn't involves the use of high consumption machines. It is a well-known that high energy consumption is associated with Data Centers. This Project doesn't make any use of them. With all of the above, we can state that the our application doesn't' have any important environmental impact.

With regard to the social impact, the development of this Project has involved a first exploration phase to address a set of challenges related with the migration of the sequential algorithm to the parallel one and migrates it to the a Big Data computing platform. Since our project requires a couple of software process iterations to be considered a commercial software project, we consider that the social impact only is related with the preliminary results obtained that might allow a future development with interesting results to attack real problems in several domains of application.

Finally, with regard the Ethical and Professional Responsibility, we state that the development of this Project is not related with no one factor that contravenes with these two aspects. By the other hand, we state that this Project was developed with the highest respect for the pursuit of the profession and therefore have been considered both Ethics and Professional Responsibility in a personal way.

7.3. Future work

The investigation using parallel meta-heuristics, specifically in the distributed ACO algortihm, could be continued in several ways. We propose the following line for future developments and investigations:

- Perform a scalability study over the Parallel ACO algorithm with a bigger dataset. We tested the scalability of the algorithm increasing the number of ants in our experiments. However, we couldn't test the impact of bigger datasets in this problem because the lack of those.

- Improve the parallel ACO algorithm with a parallel version of the pheromone matrix. We noticed in our experiments that a great amount of time is required to update the pheromone matrix in each iteration. A parallel implementation of the pheromone matrix and the pheromone update could be a great improvement in the performance of the algorithm.

- Perform test in a bigger Spark cluster. As we said in the Section 7.1.2, our algorithm could performs different in a cluster with more nodes because the impact of the communication between nodes. This communication could become a bottle neck in the algorithm performance. Also, the energy impact of a Spark cluster could be too big in comparison with the performance increase.

- Implement a parallel solution using a CP motor like Choco Solver. As it was proved in [Moreno Navas, 2015], the combination of the ACO meta-heuristic and a CP motor can improve the quality of the obtained results. Combining ACO with Spark and Choco Solver or another CP library, could improve the assignations obtained. The integration of Choco Solver with Spark or any other parallel framework is a really complex matter, and as we see in this project, maybe is impossible because the internal architecture of the library.

- The development of a specific library for CP which runs over Spark. As we prove in our work, a parallel version of the ACO algorithm improves the overall performance of the algorithm. The same maybe can be applied to a CP library. With a parallel version of a CP library, we can improve the execution times for the problems defined using Constraint programming.

Annex

	Tren 1		Tren 2		Tren 3		Tren 4		Tren 5		Tren 6
08:05:00	CRDFCEN	08:19:00	SOTON	08:48:00	BRSTLTM	09:08:00	SLSBRY	09:13:00	CRDFCEN	10:08:00	CRDFCEN
08:23:00	NWPTRTG	08:40:00	FAREHAM	08:55:00	FILTNEW	09:30:00	WRMNSTR	09:27:00	NWPTRTG	10:22:00	NWPTRTG
08:41:00	SEVTNLJ	08:48:00	COSHAM	09:08:00	SEVTNLJ	09:40:00	WSTBRYW	09:45:00	SEVTNLJ	10:39:00	SEVTNLJ
08:56:00	FILTNEW	08:57:00	FRATTON	09:26:00	NWPTRTG	09:46:00	TRWBRDG	10:02:00	FILTNEW	10:54:00	FILTNEW
09:10:00	BRSTLTM	09:00:00	PSEA	09:41:00	CRDFCEN	09:52:00	BRDFDOA	10:14:00	BRSTLTM	11:09:00	BRSTLTM
09:17:00	KEYNSHM	09:04:00	PHBR			10:06:00	BATHSPA	10:27:00	BATHSPA	11:16:00	KEYNSHM
09:24:00	OLDFLDP	09:13:00	BRHM			10:09:00	OLDFLDP	10:37:00	FRESHFD	11:23:00	OLDFLDP
09:28:00	BATHSPA	09:30:00	WORTHNG			10:16:00	KEYNSHM	10:39:00	AVNCLFF	11:27:00	BATHSPA
09:40:00	BRDFDOA	09:37:00	SHRHMBS			10:28:00	BRSTLTM	10:43:00	BRDFDOA	11:39:00	BRDFDOA
09:47:00	TRWBRDG	09:43:00	HOVE			10:36:00	FILTNEW	10:50:00	TRWBRDG	11:47:00	TRWBRDG
09:59:00	WSTBRYW	09:47:00	BRGHTN			10:49:00	SEVTNLJ	11:01:00	WSTBRYW	12:01:00	WSTBRYW
10:08:00	WRMNSTR					11:03:00	NWPTRTG	11:09:00	WRMNSTR	12:10:00	WRMNSTR
10:30:00	SLSBRY					11:18:00	CRDFCEN	11:30:00	SLSBRY	12:32:00	SLSBRY
10:48:00	ROMSEY					11:27:00	FILTNEW	11:48:00	ROMSEY	12:50:00	ROMSEY
11:02:00	SOTON					11:40:00	SEVTNLJ	12:01:00	SOTON	13:02:00	SOTON
11:25:00	FAREHAM					11:59:00	NWPTRTG	12:25:00	FAREHAM	13:26:00	FAREHAM
11:33:00	COSHAM					12:14:00	CRDFCEN	12:33:00	COSHAM	13:34:00	COSHAM
11:40:00	FRATTON							12:40:00	FRATTON	13:43:00	HAVANT
11:43:00	PSEA							12:43:00	PSEA	13:54:00	CHCHSTR
11:49:00	PHBR							12:48:00	PHBR	14:07:00	BRHM
										14:27:00	WORTHNG
										14:34:00	SHRHMBS
										14:46:00	HOVE
										14:50:00	BRGHTN

Figure 1: Horarios trenes 1-6

	Tren 7		Tren 8		Tren 9		Tren 10		Tren 11		Tren 12
11:08:00	CRDFCEN	11:08:00	SLSBRY	11:10:00	SLSBRY	12:08:00	CRDFCEN	12:48:00	BRSTLTM	13:08:00	CRDFCEN
11:22:00	NWPTRTG	11:29:00	WRMNSTR	11:31:00	WRMNSTR	12:22:00	NWPTRTG	12:55:00	FILTNEW	13:23:00	NWPTRTG
11:39:00	SEVTNLJ	11:37:00	WSTBRYW	11:40:00	WSTBRYW	12:39:00	SEVTNLJ	13:08:00	SEVTNLJ	13:34:00	SEVTNLJ
11:54:00	FILTNEW	11:43:00	TRWBRDG	11:46:00	TRWBRDG	12:56:00	FILTNEW	13:26:00	NWPTRTG	13:49:00	FILTNEW
12:10:00	BRSTLTM	11:49:00	BRDFDOA	11:52:00	BRDFDOA	13:10:00	BRSTLTM	13:41:00	CRDFCEN	14:15:00	BRSTLTM
12:23:00	BATHSPA	12:04:00	BATHSPA	11:55:00	AVNCLFF	13:17:00	KEYNSHM			14:29:00	BATHSPA
12:33:00	FRESHFD	12:07:00	OLDFLDP	11:58:00	FRESHFD	13:24:00	OLDFLDP			14:38:00	FRESHFD
12:35:00	AVNCLFF	12:14:00	KEYNSHM	12:10:00	BATHSPA	13:28:00	BATHSPA			14:41:00	AVNCLFF
12:39:00	BRDFDOA	12:29:00	BRSTLTM	12:13:00	OLDFLDP	13:40:00	BRDFDOA			14:45:00	BRDFDOA
12:46:00	TRWBRDG	12:37:00	FILTNEW	12:20:00	KEYNSHM	13:48:00	TRWBRDG			14:51:00	TRWBRDG
13:01:00	WSTBRYW	12:50:00	SEVTNLJ	12:32:00	BRSTLTM	14:03:00	WSTBRYW			15:02:00	WSTBRYW
13:09:00	WRMNSTR	13:08:00	NWPTRTG	12:39:00	FILTNEW	14:12:00	WRMNSTR			15:10:00	WRMNSTR
13:30:00	SLSBRY	13:23:00	CRDFCEN	12:52:00	SEVTNLJ	14:34:00	SLSBRY			15:31:00	SLSBRY
13:48:00	ROMSEY	13:31:00	FILTNEW	13:10:00	NWPTRTG	14:52:00	ROMSEY			15:49:00	ROMSEY
14:01:00	SOTON	13:44:00	SEVTNLJ	13:25:00	CRDFCEN	15:04:00	SOTON			16:03:00	SOTON
14:24:00	FAREHAM	14:02:00	NWPTRTG	13:37:00	BATHSPA	15:26:00	FAREHAM			16:26:00	FAREHAM
14:32:00	COSHAM	14:17:00	CRDFCEN	13:40:00	OLDFLDP	15:36:00	COSHAM			16:34:00	COSHAM
14:39:00	FRATTON			13:47:00	KEYNSHM	15:43:00	HAVANT			16:41:00	FRATTON
14:42:00	PSEA			13:59:00	BRSTLTM	15:56:00	CHCHSTR			16:45:00	PSEA
14:48:00	PHBR			14:06:00	FILTNEW	16:09:00	BRHM			16:49:00	PHBR
				14:19:00	SEVTNLJ	16:29:00	WORTHNG				
				14:37:00	NWPTRTG	16:36:00	SHRHMBS				
				14:52:00	CRDFCEN	16:48:00	HOVE				
						16:52:00	BRGHTN				

Figure 2: Horarios trenes 7-12

	Tren 13		Tren 14		Tren 15		Tren 16		Tren 17		Tren 18
13:08:00	SLSBRY	14:08:00	SLSBRY	14:10:00	CRDFCEN	15:08:00	CRDFCEN	15:08:00	SLSBRY	15:46:00	WSTBRYW
13:29:00	WRMNSTR	14:29:00	WRMNSTR	14:24:00	NWPTRTG	15:22:00	NWPTRTG	15:29:00	WRMNSTR	15:52:00	TRWBRDG
13:43:00	WSTBRYW	14:45:00	WSTBRYW	14:41:00	SEVTNLJ	15:39:00	SEVTNLJ	15:41:00	WSTBRYW	15:58:00	BRDFDOA
13:49:00	TRWBRDG	14:51:00	TRWBRDG	14:56:00	FILTNEW	15:56:00	FILTNEW	15:47:00	TRWBRDG	16:12:00	BATHSPA
13:55:00	BRDFDOA	14:57:00	BRDFDOA	15:10:00	BRSTLTM	16:14:00	BRSTLTM	15:53:00	BRDFDOA	16:15:00	OLDFLDP
14:09:00	BATHSPA	15:11:00	BATHSPA	15:17:00	KEYNSHM	16:27:00	BATHSPA	16:07:00	BATHSPA	16:22:00	KEYNSHM
14:29:00	BRSTLTM	15:29:00	BRSTLTM	15:25:00	OLDFLDP	16:37:00	FRESHFD	16:10:00	OLDFLDP	16:30:00	BRSTLTM
14:36:00	FILTNEW	15:36:00	FILTNEW	15:29:00	BATHSPA	16:39:00	AVNCLFF	16:17:00	KEYNSHM	16:37:00	COSHAM
14:49:00	SEVTNLJ	15:52:00	SEVTNLJ	15:41:00	BRDFDOA	16:43:00	BRDFDOA	16:29:00	BRSTLTM	16:45:00	FAREHAM
15:07:00	NWPTRTG	16:10:00	NWPTRTG	15:47:00	TRWBRDG	16:50:00	TRWBRDG	16:36:00	FILTNEW	17:08:00	SOTON
15:22:00	CRDFCEN	16:25:00	CRDFCEN	16:05:00	WSTBRYW	17:01:00	WSTBRYW	16:49:00	SEVTNLJ	17:19:00	ROMSEY
15:29:00	FILTNEW	16:32:00	FILTNEW	16:14:00	WRMNSTR	17:10:00	WRMNSTR	17:07:00	NWPTRTG	17:41:00	SLSBRY
15:42:00	SEVTNLJ	16:48:00	SEVTNLJ	16:34:00	SLSBRY	17:32:00	SLSBRY	17:24:00	CRDFCEN	18:01:00	WRMNSTR
16:00:00	NWPTRTG	17:06:00	NWPTRTG	16:52:00	ROMSEY	17:50:00	ROMSEY	17:31:00	FILTNEW	18:12:00	WSTBRYW
16:15:00	CRDFCEN	17:21:00	CRDFCEN	17:05:00	SOTON	18:03:00	SOTON	17:44:00	SEVTNLJ	18:18:00	TRWBRDG
				17:27:00	FAREHAM	18:27:00	FAREHAM	18:02:00	NWPTRTG	18:24:00	BRDFDOA
				17:35:00	COSHAM	18:35:00	COSHAM	18:19:00	CRDFCEN	18:38:00	BATHSPA
				17:43:00	FRATTON	18:42:00	FRATTON			18:41:00	OLDFLDP
				17:46:00	PSEA	18:45:00	PSEA			18:48:00	KEYNSHM
				17:50:00	PHBR	18:50:00	PHBR			18:56:00	BRSTLTM

Figure 3: Horarios trenes 13-18

	Tren 19		Tren 20		Tren 21		Tren 22		Tren 23		Tren: 24
16:08:00	CRDFCEN	16:08:00	SLSBRY	16:35:00	CRDFCEN	17:08:00	CRDFCEN	17:08:00	SLSBRY	17:40:00	CRDFCEN
16:22:00	NWPTRTG	16:29:00	WRMNSTR	16:49:00	NWPTRTG	17:22:00	NWPTRTG	17:29:00	WRMNSTR	17:54:00	NWPTRTG
16:38:00	SEVTNLJ	16:42:00	WSTBRYW	16:49:00	SEVTNLJ	17:39:00	SEVTNLJ	17:42:00	WSTBRYW	17:54:00	SEVTNLJ
16:53:00	FILTNEW	16:48:00	TRWBRDG	17:19:00	FILTNEW	17:54:00	FILTNEW	17:48:00	TRWBRDG	18:25:00	FILTNEW
17:10:00	BRSTLTM	16:54:00	BRDFDOA	17:40:00	BRSTLTM	18:09:00	BRSTLTM	17:54:00	BRDFDOA	18:50:00	BRSTLTM
17:17:00	KEYNSHM	16:57:00	AVNCLFF	17:53:00	BATHSPA	18:16:00	KEYNSHM	18:08:00	BATHSPA	19:03:00	BATHSPA
17:24:00	OLDFLDP	17:00:00	FRESHFD	18:05:00	BRDFDOA	18:23:00	OLDFLDP	18:29:00	BRSTLTM	19:15:00	BRDFDOA
17:28:00	BATHSPA	17:12:00	BATHSPA	18:12:00	TRWBRDG	18:27:00	BATHSPA	18:36:00	FILTNEW	19:22:00	TRWBRDG
17:40:00	BRDFDOA	17:29:00	BRSTLTM	18:19:00	WSTBRYW	18:39:00	BRDFDOA	18:52:00	SEVTNLJ	19:29:00	WSTBRYW
17:47:00	TRWBRDG	17:36:00	FILTNEW	18:28:00	WRMNSTR	18:46:00	TRWBRDG	19:10:00	NWPTRTG	19:37:00	WRMNSTR
18:01:00	WSTBRYW	17:49:00	SEVTNLJ	18:55:00	SLSBRY	19:01:00	WSTBRYW	19:26:00	CRDFCEN	19:59:00	SLSBRY
18:11:00	WRMNSTR	18:07:00	NWPTRTG	19:13:00	ROMSEY	19:09:00	WRMNSTR	19:33:00	FILTNEW	20:17:00	ROMSEY
18:32:00	SLSBRY	18:22:00	CRDFCEN	19:25:00	SOTON	19:30:00	SLSBRY	19:49:00	SEVTNLJ	20:29:00	SOTON
18:50:00	ROMSEY	18:29:00	FILTNEW	19:48:00	FAREHAM	19:49:00	ROMSEY	20:07:00	NWPTRTG	20:53:00	FAREHAM
19:03:00	SOTON	18:42:00	SEVTNLJ	19:57:00	COSHAM	20:03:00	SOTON	20:23:00	CRDFCEN	20:53:00	COSHAM
19:27:00	FAREHAM	19:00:00	NWPTRTG	20:09:00	HAVANT	20:26:00	FAREHAM			21:08:00	FRATTON
19:35:00	COSHAM	19:15:00	CRDFCEN	20:20:00	CHCHSTR	20:26:00	COSHAM			21:13:00	PSEA
19:42:00	FRATTON			20:28:00	BRHM	20:41:00	FRATTON			21:22:00	PHBR
19:45:00	PSEA			20:45:00	WORTHNG	20:45:00	PSEA				
19:50:00	PHBR			20:52:00	SHRHMBS	20:48:00	PHBR				
				20:59:00	HOVE						
				21:03:00	BRGHTN						

Figure 4: Horarios trenes 19-24

	Tren 25		Tren 26		Tren 27		Tren 28		Tren 29		Tren 30
17:46:00	WSTBRYW	18:08:00	CRDFCEN	18:08:00	SLSBRY	19:08:00	SLSBRY	19:10:00	CRDFCEN	20:08:00	SLSBRY
17:52:00	TRWBRDG	18:22:00	NWPTRTG	18:31:00	WRMNSTR	19:28:00	WRMNSTR	19:24:00	NWPTRTG	20:29:00	WRMNSTR
17:58:00	BRDFDOA	18:39:00	SEVTNLJ	18:44:00	WSTBRYW	19:43:00	WSTBRYW	19:42:00	SEVTNLJ	20:42:00	WSTBRYW
18:01:00	AVNCLFF	18:56:00	FILTNEW	18:50:00	TRWBRDG	19:49:00	TRWBRDG	19:57:00	FILTNEW	20:48:00	TRWBRDG
18:04:00	FRESHFD	19:12:00	BRSTLTM	18:56:00	BRDFDOA	19:55:00	BRDFDOA	20:15:00	BRSTLTM	20:54:00	BRDFDOA
18:18:00	BATHSPA	19:19:00	KEYNSHM	19:10:00	BATHSPA	20:09:00	BATHSPA	20:28:00	BATHSPA	21:08:00	BATHSPA
18:20:00	OLDFLDP	19:26:00	OLDFLDP	19:31:00	BRSTLTM	20:31:00	BRSTLTM	20:38:00	FRESHFD	21:29:00	BRSTLTM
18:27:00	KEYNSHM	19:30:00	BATHSPA	19:38:00	FILTNEW	20:38:00	FILTNEW	20:40:00	AVNCLFF	21:36:00	FILTNEW
18:35:00	BRSTLTM	19:42:00	BRDFDOA	19:51:00	SEVTNLJ	20:54:00	SEVTNLJ	20:44:00	BRDFDOA	21:49:00	SEVTNLJ
18:42:00	COSHAM	19:49:00	TRWBRDG	20:09:00	NWPTRTG	21:09:00	NWPTRTG	20:51:00	TRWBRDG	22:07:00	NWPTRTG
18:50:00	FAREHAM	19:59:00	WSTBRYW	20:26:00	CRDFCEN	21:28:00	CRDFCEN	21:01:00	WSTBRYW	22:26:00	CRDFCEN
19:13:00	SOTON	20:08:00	WRMNSTR	20:33:00	FILTNEW	21:35:00	FILTNEW	21:09:00	WRMNSTR	22:33:00	FILTNEW
19:25:00	ROMSEY	20:31:00	SLSBRY	20:46:00	SEVTNLJ	21:51:00	SEVTNLJ	21:31:00	SLSBRY	22:46:00	SEVTNLJ
19:46:00	SLSBRY	20:50:00	ROMSEY	21:04:00	NWPTRTG	22:06:00	NWPTRTG	21:51:00	ROMSEY	23:04:00	NWPTRTG
20:06:00	WRMNSTR	21:03:00	SOTON	21:21:00	CRDFCEN	22:25:00	CRDFCEN	22:04:00	SOTON	23:23:00	CRDFCEN
20:24:00	WSTBRYW	21:26:00	FAREHAM					22:27:00	FAREHAM		
20:30:00	TRWBRDG	21:26:00	COSHAM					22:27:00	COSHAM		
20:36:00	BRDFDOA	21:40:00	FRATTON					22:40:00	FRATTON		
20:39:00	AVNCLFF	21:43:00	PSEA					22:44:00	PSEA		
20:42:00	FRESHFD	21:48:00	PHBR					22:49:00	PHBR		
20:54:00	BATHSPA										
20:57:00	OLDFLDP										
21:04:00	KEYNSHM										
21:12:00	BRSTLTM										

Figure 5: Horarios trenes 25-30

	Tren 31		Tren 32		Tren 34
20:18:00	CRDFCEN	21:46:00	BRGHTN	22:05:00	PHBR
20:32:00	NWPTRTG	21:50:00	HOVE	22:12:00	PSEA
20:49:00	SEVTNLJ	21:56:00	SHRHMBS	22:16:00	FRATTON
21:04:00	FILTNEW	22:03:00	WORTHNG	22:16:00	COSHAM
21:25:00	BRSTLTM	22:18:00	BRHM	22:32:00	FAREHAM
21:38:00	BATHSPA	22:26:00	CHCHSTR	22:57:00	SOTON
21:50:00	BRDFDOA	22:47:00	HAVANT	23:09:00	ROMSEY
21:57:00	TRWBRDG	22:58:00	FRATTON	23:28:00	SLSBRY
22:05:00	WSTBRYW	23:01:00	PSEA	23:48:00	WRMNSTR
22:13:00	WRMNSTR	23:05:00	PHBR	23:56:00	WSTBRYW
22:34:00	SLSBRY				
22:52:00	ROMSEY				
23:06:00	SOTON				
23:28:00	FAREHAM				
23:28:00	COSHAM				
23:41:00	FRATTON				
23:45:00	PSEA				
23:48:00	PHBR				

Figure 6: Horarios trenes 31-33

BRGHTN	BRSTLTM	CRDFCEN	PHBR	ROMSEY	SLSBRY	SOTON	WSTBRYW
50	50	50	50	50	50	50	50

Figure 7: Resources and Base Stations

Bibliography

[Boy, 2017] (2017). Chapter 10 - {GPU} computing applied to linear and mixed-integer programming. In Sarbazi-Azad, H., editor, *Advances in {GPU} Research and Practice*, Emerging Trends in Computer Science and Applied Computing, pages 247 – 271. Morgan Kaufmann.

[Apache software, 2000] Apache software, F. (2000). Apache cassandra.

[Apache software, 2009] Apache software, F. (2009). Apache hadoop.

[Arora and Barak, 2007] Arora, S. and Barak, B. (2007). *Computational Complexity: A Modern Approach.* Princeton University, New York, NY, USA.

[Cahon et al., 2004] Cahon, S., Melab, N., and Talbi, E.-G. (2004). Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380.

[Conforti et al., 2014] Conforti, M., Cornuejols, G., and Zambelli, G. (2014). *Integer Programming.* Springer International Publishing.

[Dorigo et al., 1996] Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41.

[Glover, 1977] Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166.

[Glover, 1989] Glover, F. (1989). Tabu search part 1. *ORSA Journal on Computing*, 1(3):190–206.

[Goldberg and Holland, 1988] Goldberg, D. E. and Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99.

[Hashem et al., 2015] Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., and Khan, S. U. (2015). The rise of âœbig dataâ on cloud computing: Review and open research issues. *Information Systems*, 47:98 – 115.

[Institute, 2000] Institute, C. M. (2000). Millenium problems.

[Karau et al., 2015] Karau, H., Konwinski, A., Wendell, P., and Zaharia, M. (2015). *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, Inc., 1st edition.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

[Laney, 2001] Laney, D. (2001). 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group.

[Manfrin et al., 2006] Manfrin, M., Birattari, M., Stützle, T., and Dorigo, M. (2006). *Parallel Ant Colony Optimization for the Traveling Salesman Problem*, pages 224–234. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Martello and Toth, 1990] Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA.

[Mezmaz et al., 2011] Mezmaz, M., Melab, N., Kessaci, Y., Lee, Y., Talbi, E.-G., Zomaya, A., and Tuyttens, D. (2011). A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *Journal of Parallel and Distributed Computing*, 71(11):1497 – 1508.

[Moreno Navas, 2015] Moreno Navas, D. (2015). Maximum coverage ant colony optimizer.

[Papadimitriou, 1976] Papadimitriou, C. H. (1976). The np-completeness of the bandwidth minimization problem. *Computing*, 16(3):263–270.

[Parejo et al., 2012] Parejo, J. A., Ruiz-Cortés, A., Lozano, S., and Fernandez, P. (2012). Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing*, 16(3):527–561.

[Pérez and Caparrini, 2003] Pérez, M. and Caparrini, F. (2003). *Teoría de la Complejidad Computacional*. Secretariado de Publicaciones de la Universidad de Sevilla.

[Petit et al., 2016] Petit, O., Lapel, N., and and, S. P. (2016). Spark or.

[Prud'homme et al., 2016] Prud'homme, C., Fages, J.-G., and Lorca, X. (2016). *Choco Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S.

[Singh and Reddy, 2014] Singh, D. and Reddy, C. (2014). A survey on platforms for big data analytics. *Journal of Big Data*, 2(1):1–20.

[Stützle and Hoos, 2000] Stützle, T. and Hoos, H. H. (2000). Max-min ant system. *Future Gener. Comput. Syst.*, 16(9):889–914.

[Talbi, 2002] Talbi, E.-G. (2002). A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5):541–564.

[Voß, 2001] Voß, S. (2001). *Meta-heuristics: The State of the Art*, pages 1–23. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Wang et al., 2015] Wang, Z., Wang, H., Xing, H., and Li, T. (2015). Ant colony optimization algorithm based on spark. *Journal of Computer Applications*, 35(10):2777.