

Job sequencing and tool switching problem metaheuristic



Alejandro Miguel Piquer Robledo

Supervisor: Jean-Charles Delvenne

Lucas Van Wunnik (UPC)

Readers: Daniele Catanzaro

Christophe De Vleeschouwer

Université Catholique de Louvain

Acknowledgements

I would like to acknowledge all the people which directly or indirectly have taken part of this project. First of all to my family, which has supported me since I decided to come across my engineering studies six years ago. Rough times are not easy to deal with. Secondly but not less important, to one special person which I left in Barcelona in the beginning of this adventure abroad, my girlfriend. Your support has helped me to surpass all the difficult moments in this experience.

For the people in UCL framework I would like to thank the flexibility of my co-supervisor Jean-Charles Delvenne to adopt a topic not directly related with the EPL and to my supervisor Daniele Catanzaro for providing me the guidelines since the moment I arrived to Louvain-la-neuve. Without your help this project wouldn't be a reality. Last but not least I would like to acknowledge all the people who has helped me during my stage in Belgium, this project also has a part of you.

Abstract

During the following pages it will be presented a metaheuristic towards the solution of the *job Sequencing and tool Switching Problem* (SSP), a NP-hard combinatorial optimization problem arising from computer and manufacturing systems. Putting interest in previous works, the precedents in the literature will be studied in order to understand the scope of the solution development. Focusing on the current state of the art, an alternative path will be presented with the aim to achieve good quality solutions in a feasible time or even settle a new basis for future works. The metaheuristic will be chosen between two developed candidates, which will be defined in a theoretical way and later will be executed to prove the results. Through computational experiments it will be shown how the designed metaheuristics can achieve the best results known in the literature and even improve them in some of the cases. Analysing the results of each proposed metaheuristic one of them will be selected as the proper one for the SSP. After analysing the computational results it will be presented an informatics tool able to calculate the sequence and the tool distribution for the datasets studied in Laporte et al. (2004) [32].

Table of contents

List of figures	iv
List of tables	v
1 Introduction	1
2 Problem statement	3
2.1 Description of the problem	3
2.1.1 Definition	3
2.1.2 Assumptions	6
2.1.3 Notation	7
2.2 Computational complexity	8
2.2.1 Travelling salesman problem definition	8
2.2.2 NP-hardness results	8
2.2.3 Finding the minimum number of setups for a fixed job sequence	9
2.2.4 Computational conclusions	11
2.3 Integer linear programming formulations	11
2.3.1 Tang and Denardo's formulation	11
2.3.2 Laporte et al. formulation	12
2.3.3 Catanzaro et al. formulation	14
2.3.4 Conclusions of the integer linear programming formulations	15
2.4 Integer non-linear programming formulations	15
2.4.1 Bard formulation	16
2.4.2 Ghiani et al. formulation	16
2.4.3 Furrer & Mütze formulation	17
2.4.4 Conclusions of the integer non-linear programming formulations	18
2.5 Current state of the art	18
3 Metaheuristic definition	20
3.1 Introduction	20
3.2 Definition	20

3.3	Classification	22
3.4	Metaheuristic applied to the travelling salesman problem	23
3.5	Metaheuristic applied to SSP	24
3.5.1	Tabu Search definition	24
3.5.2	Application for the SSP	25
4	Metaheuristics for the SSP	26
4.1	Introduction to metaheuristic development	26
4.2	Greedy Metaheuristic development	26
4.2.1	Keep tool needed soonest algorithm	28
4.3	Improving towards the solution	29
4.3.1	Ant Colony Optimization definition	30
4.3.2	Ant colony optimization scheme	33
4.3.3	Ant Colony System	36
4.3.4	Local search for ACS	37
5	Computational results	39
5.1	Introduction	39
5.2	Random Search Procedure	40
5.3	GRASP results	41
5.4	ACS results	43
5.4.1	ACS behaviour	45
5.5	Comparing metaheuristics	49
5.5.1	Dataset A	50
5.5.2	Dataset B	50
5.5.3	Dataset C	51
5.5.4	Dataset D	52
5.6	Conclusions computational results	52
5.7	Comparison with current state of the art	54
5.8	Plotting solutions	55
5.9	Python coding	55
6	Conclusions	56
	Bibliography	58
	Appendix A Graph-theoric definitions	61
	Appendix B Instances definition	63
	Appendix C ACS python based solution	64

List of figures

4.1	Demonstration of ants behaviour	30
4.2	ACO algorithm scheme	33
4.3	Ant decision scheme	34
5.1	Pheromone map obtained with one iteration	46
5.2	Pheromone map obtained with five iterations	47
5.3	Pheromone map obtained with fifteen iterations	47
5.4	Pheromone map obtained with fifty iterations	48
5.5	Distribution of the tools for the σ sequence found	49
5.6	Obtained box plot for the dataset C	51
5.7	Obtained box plot for the dataset D	52

List of tables

2.1	Example of an instance of the SSP	4
2.2	Solution for the instance of Figure 1	4
2.3	Result by applying dominance rule	5
5.1	Random search simulations results for the different datasets	40
5.2	GRASP averages times for one iteration	42
5.3	GRASP simulations results for the different datasets	43
5.4	ACS simulations results for the different datasets	44
5.5	ACS simulations results with improvements	45
5.6	Results of simulations for Random search, GRASP and ACS	50
5.7	ACS results when extending time	53
5.8	Comparison with current state of the art	54
B.1	Definition of the used datasets	63

Chapter 1

Introduction

During the last few decades, the manufacturing systems and industrial logistics have suffered lots of changes in order to improve the results in the fields of processing and delivery times. Obtaining a product with quality may no longer add value and companies must try to decrease the cost in their different departments to get more gains selling a product more adapted to the market. This economy is ruled though the law of supply and companies don't have enough margin versus the competition to increase the prices to get more benefits. This point awards more importance to logistics and supply chain management. Sometimes the logistics in a company depends on a third party, and to improve the results companies must deal with the different suppliers to share thoughts and to select the right path. Even so, companies can work in their own process and optimal solutions can be reached using the available resources, without any investment. This fact has caused companies during the last couple decades to look for the optimization of their own processes to work nearly on the maximum capacity of the supply chain, improving the margins on the product, leading to higher earnings.

In a manufacturing system it happens often that to get the final product, the item must be proceed through several steps. Each step uses different tools to modify the shape and the product characteristics with the aim of getting a final product that can accomplish all the requirements. In some cases these steps or jobs can get interchange with each other, achieving higher values of adaptability in the manufacturing process. Nowadays the machines are more flexible, and the same line can proceed different products. The most well-known example is the car industry, which can produce different models of the brand on the same line, making the difference between the products in the tools used or the ways to use them. Due to the flexibility in some processes the order of the jobs can be modified and the involved tools can be changed in each step. For each job it will be mandatory to have the necessities tools but sometimes the tools required leave free space for more tools on the magazine. At the end, the efficient switching of the tools between each job can be translated into the reduction of the costs, which leads to the idea of designing an optimal solution for the jobs and the tools. In the literature of combinatorial problems, some authors found interesting this idea of minimizing tool switching costs, naming this kind of problem: the *job Sequencing and tool Switching Problem* (SSP). The SSP nowadays has solved using different strategies sequences up to 25 jobs through linear

programming, but for larger datasets it takes too much time to do in an exact mode. The results through integer non-linear programming also do not reach optimal values for all the sizes of instances. Additionally, the same time related issue is present for large dataset instances.

Due to the necessity of companies getting better results in their processes in a short time, developing a metaheuristic in a realistically defined time is a necessity to achieve an optimal distribution in the industrial processes. Analysing the precedent literature in the fields of the metaheuristic will be presented an alternative path to achieve good quality results in a feasible time. This alternative will need to achieve low deviations with the best solutions known and achieve them in a feasible amount of time. Through the development of the metaheuristic it will be tried to settle a basis for next steps, for instance including real data of a company with the aim to study the economic impact of an improvement in the tool switching. All the study presented in this project is based on the ideal case, the case described in the literature as the *job Sequencing and tool Switching Problem*.

Chapter 2

Problem statement

2.1 Description of the problem

2.1.1 Definition

The importance of tool management for the efficient use of automated manufacturing systems arises in the metal working industry in the numerically controlled (N/C) machines. Each machine performs operations by using the tools placed in the magazine. The capacity of the magazine is limited, and the machine needs to do all the operations in order to complete the task. The behaviour of this flexible machines has been described in publications as Browne et al. (1984) [10], providing a definition for the procedures based on jobs which need several tools. During the first steps of the numeric controlled machines the fact of interchanging was considered negligible in terms of time, but it has been proven that this assumption isn't correct. The experience in such different areas as the metalworking industry [3], the computer memory allocation [23, 37], pharmaceutical packaging [35] and mailroom insert planning [20] has shown that the amount of time for tool changes can't be negligible. The basis for this approach is following: tool changes times might be negligible but tools must undergo through a fine tuning process every time that a switch happens, and this takes a notorious amount of time relative to the job time. This implies a notorious quantity of time is being used without productivity; just to get the tool ready for the next job. By minimizing the tool switching between jobs we will be able to decrease the time wasted, and in turn, this amount of time would be used for more productive actions. Tool switching operations can be performed either in groups or one at a time, but the core of this project is focused in the idea of minimizing the total number of tools switches, not the number of times of tool switching, as analysed by Tang & Denardo (1988) [41]. The switching cost may be tool and/or application dependent as shown in Tang & Denardo (1988) [40], Crama et al. (1994) [13], Catanzaro et al. (2015) [11], Privault & Finke (2000) [37].

Although times have changed most part of the industries are working in the same way from twenty years to now. The tools have improved during this time, the machines are more accurate and these two main facts make us to realize the following question: Are we working in the right way? Improve

without investment is the first step to get more benefits, and the application of an optimal job sequence and tool switching will help companies increase margins. In this document has studied the case in which the order of the jobs isn't restricted, accordingly the job sequence won't be constrained as the real processes (e.g Job 4 must be done before the Job 6 and after the Job 2). This assumption awards the problem more flexibility and it will establish the basis for future works.

To simplify the calculation of the cost between each step of the sequence, the more basic (although the main) version of the switching cost will be utilized, characterized by being tool independent. This variant allows the ability to exchange different tools with independence, without considering what tools are changing. This problem is known in the literature as *job Sequencing and tool Switching Problem* (Crama et al. 1994 [13], Ghiani et al. 2010 [24], Laporte et al. 2004 [32]; Tang & Denardo 1988 [40]). An example of a in instance of the SSP problem is shown in the following table 2.1, first shown by Tang & Denardo:

Jobs	1	2	3	4	5	6	7	8	9	10
Tools	1	1	2	1	3	1	1	6	3	5
	4	3	6	5	5	2				7
	8	5	7	7	8	4				
	9		8	9						
Magazine capacity: 4										

Table 2.1 Example of an instance of the SSP

The first row in the table provides the labels of the jobs and the corresponding processing order; the last row provides the capacity of the magazine. Each column in the table refers to the set of tools needed for a specific job. The empty entries in each column indicates the free slots in the magazine. The magazine capacity in this example is four, so for every job, the tools required should be equal or less than the capacity. A feasible solution for the table 2.1 would be:

Jobs	8	1	6	4	2	5	10	3	9	7
Tools	④	4	4	5	5	5	5	2	3	1
	6	1	1	1	1	⑦	7	7	⑦	⑦
	⑧	8	2	7	3	3	⑥	6	⑥	⑥
	⑨	9	⑨	9	⑧	8	⑧	8	⑧	⑧
Magazine capacity: 4										

Table 2.2 Solution for the instance of Figure 1

The circled values inside the table represent the tools charged on the magazine which are not used during a specific job denoted by the first row of the column. This solution guides to a tool switch

value of 15. It can be seen how the tools needed by the jobs as 7, 8, 9 and 10 are part of other jobs, guiding to the idea of a dominance between jobs. If one job includes a whole set of tools which belong to a larger set of works for another job, they can be erased from the list of candidates. These jobs will be pre-processed as proven in Catanzaro et al. 2015 [11], where it's extracted the idea that the application of this dominance rule doesn't affect the result of the SSP. The remaining jobs once the dominance rule has been applied is:

Jobs	1	2	3	4	5	6
Tools	1	1	2	1	3	1
	4	3	6	5	5	2
	8	5	7	7	8	4
	9		8	9		
Magazine capacity: 4						

Table 2.3 Result by applying dominance rule

Tang & Denardo (1988) [40] also investigated the relationship between the SSP and the Travelling Salesman Problem (TSP) (Garey & Johnson 2003 [22]) reaching a relationship between them. An instance of the SSP degenerates into an instance of the TSP when the tool set of each job has a constant value and is equal to the capacity of the magazine and the cost remains constant and independent for the tools. Hybrid methods are using the principles of the TSP for the problem solution (Ahmadi et al. 2018 [1]). Tang & Denardo (1988)[40] denoted in their publication that the SSP can be decomposed into two nested subproblems:

1. Sequencing problem (SP), consisting of identifying an optimal job processing sequence.
2. Tooling problem (TP), for a given job processing sequence, what tools should be loaded in the tool magazine at each position of the sequence in order to minimize the total number of tools switches.

The authors showed that the TP can be solved in polynomial time but SP in contrast constitute the hardest part of the problem. Even though, most of the publications use the same strategy towards the TP and redefine the ones used for the SP.

2.1.2 Assumptions

Crama et al. (1994) [13] summarised the assumptions which must be taken for the resolution of the tool switching problem in the *job Sequencing and tool Switching Problem* framework:

1. All the operations that are required for each job are performed on the same machine. No tool sharing.
2. All the tools are kept in the tool storage area. Each machine has an automatic tool interchanging device that can switch tools between the magazine and the tools storage area.
3. The tool magazine is always fully loaded. It has been proven that doesn't affect to the generality of the model. Indeed, since no cost is incurred for tools staying in the magazine, one may consider that the first C tools to be used are all incorporated in the very first loading. Hence, a tool only needs to be removed when it is replaced by another one.
4. Each tool is assumed to fit in one slot of the magazine. This assumption allows to solve difficulties but restricts the existence of tools which for size can't have adjacent tools busy.
5. The capacity of the magazine can be greater than the maximum required tools in a job.
6. The time needed to remove and insert each tool is constant. This assumption will guide to the simplification of the problem. Otherwise the development of the problem should be adapted to be tool dependent.
7. Tools cannot be changed simultaneously. This assumption considers trying to group e.g. three changes of tools between two jobs is the same as doing one change in three different jobs.
8. The subset of tools required to carry out each job is fixed in advance. All the jobs are defined in advance and they won't change during the allocation process.
9. The list of jobs is completely known. This assumption is realistic in short-term planning.
10. Tools don't break nor wear out. The assumption is justified with the application of a short-term planning versus a longer life value for the tool.

All these assumptions will be adopted for the development of this project in order to keep the core of the SSP as close as possible to the precedent solutions. Establishing the same basis this project will be assumed as a new way to solve the SSP.

2.1.3 Notation

Consider a set $J = \{1, 2, \dots, N\}$ of N jobs to process in the flexible machine and define $T = \{1, 2, \dots, M\}$ as the set of M tools necessary to process the jobs in J . Hence, T_j will denote the set of tools necessary to carry out j , where $j \in J$; J_t as the set of tools using tool $t \in T$; and $C \geq \max_j \{|T_j|\}$ the maximum number of tools which can be stored in the magazine at any time. The data of the jobs and the tools needed for each job will be denoted with the matrix A , with $M \times N$ dimension. a_{ji} will take the value of 1 if the tool i is required for by the job j and 0 in case of no necessity. The job sequence will be denoted as σ , composed by natural numbers from 1 to N , placed in an order to optimize the tool switches. The optimal solution for the tools placed in each job will be denoted by the matrix W , with $M \times N$ dimension. Thus, each element of σ will be placed in a column of W and each row will indicate the tools charged in the magazine for each step. Let w_{mn} where $m = 1, \dots, M$ and $n = 1, \dots, N$ be a binary variable which will indicate the use of tool n in the processing order m .

There are two types of decision variables, as described in Tang & Denardo (1988) [40]:

- Sequence or order in which the jobs will be processed. This will be stored in σ .
- Set of tools placed on the machine before each job is processed. Defined by the variable W .

For the example described in the table 2.2, before applying the dominance rule, the definition of the variables would be:

$$\begin{aligned}
 J &= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \\
 T &= \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
 C &\geq 4 \\
 \sigma &= [8, 1, 6, 4, 2, 5, 10, 3, 9, 7] \\
 W &= \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}
 \end{aligned}$$

The columns of W indicate the tools needed for each job, following the sequence known by σ . Each row indicates the use of a tool, where w_{m1} will take the value of 1 if the tool m is used for job 1.

The notation explained in this chapter will be the same during the whole document, due to the fact that the SSP will always contain as a solution a σ sequence and a W matrix which will contain the distribution of the tools in each job.

2.2 Computational complexity

Crama et al. (1994) [13] analysed the tool switching problem complexity. During this section the main conclusions will be described in order to concern about the computational complexity of the SSP. For the reader the following information may be recalled: a problem is considered NP-hard if it is at least as hard as the travelling salesman problem. The definition of the travelling salesman problem is shared in the subsection 2.2.1.

2.2.1 Travelling salesman problem definition

The travelling salesman problem was first formulated in 1930 and it is based on the following question: "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" . The traveller needs to visit all the cities and has to optimize the path to get the minimum value of the route costs, denoted in the definition of the problem as the distance between each pair of cities. Lin & Kernighan (1973) [34], published an algorithm for the solution of this kind of problem, basing the solution in the graph formulation. This problem established the basis for the definition of the limits of the NP-hard. As shared in section 2.2.2, the SSP is considered NP-hard due to the complexity of the solution.

2.2.2 NP-hardness results

Tang & Denardo in the publication of 1988 claim that the tool switching problem is NP-hard. They didn't present a formal proof of this assertion, but rather inferred it from the observations of the similarities between the SSP and the TSP. Crama et al.(1994) [13] supported Tang & Denardo claim presenting the following examples to proof the NP-hardness complexity:

Input: A $M \times N$ matrix A and a C capacity.

Problem 1: Find a job sequence for A minimizing setups required on a machine with capacity C .

Proof: Let $G = (V, E, d)$ be a graph and $H = (E, I, \delta)$ be its edge-graph¹. Let assume the problem of finding a minimal-length travelling salesman path in H (problem 2 in the appendix A). The job scheduling and tool switching problem is going to be proved as NP-hard problem with the comparison as a special case for any $C \geq 2$. For simplicity it will be only explained the for $C = 2$.

Let $V = \{1, 2, \dots, M\}$ and $E = \{e_1, e_2, \dots, e_N\}$. Define an $M \times N$ matrix A , with rows associated to the nodes of G , columns associated to the edges of G , and such that:

$$\begin{aligned} a_{ij} &= 1 \text{ if edge } e_j \text{ contains node } i, \\ &= 0 \text{ otherwise.} \end{aligned}$$

¹See appendix A for information about graph definition

Consider now A as an instance of the tool switching problem, with capacity $C = 2$. A job sequence for this problem corresponds to a permutation of E , and hence to a travelling salesman path in the edge-graph of G . Also, it is easy to see that the number of tool switches between two jobs j and k , corresponding to the edges e_j and e_k of G is:

- equal to 1 if e_j and e_k share a common node, that is, if $\delta(e_j, e_k) = 1$ in H .
- equal to 2 if e_j and e_k do not share a common node, that is, if $\delta(e_j, e_k) = +\infty$ in H .

This assertion immediately implies that an optimal job sequence for A (with capacity 2) always corresponds to a minimal length travelling salesman path in H . Hence, we can solve the problem 2 of the appendix A solving the Problem 1, and this entails that Problem 1 is NP-hard. These conclusions were extracted by Crama and provide to the SSP the condition of NP-hard problem.

To see how this assumption would work for $C > 2$, it suffices to adapt the definition of A in the previous argument, by adding $C - 2$ rows of 1s to it; that is, A now has $(M + C - 2)$ rows, and $a_{ij} = 1$ if $i \geq M + 1$. The reasoning goes through with this modification.

2.2.3 Finding the minimum number of setups for a fixed job sequence

As it has been described in section 2.1, the tool switching problem naturally discomposes into two interdependent problems: the sequencing of jobs and the tool switching problem. Tang & Denardo (1988) [40] proved that the sequencing subproblem is the hard part of the work, because the tooling problem can be solved in $O(MN)$ operations by applying the algorithm *Keep Tool Needed Soonest* (KTNS). This algorithm is based on the necessity of keeping always the tools which will be needed the soonest. Tang & Denardo proved that the KTNS can provide solutions using the 0-1 linear programming, giving another perspective to the tooling problem.

Assuming that the tool problem can be solved with the KTNS procedure, the terminology can be re-defined in an appropriate form. Considering a job sequence σ fixed beforehand, let the $M \times N$ (0,1)-matrix P be defined by:

$$\begin{aligned} p_{ij} &= 1 \quad \text{if tool } i \text{ is required for the } j\text{th job in } \sigma, \\ &= 0 \quad \text{otherwise} \end{aligned}$$

P is the matrix obtained by permuting the columns of the data for each job an tool according to the job sequence at hand. Considering c_j as the remaining capacity of each step, the tooling policy will be based in permuting c_j entries for 0 to 1 in the j th column of P . Remaining capacity is defined as:

$$c_j = C - \sum_{i=1}^M p_{ij}$$

The aim for the definition of c_j is to know how many free slots are available on the magazine in each step. These free slots are available to insert another tools and these will be inserted in the

denominated as 0-blocks. Let a 0-block be the maximal subset of consecutive zeroes in a row of P which accomplish the following statements:

1. $1 < j \leq j+k < N$
2. $p_{ij} = p_{i,j+1} = \dots = p_{i,j+k} = 0$
3. $p_{i,j-1} = p_{i,j+k+1} = 1$

A 0-block is defined as the maximal time interval before and after which tool i is needed, but during which it is not needed. Flipping an element of P from 0 to 1 is associated with an extra setup for the tool i . Flipping the elements only reduce the number of extra setups if these elements belong to a 0-block, and if all other elements of this 0-block are also flipped. Thus, the definition of the 0-blocks and the can be explained with the following example:

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Assuming a $C = 2$ the definition of c_j for $j \in [1, 6]$ is:

$$c = [0, 0, 1, 1, 1, 0]$$

The 0-blocks definition would be:

1. $\{(1, 3), (1, 4)\}$
2. $\{(3, 2)\}$
3. $\{(3, 5)\}$

And the matrix resultant permuting 0-blocks is:

$$T = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

As it could be seen the permuting of the 0-blocks award a reduction of the tool switches, proving that once the sequence is defined is not difficult to calculate one set of tools for each step. This will be the basis for the *Keep Tool Needed Soonest* algorithm, the permuting of the free slots of the magazine for tools which will be needed the soonest. The steps in which the KTNS algorithm is based will be presented in the chapter 4, while the theory of the developed metaheuristic is being explained.

2.2.4 Computational conclusions

Once established the definition, all the assumptions for the machine behaviour and the complexity of the problem study, can be extracted the computational conclusions. During the next sections, several solutions already designed for the SSP will be analysed, following the steps presented in the literature from Tang & Denardo (1988) [40] until now. Through the evolution from the integer linear programming of the SSP to the metaheuristic development it will be understood the strengths and weakness of each type of resolution.

The datasets used in order to check the improvement of the results regarding the other methods will be the same as described in Laporte et al. (2004) [32]. In the appendix B the dataset information is shared, for the information of the reader. Through the analysis of the current state of the art a new metaheuristic will be developed, in order to achieve equal or better results using less resources.

2.3 Integer linear programming formulations

Using the same notations presented in section 2.1.3 we should understand the precedents in the solution of the SSP before starting the development of a new one. The background of this problem has been studied since 1988, by Tang & Denardo [40], assuming that the switching cost is tool independent. Through a research in the literature, in this section three formulations currently developed are shared with the aim to solve the SSP with linear programming. Starting with the Tang & Denardo's formulation, presented in 1988, we will evolve through the years to achieve a tighter formulation released in 2015 by Catanzaro et al. [11]. This also will be explained by a half-way formulation to understand the improvements done in the fields of the constraints by Laporte et al.(2004) [32].

2.3.1 Tang and Denardo's formulation

Denote $K = \{1, 2, \dots, n\}$ as the n positions in the job sequence to be determined. Let u_j^k be a decision variable equal to 1 if the job j is processed in the position k and 0 otherwise. Similarly, let w_t^k be a decision variable equal to 1 if tool t is present when processing job k and not when processing job $k - 1$, and 0 otherwise. The formulation presented by Tang & Denardo in 1988 was:

$$\min \sum_{t \in T} v_t^1 + \sum_{t \in T} \sum_{k \in K \setminus \{1\}} w_t^k \quad (2.3.1.1)$$

$$s.t. \sum_{j \in J} u_j^k = 1 \quad \forall k \in K \quad (2.3.1.2)$$

$$\sum_{k \in K} u_j^k = 1 \quad \forall j \in J \quad (2.3.1.3)$$

$$\sum_{j \in J_t} u_j^k \leq v_t^k \quad \forall k \in K, t \in T \quad (2.3.1.4)$$

$$\sum_{t \in T} v_t^k \leq C \quad \forall k \in K \quad (2.3.1.5)$$

$$v_t^k - v_t^{k-1} \leq w_t^k \quad \forall k \in K \setminus \{1\}, t \in T \quad (2.3.1.6)$$

$$u_k^k \in \{0, 1\} \quad \forall k \in K, t \in T \quad (2.3.1.7)$$

$$v_t^k, w_t^k \in \{0, 1\} \quad \forall k \in K, t \in T \quad (2.3.1.8)$$

The objective function 2.3.1.1 minimizes the number of tool switches. In particular, the added value $\sum_{t \in T} v_t^1$ takes into account the loading of the magazine when processing the first job of the sequence. Constraints 2.3.1.2 and 2.3.1.3 are assignment constraints, they take into account that all the jobs will be done and they will be assigned only one time in the sequence. 2.3.1.4 constraint impose that no job requiring tool t can be processed in the position k if such a tool is not present in the magazine at that position. Imposing the necessity for the job tool usage will be achieved the sequence for the tools in each step. Regarding to the magazine capacity the constraint 2.3.1.5 will guarantee that it is never exceeded. It should take into account that all the capacity constraints would be useless in the case of a number of tools lower or equal than the capacity of the magazine. Finally the constraint 2.3.1.6 impose that a tool switch must be counted whenever a tool is present on the step k but it was not present in the next step $k - 1$. The constraint 2.3.1.7 and 2.3.1.8 restrict the value of the decision variables to 0 and 1.

2.3.2 Laporte et al. formulation

To overcome several drawbacks in Tang & Denardo (1988) formulation, Laporte et al. (2004) [32] proposed an alternative integer linear programming formulation that exploits the analogies between the SSP and the TSP. Due to some relaxations in subsection 2.3.1 formulation, sometimes the obtained results could be assumed as correct although they were wrong. To solve this issue, Laporte et al. proposed the creation of a new dummy job 0.

This job represented the start and stop operations and was set by convention $T_0 = \emptyset$. Subsequently they modelled the SSP circuit with all the jobs and the new dummy job 0. In order to describe in detail the proposed formulation, denote $J_0 = J \cup \{0\}$, $J^i = J \setminus \{i\}$, for all $i \in J$, and $J_0^i = J \setminus \{i\}$, for all $i \in J$. Let x_{ij} be a decision variable equal to 1 if a job j follow immediately job i in the sequence and 0 otherwise, for all $i, j \in J_0$, $i \neq j$. Moreover, let y_j^t be a decision variable equal to 1 if tool $t \in T$ is in the magazine when job j is processed and 0 otherwise, for all $j \in J_0$, $t \in T$. Once presented the new variables, the following formulation presented by Laporte et al. (2004) was:

$$\min \sum_{j \in J} \sum_{t \in T_j} z_j^t \quad (2.3.2.1)$$

$$s.t. \sum_{j \in J_0^i} x_{ij} = 1 \quad \forall i \in J_0 \quad (2.3.2.2)$$

$$\sum_{i \in J_0^j} x_{ij} = 1 \quad \forall j \in J_0 \quad (2.3.2.3)$$

$$\sum_{i,j \in S: i \neq j} x_{ij} \leq |S| - 1 \quad \forall S \subset J_0, 2 \leq |S| \leq n - 1 \quad (2.3.2.4)$$

$$\sum_{t \in T} y_j^t \leq C \quad \forall j \in J \quad (2.3.2.5)$$

$$x_{ij} + y_j^t - y_i^t \leq z_j^t + 1 \quad \forall i, j \in J, i \neq j, t \in T \quad (2.3.2.6)$$

$$x_{0j} + y_j^t \leq z_j^t + 1 \quad \forall j \in J, t \in T \quad (2.3.2.7)$$

$$y_j^t = 1 \quad \forall j \in J, t \in T_j \quad (2.3.2.8)$$

$$z_j^t = 0 \quad \forall j \in J, t \in T \setminus T_j \quad (2.3.2.9)$$

$$y_j^t, z_j^t \in \{0, 1\} \quad \forall j \in J, t \in T \quad (2.3.2.10)$$

$$y_{ij} \in \{0, 1\} \quad \forall i, j \in J_0 \quad (2.3.2.11)$$

The objective function 2.3.2.1 minimizes the overall number of tool switches. Constraints 2.3.2.2 and 2.3.2.3 are created for the assignment of all the jobs. 2.3.2.4 prevents the creation of subtours during the creation of the path. Constraint 2.3.2.5 is a capacity one, to avoid the exceed of capacity during the tool assignment. Constraints 2.3.2.6 prevent that job j follows job i in the optimal sequence if, at the same time, job j requires tool t and tool t is neither carried from job i nor inserted for job j . Constraints 2.3.2.7 is a special case for the constraint 2.3.2.6 and account for the loading of the magazine relative to the transition from the dummy job 0 to the first job in the optimal job processing sequence. The constraints 2.3.2.8 and 2.3.2.9 ensure that a tool switch is only performed when the tool is required to immediately execute a job.

This second formulation presented by Laporte clearly dominates the formulation 2.3.1. Even so, the formulation took 1 hour for the computational experiments of 10 jobs instances. In order to improve the lower bounds, Catanzaro and his colleagues developed an improved integer linear programming with an alternative formulation.

2.3.3 Catanzaro et al. formulation

With the aim of achieve a tighter formulation in 2015 was published a new formulation. In Catanzaro et al. (2015) [11] through several steps the previous formulations are improved. These different formulations proposed in the document allow to optimize the running time, but in this subsection it will be only analysed the main formulation developed in the publication.

Considering a decision variable y_{ij}^t equal to 1 if in the processing sequence a tool $t \in T$ is carried in the transition from job i to job j , $i, j \in J_0$, $i \neq j$, and 0 otherwise. Similarly, let z_{ij}^t be a decision variable equal to 1 if in the processing sequence a tool $t \in T$ has to be added in the transition from job i to job j , $i, j \in J_0$, $i \neq j$, and 0 otherwise. Then, a valid formulation for the SSP is the following:

$$\min \sum_{j \in J: i \neq j} \sum_{t \in T_j} z_{ij}^t \quad (2.3.3.1)$$

$$s.t. \sum_{j \in J_0^i} x_{ij} = 1 \quad \forall i \in J_0 \quad (2.3.3.2)$$

$$\sum_{i \in J_0^j} x_{ij} = 1 \quad \forall j \in J_0 \quad (2.3.3.3)$$

$$\sum_{i, j \in S: i \neq j} x_{ij} \leq |S| - 1 \quad \forall S \subset J_0, 2 \leq |S| \leq n - 1 \quad (2.3.3.4)$$

$$\sum_{i \in J_0^j} (y_{ij}^t + z_{ij}^t) = 1 \quad \forall j \in J_0, t \in T_j \quad (2.3.3.5)$$

$$y_{ij}^t + z_{ij}^t \leq x_{ij} \quad \forall i, j \in J_0 : i \neq j, t \in T : t \notin T_i, t \in T_j \quad (2.3.3.6)$$

$$\sum_{t \in T} (y_{ij}^t + z_{ij}^t) \leq Cx_{ij} \quad \forall i, j \in J_0 : i \neq j \quad (2.3.3.7)$$

$$\sum_{k \in J_0^i} (y_{ki}^t + z_{ki}^t) \geq \sum_{j \in J_0^i} y_{ij}^t \quad \forall i \in J_0, t \in T \quad (2.3.3.8)$$

$$y_{ij}^t \geq x_{ij} \quad \forall i, j \in J_0 : i \neq j, t \in T_i \cap T_j \quad (2.3.3.9)$$

$$z_{ij}^t = 0 \quad \forall i, j \in J_0 : i \neq j, t \in T_i \setminus T_j \quad (2.3.3.10)$$

$$y_{0j}^t = 0 \quad \forall j \in J, t \in T \quad (2.3.3.11)$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in J_0 : i \neq j \quad (2.3.3.12)$$

$$y_{ij}^t, z_{ij}^t \in \{0, 1\} \quad \forall i, j \in J_0 : i \neq j, t \in T \quad (2.3.3.13)$$

Constraints 2.3.3.2, 2.3.3.3 and 2.3.3.4 are the same as the previous formulation presented in subsection 2.3.2 (2.3.2.2 / 2.3.2.3 / 2.3.2.4). Constraints 2.3.3.5 impose that in the transition from job i to job j given a tool $t \in T_j$ is either carried from job i or it is added in the transition. Constraints 2.3.3.6 impose tat a tool $t \in T_j$ can be either carried from a previous job or added in the transition, only if such a transition is in the optimal job processing sequence. Constrains 2.3.3.7 are similar to the capacity

constraints (2.3.2.5) of the previous formulation 2.3.2. Constraints 2.3.3.8 impose that if a tool $t \in T$ is carried in the transition from job i to job j then it must be either carried or added in the transition from another job k to job i , $k \neq i$. Constraints 2.3.3.9 impose to carry all of the tools in $T_i \cap T_j$ when considering the transition from job i to job j . Constraints 2.3.3.10 are analogous with constraints 2.3.2.9 in the Laporte et al. formulation. Finally, constraints 2.3.3.11 account for the loading of the magazine relative to the transition from the dummy job 0 to the first job in the optimal job processing sequence. 2.3.3.12 and 2.3.3.13 grant the binary mode for the decision variables.

For the reader's information, know that this formulation was improved in several steps to get a tighter structure with the aim to improve the running times. These improvements are explained in the whole publication Catanzaro et al. (2015) [11].

2.3.4 Conclusions of the integer linear programming formulations

The job scheduling and tool switching problem through the years has been solved using the integer linear programming as seen in subsections 2.3.1, 2.3.2 and 2.3.3. These formulations are able to achieve globally optimal results and the original formulation from Tang & Denardo was improved to get the formulation presented by Catanzaro et al. (2015) [11]. The fastest running times are provided by the formulation presented in the publication of Catanzaro, achieving good results for the largest examples of instances. Nevertheless, formulation 5 presented in the article is characterized by very large gaps. The main issue that guides to this is the fact of the large number of involved variables and valid inequalities, which slows down the solution times for the formulation with a the largest number of jobs and tools. Due to the necessity of working with larger number of tools and jobs another path needed to be developed, with the aim to achieve enough good results in less running time.

2.4 Integer non-linear programming formulations

An alternative path to the integer linear programming is found in literature by authors as Bard, Ghiani or Furrer. They wanted to set an alternative to the ILP and during this section some of the most known formulations will be explained for the information of the reader. Nevertheless, this integer non-linear programming formulations (INLP) won't be able to achieve the global optimal values in all the cases, due to the relaxation of the constraints during the running procedures. On the other hand, these formulations will be able to proceed larger instances, increasing the number of jobs and getting closer to the project aim: the development of a tool to solve larger instances in a feasible time.

Starting with Bard (1988) [5] pure INLP formulation, we will see the first steps in this field is in contrast with the ILP. Once presented the first formulation other mixed INLP formulations will be introduced, made by authors in literature as Ghiani et al. (2010) [24] or Furrer & Mütze (2017) [20]. Other authors in the literature explored this way of solving the SSP, but only the first and last precedents will be introduced.

2.4.1 Bard formulation

Following a parallel path regarding the ILP formulation presented by Tang & Denardo in 1988, Bard [5] in 1988 introduced the formulation of a INLP to solve the job sequencing and tool switching problem. Considering the same assumptions as commented previously in the section 2.1.1.

To specify the model two sets of zero-one variables must be introduced: let $x_{jn} = 1$ if a job j is assigned to position n and 0 otherwise ($j, n = 1, \dots, N$); similarly let $y_{in} = 1$ if a tool i is on the magazine at instant n and 0 otherwise ($i = 1, \dots, M ; n = 1, \dots, N$). For convenience of the author, the problem was formulated through the following integer non-linear programming formulation:

$$\min F(x, y) = \sum_{n=1}^N \sum_{i=1}^M y_{in} (1 - y_{i, n-1}) \quad (2.4.1.1)$$

$$s.t. \sum_{n=1}^N x_{jn} = 1 \quad j = 1, \dots, N \quad (2.4.1.2)$$

$$\sum_{j=1}^N x_{jn} = 1 \quad n = 1, \dots, N \quad (2.4.1.3)$$

$$\sum_{i=1}^M y_{in} \leq C \quad n = 1, \dots, N \quad (2.4.1.4)$$

$$r_{ij} x_{jn} \leq y_{in} \quad \forall i, j, n \quad (2.4.1.5)$$

$$x_{jn}, y_{in} \in \{0, 1\} \quad \forall i, j, n \quad (2.4.1.6)$$

In this formulation y_{i0} ($i = 1, \dots, M$) are given initial conditions, so $y_{i0} = 1$ for all i . Constraints 2.4.1.2 and 2.4.1.3 ensure that each job is assigned to exactly one position of the sequence, while 2.4.1.4 restricts the magazine lading to no more than C tools. Note that the assumption concerning constant interchange times permits the inequality in 2.4.1.5 to be strengthened to an equality. Constraint 2.4.1.6 assures that if job j requires tool i ($r_{ij} = 1$) and is assigned to position n ($x_{jn} = 1$), then tool i will be on the magazine at instant n ($y_{in} = 1$). The objective function performs as a switch counter and can be linearised in a number of ways, as suggest the publication of Tang & Denardo (1988) [40], introducing $M \cdot N$ counting variables. Through some lagrangian relaxation the results of this INLP are achieved in feasible time for instances up to 25 jobs, but they are not the globally optimal ones.

2.4.2 Ghiani et al. formulation

Following the steps established by Brad in 1988, the publication of Ghiani et al. (2010) [24] introduced a new formulation for the SSP solving. Through the development of a branch-and-cut procedure it was achieved the solution of much larger instances that those reported in the literature (see subsection 2.4.1). Proofing the fact of a symmetry property² for the sequences of the tool magazine, it was

²See "Symmetry property" section in Ghiani et al. (2010) [24]

showed how the SSP allowed the authors to speed up the branch-and-cut procedure.

Using the definition of lower bounds a branch-and-cut procedure was designed based on the formulation of the SSP as a shortest Hamiltonian cycle problem with a non-linear objective function. Modelling the problem as a graph $G = (V, A)$ with V representing the set of jobs (including a dummy job 0), the transitions between jobs were represented as arcs A . Giving a party type sequence $(0, u_1, \dots, u_{r-1}, u_r, \dots, u_{r+1}, \dots, u_n, 0)$, with $u_r = 1$ and $u_{r+1} = j$, the transition cost c_{ij}^k from i to j depends on the whole subsequence $p_k = (0, u_1, \dots, u_{r-1}, i)$ and can be computed in polynomial time through the KTNS rule explained in Tang & Denardo (1988) [40] publication. Let x_{ij} be a binary variable equal to one if and only if job j is processed just after job i and let $x = (x_{ij})$ be a vector describing the party type sequencing. The transition cost from i to j can be expressed as follows:

$$c_{ij} = \max\{0, \max_{p_k \in K_i} \{c_{ij}^k (\sum_{(r,s) \in p_k} x_{rs} + x_{ij} - |p_k| - 1)\}\} \quad (2.4.2.1)$$

Where K_i is the set of all paths $p_k = (0, u_1, \dots, u_{r-1}, i)$ in the graph G associated with one subsequence k . Let be noted that $c_{ij}(x) \neq c_{ji}(x)$. The SSP was formulated in the non-linear programming as:

$$\min \sum_{i,j \in A} c_{ij}(x) \quad (2.4.2.2)$$

$$s.t. \sum_{j \in V \setminus \{i\}} x_{ij} = 1 \quad i \in V \quad (2.4.2.3)$$

$$\sum_{i \in V \setminus \{j\}} x_{ij} = 1 \quad j \in V \quad (2.4.2.4)$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 = 1 \quad S \subset V \text{ and } 2 \leq |S| \leq n - 1 \quad (2.4.2.5)$$

$$x_{ij} \in \{0, 1\} \quad i, j \in A \quad (2.4.2.6)$$

The non-linear objective function 2.4.2.2 represent the number of tool switches whereas relationships 2.4.2.3 and 2.4.2.4 are assignment constraints and inequalities 2.4.2.5 prevent the formation of subtours. With this formulation, the usage of a lower bounds procedure described in the article and the tailored implementation of the KTNS, the branch-and-cut was developed towards the solution of the SSP. In his article, Ghiani et al. (2010) [24] compares the computational results with the previous exact procedure. Also, the computational results showed that for instances up to 45 part types and 30 tools valid solutions were reached.

2.4.3 Furrer & Mütze formulation

Following the steps of Ghiani and using the basis established by Bard in 1988, Furrer & Mütze (2017) [20] published a branch-and-bound algorithm with the aim to solve the SSP using the integer non-linear programming. Through a first phase establishing a lax-minimization, the second phase is focused on the resolution of the SSP through a branch-and-bound strategy. Combining the knowledge provided

by Crama et al. (1994) [13] in the fields of minimizing the tool switches for a fixed program, the formulation of Furrer performed using the zero-blocks logic definition. Running different algorithms explained in detail in the article ³, the authors were able to achieve better results than the previous formulations.

During a time Furrer & Mütze formulation was considered the state of the art for the SSP, due to the quality of the results and the usage of a feasible time to achieve them.

2.4.4 Conclusions of the integer non-linear programming formulations

Understanding the works in the field of the integer non-linear programming formulation towards the resolution of the SSP can be seen how the obtained results and the conclusions made by the authors are comparable in terms of tool switches respective of the integer linear programming formulations presented in the section 2.3. The main difference between these two paths to solve the SSP is through non-linear programming the optimal, where the value is not guaranteed for the instances, but instead the algorithms can be applied for bigger instances, reaching higher values of jobs and tools. Since the publication of the Bard formulation other authors apart from Ghiani et al. and Furrer & Mütze (2017) have worked in other type of integer non-linear programming, but it has been considered to explain the main progress in the literature.

2.5 Current state of the art

Once explained the precedents in the fields of ILP and INLP the current state of the art will be analysed to understand the nowadays best solution for the SSP. Taking into account that the integer linear programming takes too many resources in terms of time and that integer non-linear programming doesn't provide the global optimal solution a new path should be explored.

In the beginning of 2018 the Ahmadi et al. [1] publication was released, establishing a solution for the SSP based on genetic algorithms and Q-learning. Through the formulation of the SSP as a Travelling Salesman Problem of Second Order (2-TSP), introduced at first time by Jäger & Molitor [29], Ahmadi et al. presented a new approach for the tool switching calculation. For a given sequence (i, j, k) , the 2-TSP measure the number of tool switches required in transition from job j to job k by considering both job i and job j , not only job j . In the publication is shared how through genetic algorithms as the presented by Starkweather et al. (1991) [38] and a patching heuristic based in Karp & Steel (1985) [31] the 2-SSP will be a better approximation to the unknown optimal solution.

The 2-TSP is considered also a problem of NP-hard complexity as explained in the article and the formulation is based in a second order assignment problem (2-AP). In the AP, we have n workers and n jobs and each worker performs one job and each job is performed by one worker. Now we wish to find an assignment solution such that the sum of the costs of the individual assignments of workers to the jobs is minimized. The following formulation is presented by Ahmadi et al. as the current state of

³For more information about the procedure read the whole article presented by Furrer & Mütze (2017) [20]

the art for the SSP:

$$Z = \min \sum_{i \in V} \sum_{\substack{j \in V \\ j \neq i}} \sum_{\substack{k \in V \\ k \neq i \neq j}} C_{ijk} X_{ijk} \quad (2.5.0.1)$$

$$s.t. \sum_{\substack{j \in V \\ i \neq j}} \sum_{\substack{k \in V \\ k \neq i \neq j}} X_{ijk} = 1 \quad \forall i \in V \quad (2.5.0.2)$$

$$\sum_{\substack{j \in V \\ i \neq j}} \sum_{\substack{k \in V \\ k \neq i \neq j}} X_{ijk} = 1 \quad \forall j \in V \quad (2.5.0.3)$$

$$\sum_{\substack{i \in V \\ i \neq k}} \sum_{\substack{j \in V \\ j \neq i \neq k}} X_{ijk} = 1 \quad \forall k \in V \quad (2.5.0.4)$$

$$\sum_{\substack{k \in V \\ k \neq i \\ k \neq j}} X_{ijk} = \sum_{\substack{k \in V \\ k \neq i \\ k \neq j}} X_{kij} \quad \forall i \in V, j \in V, j \neq i \quad (2.5.0.5)$$

$$X_{ijk} \in \{0, 1\} \quad (2.5.0.6)$$

The formulation published by Ahmadi et al. is similar in big terms to the formulations presented previously, but working with a travelling salesman problem of second order is a bit different in terms of constraints. X_{ijk} is a binary variable which is 1 if pair (i, j) precedes job k and is 0 otherwise. C_{ijk} is the number of tool switches from job j to job k given the predecessor is job i .

Equations 2.5.0.3, 2.5.0.4, 2.5.0.5 mean that each vertex appears exactly once as the first time, second or third vertex in a path of three vertices of a tour. Equation 2.5.0.6 specifies that if there is a path from (i, j) to k , then there is another path from k to (i, j) . Solving this 2-AP leads to generate unknown number (Y) of subtours which are locally optimal. To step towards the solution and in order to obtain a Hamiltonian tour, the Karp-Steele patching algorithm [31] is used. This patching algorithm is described in the article⁴ and provides the best path to solve the SSP taking into account all the possibilities. Through a Q-learning procedure and genetic algorithms the 2-AP was enhanced in the operators sequence selection and helped to reach better results given previously in the literature. The whole formulation is named DQCA (Dynamic Q-learning Genetic Algorithm) and obtained improved results in terms of solution and resources. The previous state of the art was defined by Paiva & Carvalho (2017) [36], and the results of Ahmadi et al. has replaced it as the best solution developed for the SSP.

⁴See section 3.1.2 in Ahmadi et al. [1]

Chapter 3

Metaheuristic definition

3.1 Introduction

The term *metaheuristic* was first introduced in Glover (1986) [26] and derives from the composition of two Greek words: *heuristic* and *meta*. Heuristic derives from the verb *heuriskein* which means "to find", while the suffix *meta* means "beyond, in an upper level". Before this term was widely adopted, metaheuristic were often called *modern heuristic*.

As described in the literature (Blum & Roli 2003 [8], Garcia et al. 2013 [21] and Boussaïd & Lepagnot 2013 [9]), the metaheuristics are the right solution for the combinatorial problems, but there is not a general one to solve all kinds of problems. Depending of the behaviour of the problem, the solution will require different elements to be applied. The combinatorial problems are present in most of the processes in our world and solving them in a mathematical way takes too many resources, mainly in the dimensions of time, due to the exponential expansion of combinatorics. Focusing on the problem studied in this project, the *job Sequencing and tool Switching Problem*, the fact of developing a metaheuristic was awarded as an alternative path instead of the ILP and the INLP, seen in sub-chapters 2.3 and 2.4 respectively. During this chapter it will be introduced the terms of a metaheuristic and two of them will be explained, taking into account the aim of solving the SSP and improving the solution for the problem.

3.2 Definition

Many optimization problems of practical as well as theoretical importance consists of the search for the "best" configuration of a set of variables to achieve the set goals. In the Blum & Roli publication in 2003⁵, it is explained that the different categories in the field of combinatorial optimization: those where the solutions take *real-valued* variables, and those where the solutions are taking *discrete* variables. In the literature the SSP has been described as a problem with encoded *discrete* variables, and for the metaheuristic development will be kept in the same way. The combinatorial problems

⁵See chapter 1 in Blum & Roli (2003) [8]

can be solved in two ways through the algorithms: the *complete* one, where the optimal solution is reached in a bounded time, and the *approximate* one, in which it sacrifices the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time. As discussed during the document, the main problem of finding a solution for the SSP is the fact of doing it in a feasible time. For the development of a metaheuristic for the problem studied, it will be selected by an *approximate* strategy, due to the precedents in the literature which tried to find the optimal values through exact modes. This approximation won't guaranty an optimal value but instead will rely on the algorithm to work with bigger instances.

Among the approximated methods it is the strategy is distinguished of the constructive methods and the *local search* methods. Constructive algorithms generate solutions from scratch by adding to an initial part solution, until a solution is complete. They are typically the fastest approximate methods, yet they often return solutions of inferior quality when compared to local search algorithms. On the other hand the local search algorithms start from some initial solution and iteratively try to replace the current solution with a better solution in an appropriately defined neighbourhood. Let a neighbourhood be a function of $N : S \rightarrow 2^S$ that assigns to every $s \in S$ a set of neighbours $N(s) \subseteq S$. In this case $N(s)$ is called the neighbourhood of s . The main characteristics of the main properties of a metaheuristic, as defined per Blum & Roli [8] are:

- Metaheuristics are strategies that "guide" the search process.
- The goal is to efficiently explore the search space in order to find near optimal solutions, using less resources.
- Techniques which constitute metaheuristics algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- Metaheuristics are not problem specific.

Summarizing it could be said that metaheuristics are high level strategies for exploring search spaces by using different methods. Hereby is important the balance between diversification and intensification. Diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience. Then, the fact of intensification would refer to quickly identify regions in the search space with high quality solutions and on the other side not to waste too much time in regions of the search space which are either already explored or which do not provide high quality solutions.

The search strategies of different metaheuristics are highly dependent on the philosophy applied in the own metaheuristic. Some of them contain an intelligent behaviour as an extension for the local search

algorithms. Other incorporate a learning component that enhances the exploration of new areas for the search, as presented in this document. The main issue about the metaheuristic development is how to adapt the theory to a specific problem, for example the SSP. In the next chapter some of the classifications of the metaheuristic will be presented, just for information of the reader.

3.3 Classification

Before introducing one existing metaheuristic for the *job Sequencing and tool Switching Problem* let it be classified first by the different by categories in the metaheuristic algorithms. Understanding each classification and the strengths of each category the SSP metaheuristic will start to be defined. The classification is:

- *Nature-inspired vs. non-nature inspired.* Depending of the origin of the algorithm it will be defined as nature-inspired algorithms, like Genetic algorithms and ant algorithms, and non nature-inspired ones such as Tabu Search and iterated local search.
- *Population-based vs. single point search.* Another characteristic that can be used for the classification of metaheuristics is the number of solutions used at the same time: Does the algorithm work with a single solution or a population at any time? Single solutions algorithms are called trajectory methods and encompass local search based metaheuristics, like tabu search or Iterated local search. They all share the property of describing a trajectory during the search in the space. In the other hand the population based algorithms perform searches which describe the evolution of a set of points in the search space.
- *Dynamic vs. static objective function.* While some algorithms keep the objective function given in the problem representation "as it is", some other as Guided local search modify it during the search. The idea behind this approach is to escape from local minima by modifying the search landscape.
- *One vs. various neighbourhood structures.* Most metaheuristic algorithms work o one single neighbourhood structure. Others work a set of neighbours structure which gives the possibility to diversify the search by swapping between fitness landscapes.
- *Memory usage vs. memory-less methods.* There are several different ways of making use of memory. The first usually keeps track of recently performed moves or decisions taken. The second is usually an accumulation of parameters about the search. The way to use the memory will denote the classification of the metaheuristic.

3.4 Metaheuristic applied to the travelling salesman problem

In the following section it will be explained how a metaheuristic is applied to the TSP formulation. As it has been detailed previously, the SSP can be defined as a TSP, by considering the jobs as the different cities to visit and the tools as a different problem. Before starting to develop a metaheuristic for the SSP, it first will be explained an ant based metaheuristic for the travelling salesman problem. The definition of the problem is stated in the subsection 2.2.1 in the chapter 2.

Ant algorithms are a recently developed, population-based approach which has been successfully applied to several *NP*-hard combinatorial optimization combinatorial problems. As the name suggests ant algorithms have been inspired by the behaviour of real ant colonies, in particular by their foraging behaviour. The main idea of the ant algorithms is the indirect communication between a colony of agents, established with pheromone trails. The pheromones are a kind of distributed numeric information which is modified by the ants to reflect their experience while solving a particular problem. All the ant algorithms applied to the TSP fit perfectly with the ant colony optimization metaheuristic (ACO). Between the different types of ACO metaheuristic in this section, it will be explained that the more basic ant model towards the solution of the travelling salesman problem, the ant system. The SSP can be modelled as different nodes and being solved as a TSP instance, so the fact of showing the solution of the TSP with the usage of ACO based metaheuristics can open a new framework in the SSP metaheuristics.

In ACO algorithms ants are simple agent which, in the TSP case, construct tour by moving from city to city on the problem graph. The ant's solution construction is guided by artificial pheromone trails and a priori available heuristic information. When applying ACO algorithms to the TSP, a pheromone strength $\tau_{ij}(t)$ is associated to each arc (i, j) , where $\tau_{ij}(t)$ is a numerical information which is modified during the run of the algorithm and t is an iteration counter. If we are applying the TSP to a symmetric instance $\tau_{ij}(t) = \tau_{ji}(t)$, and in the asymmetric instances would be $\tau_{ij}(t) \neq \tau_{ji}(t)$. Initially, each of the m ants is placed on a randomly chosen city and then iteratively applies at each city a state transition rule. An ant constructs a tour as follows. At a city i , the ant chooses a still unvisited j in the possible neighbourhood $N(s)$ through a probabilistic rule, based on the pheromone strength and the heuristic information. This heuristic information can be defined as the distance between the arcs, a function... Ants probabilistically prefer cities which are close and are connected by arcs with a high pheromone trail strength. To construct a feasible solution each ant has a limited capacity of memory, a kind of tabu list, in which the current partial tour is stored. The memory is used to define at each construction the already visited cities and the remaining nodes to visit.

After all ants have constructed a tour, the pheromones are updated. This is typically done by first lowering the pheromones trail strengths by a constant factor, and then the ants are allowed to deposit new pheromone values in the pheromone map once the evaporation value has already updated the pheromone map. The trail update is done in such a form that arcs contained in shorter tours and/or visited by many ants receive a higher amount of pheromone and are therefore chosen with a higher probability in the following iterations of the algorithm. The best performing ACO algorithms for

the TSP improve the tours constructed by the ants applying a local search algorithm. Hence, these algorithms are hybrid algorithms combining probabilistic solution construction by a colony of ants with standard local search algorithms. Such a combination may be very useful to improve the obtained results, achieving better results. On the other hand the usage of local searches modifies the behaviour of the ants, changing the obtained pheromone based results for local search results. Keeping the better of the two alternatives, the algorithm will be more complete. In general, all ACO algorithms for the TSP follow a specific algorithm scheme, which is outlined in algorithm 1. After initializing the pheromone trails and some parameter a main loop is repeated until the termination conditions. In the main loop, first the ants construct feasible tours, then the generated tours are improved with local search, and finally the pheromone trails are updated.

Algorithm 1 ACO algorithm for TSPs

```

Set parameters, Initialize pheromone trails
while termination conditions not met do
    Construct solutions
    Apply Local Search      {Optional}
    Update trails
end while
  
```

3.5 Metaheuristic applied to SSP

Once defined and classified the concept of metaheuristic it will start to be applied to the SSP, being explained through a case in the literature. In 2003 it was released by Al-Fawzan & Al-Sultan [2] a Tabu Search based metaheuristic to solve the *job Sequencing and tool Switching Problem*. The approach assumed by the authors will set a basis for the metaheuristic development, taking into account the formulation of the problem and the results obtained. In the first part of the section will be defined the Tabu Search metaheuristic and in the second part the application for the SSP will be detailed.

3.5.1 Tabu Search definition

As characterized before, a Tabu Search (TS) metaheuristic would be classified in the following fields:

- Non-nature inspired
- Single point search
- Static objective function
- One neighbourhood
- Memory usage methods

The TS is among the most cited and used metaheuristics for combinatorial problems. The basic idea was first introduced in a publication of Glover in 1986 [26], based on earlier ideas published by Glover (1977) [25]. The tabu search starts at some initial point and then moves successively among neighbouring points. A candidate list of solution is generated in each step, by a strategy designed to include higher quality neighbours on the candidates list. At each iteration a movement is made to the best point in the neighbourhood of the current point that is a member of the candidate list. The short term memory is implemented as a tabu list that keeps track of the most recently visited solutions and forbids moves toward them. The neighbourhood of the current solution then is restricted to the solutions which don't belong to the tabu list. The size of the tabu list can be either fixed or variable. As explained in Al-Fawzan & Al-Sultan (2003) [2], tabu search requires the following ingredients:

- Initial point.
- Mechanism for generating some candidate neighbourhood of the current point.
- Tabu list.
- Aspiration criterion.
- Stopping criterion.

The above tabu search version presented is a primitive one that considers only short term memory. During the course of the years it has been improved with versions⁶ with long-term memory elements.

3.5.2 Application for the SSP

Metaheuristic development using tabu search for the *job Sequencing and tool Switching Problem* was mainly focused on the job sequencing problem, since it is the hard part of the problem, as commented in the subsection 2.2.3. The usage of the short term memory can be summarized as follows: start with any initial sequence of jobs and compute its number of tool switches and store it as the best solution so far. Once this sequence is used, generate all the neighbours from this initial point. For every sequence, the optimal loading policy will be calculated by keeping the tool needed soonest algorithm, first seen in Tang & Denardo [40] in 1988. Then the best nontabu improving sequence is then selected and its associated attribute is stored in the tabu list. This procedure will be repeated and controlled by tabu search until a predetermined number of iterations is performed. Through some improvements to this algorithm, namely the generation of random sequences and the integration of long term memory, the tabu search metaheuristic was developed for the SSP by Al-Fawzan & Al-Sultan (2003) [2].

The main conclusion extracted by the author of the metaheuristic was that the results weren't so far of the best solutions known. Integrating the long term memory the metaheuristic reached higher quality solutions and this fact setted a precedent in the SSP resolution. The conclusions extracted of the metaheuristic motivated to follow the steps in the metaheuristic field towards the development of a new one to solve the proposed SSP.

⁶For a longer explanation of the tabu search metaheuristic and all the elements, read Glover & Laguna (2013) [27]

Chapter 4

Metaheuristics for the SSP

4.1 Introduction to metaheuristic development

Once analysed publications towards the SSP solution in chapter 2 and the metaheuristic definition in chapter 3, in this 4th chapter the metaheuristics for the SSP will be developed. While the computational results will be shown in chapter 5, in this chapter will be presented the theoretical basis for the metaheuristic development.

Through the metaheuristic development it will be analysed which of them fits better for the SSP. As presented in the subsection 2.2.3, Tang & Denardo [40] concluded that the logic of the zero blocks guides to a solution of the tool switching problem in the SSP. In order to achieve a valid solution using the mentioned blocks, it was decided to develop a greedy based metaheuristic using the *Keep Tool Needed Soonest* for the determination of the sequence of the jobs and the tools needed in each step. The KTNS is based on the same logic presented by Tang & Denardo, and using it in the metaheuristic would help to achieve optimal values as result. Through the combinations of partials sequences and selecting the minimum cost between one job and the next one, the aim is to achieve an optimal solution in a feasible time, as established in the scope of the project. This approach was first shared by Crama et al. (1994) [13], named as multiple start greedy, and the complexity of the algorithm is $O(MN^4)$ in the worst cases, due to the application of $O(N^2)$ times of the KTNS algorithm. Nevertheless, the possibilities of achieving good initial results pushed to the election of this method as first attempt, even disregarding the high complexity. In the next chapters the methodology of this first attempt will be explained in detail, for the information of the reader.

4.2 Greedy Metaheuristic development

Crama et al. [13] developed a heuristic for the SSP solution based on the greedy algorithm. Through the selection of the best option in each step was expected to get the best optimal solution for the field/group studied. The first SSP metaheuristic attempt was decided to be based on the same approach. For a list of jobs candidates two list are defined: the list of the jobs already added to the "best" sequence

and the remaining jobs which await to be added to the "best" sequence. The first list will be noted as σ , while the other one will be Q . The following steps define the procedure of the greedy based heuristic, in which the starting job will be the job number 1:

1. Start with the partial job sequence $\sigma = (1)$; let $Q = \{2, 3, \dots, N\}$.
2. For each job j in Q , let $c(j)$ be the cost of the partial sequence $[\sigma, j]$ (the number of tool switches needed). The tool distribution will be calculated with the *Keep Tool Needed Soonest*.
3. Let i be a job in Q for which $c(i) = \min_{j \in Q} c(j)$; let $\sigma := (\sigma, i)$ and $Q := Q \setminus \{i\}$.
4. If Q is not empty, then repeat step 2; else, stop with the complete sequence σ .

To increase the possibilities of getting an optimal solutions it makes sense to take the following approach: run N times greedy, once for each initial sequence $\sigma(j)$ ($j = 1, 2, \dots, N$), and retain the best complete sequence found. It can be seen how this heuristic dominates greedy, in terms of quality of the job sequence it produces. To improve the quality of the solution, this approach is reinforced with the fact of doing i iterations, where i represents the number of times that multiple start greedy will run to attempt the improvement of the previous solutions. This type of metaheuristic is named *Greedy Randomized Adaptive Search Procedure* (GRASP) and was introduced first time by Hart & Shogan (1987) [28]. The calculation of the total costs of the sequence will be done through the *Keep Tool Needed Soonest* algorithm, which will provide a right distribution of the tools needed in each job to reduce the tool switches. This algorithm is explained in detail in the next subsection, 4.2.1. Following the metaheuristic definition done in 3.3, the GRASP metaheuristic would be defined as:

- Non-nature inspired
- Single point search
- Static objective function
- Various neighbourhood
- Memory less methods

As previously discussed, the weak point of the greedy based heuristics is the exponential growth when the instance to be solved increases the size. Even though, it was determined to check the results of the GRASP metaheuristic for the *job Sequencing and tool Switching Problem*. The metaheuristic is summarised as:

Algorithm 2 Procedure GRASP metaheuristic for SSP

```

while iterations  $\leq$  limit or time  $\leq$  limit do
  for job in AllInstanceJobs do
     $\sigma = job$ 
     $Q = AllInstanceJobs \setminus \{job\}$ 
    while  $Q \neq \emptyset$  do
       $Cost_i = KTNS(\sigma + Q_i)$ 
       $Q_\sigma = \min_{Q_i \in Q} Cost_i$ 
       $\sigma = \sigma + Q_\sigma$ 
       $Q = Q \setminus \{Q_\sigma\}$ 
    end while
  end for
end while

```

4.2.1 Keep tool needed soonest algorithm

Tang & Denardo in 1988 [40] affirmed that a suitable solution once the sequence is established would be the application of the *Keep Tool Needed Soonest* algorithm. The aim of this algorithm is determine the set of tools to be placed on the machine at each instant so that the total of tool switches is minimized. Decreasing the tool switches for one fixed sequence and selecting in every iteration the sequence which minimize the costs, it could be reached the solution. This KTNS procedure is based in the following properties:

1. At any instant, no tool is inserted unless it is required by the next job.
2. If a tool must be inserted, the tools that are kept (not removed) are those needed the soonest.

Hence, the application of the KTNS guides to achieve a properly set of tools for each step in a sequence defined. This tool distribution will minimize the tool switching. Through two sets definition, the tooling problem of the SSP will be solved. The first of these sets is $T(i, n)$, which refers to the set of all the instants at or after instant n at which tool i is needed. The other is $L(i, n)$, which for all the tools will indicate the first instant at or after instant n at which tool i is needed. The explanation can be completed with the following formulation:

$$T(i, n) = \{m - 1 : \exists j, m > n, A_{ji} = 1 \text{ and } x_{jm} = 1\} \cup \{N\}$$

$$L(i, n) = \min\{m : m \in T(i, n)\}$$

For a complete definition of $L(i, n)$, is setted $L(i, n)$ equal to N if a tool i is not needed after instant n . The KTNS procedure is completed with the vector J_i , which defines in each step which tools should be on the magazine. The value of 1 indicates that the tools i is needed to be on the magazine on the

given instant n . The value 0 indicates that the tool is no needed to be charged on the magazine.

The steps to obtain a solution are:

- Step 1: Set $J_i = 1$ for C values of i having minimal values of $L(i, 0)$. Break the ties arbitrarily. Set $J_i = 0$ for the remaining $M - C$ values of i . Set $n = 1$.
- Step 2: Set $W_n = J$. Stop if $n = N$.
- Step 3: If each i having $L(i, n) = n$ also has $J_i = 1$, set $n = n + 1$ and go to step 2.
- Step 4: Pick i having $L(i, n)$ and $J_i = 0$. Set $J_i = 1$.
- Step 5: Set $J_k = 0$ for ta k that maximizes $L(p, n)$ over $\{p : J_p = 1\}$. Go to step 3.

Steps 2 and 3 assure that W_n consists of all the tools required by the n th job in the sequence. All the tools to be inserted (property 1 of the KTNS policy) are considered in the steps 1 and 4. Furthermore, the tools to be kept (property 2 of the KTNS) are determined by the tools to be removed from the machine, which are considered in the step 3.

The application of this procedure has been used in the literature by Tang & Denardo (1988) [40], Catanzaro et al. (2015) [11], Laporte et al. (2004) [32] and Al-Fawzan & Al-Sultan (2003) [2].

4.3 Improving towards the solution

In order to find a better alternative towards the solution of the SSP, it was thought to develop a metaheuristic with lower complexity values. The GRASP metaheuristic of chapter 4.2 was characterized of the high complexity values, which could lead to poor quality results. The development of an alternative which could use lower values of calculation complexity could achieve better results, improving the usage of the available resources. The proposed alternative was decided to be an Ant Colony Optimization (ACO), which started to be developed from the more basic formulation: the ant system (AS). This is the same as explained for TSP in 3.4, but this time applied for SSP. This metaheuristic approach was released by Dorigo (1992) [16], and keeping the metaheuristic classification presented in 3.3 is defined as:

- Nature inspired
- Population based
- Static objective function
- Various neighbourhood
- Memory usage methods

The way to find the solution is completely different to the GRASP metaheuristic presented in section 4.2, and in the literature the ACO isn't used to solve the SSP. In subsection 4.3.1 it is presented the ACO definition, with the explanation of all the components. After that it will be characterized for the SSP.

4.3.1 Ant Colony Optimization definition

Ant Colony Optimization is one of the most recent techniques for approximate optimization, and authors as Blum (2005) [7], Dorigo & Stützle (2010) [19] and Dorigo & Birattari (2011) [17] described in the literature how to develop this nature inspired metaheuristic. The inspiration source of ACO algorithms are the real ant colonies, more specifically by the ant foraging behaviour. The core of the ants behaviour is the way how they can establish an indirect source of communication between them by means of chemical pheromone trails. These pheromones enable them to find the short path between their nest and food sources. This characteristic is treated as the main principle of the ACO metaheuristic, allowing to establish a communication between the different ants with the aim to find the shortest path.

Ants are social insects. They live in colonies and their behaviour is governed by the goal of colony survival rather than focusing on the survival individuals. When searching for food, ants initially explore the area surrounding their nest in random manner. While they are moving they are spreading a chemical pheromone trail on the ground, which will be smelled by the others. When they are aimed to select their way, they tend to choose in probability the paths marked by strong pheromone concentrations. Once they get a food source, they evaluate the quantity and the quality of the the food and bring some back to the nest. During this trip, the amount of pheromone that an ant leaves on the ground depends on variables as the quality of the food or the quantity. The pheromone trails will guide other ants to the food source and will provide a guide to take the shortest path from the nest to the food.

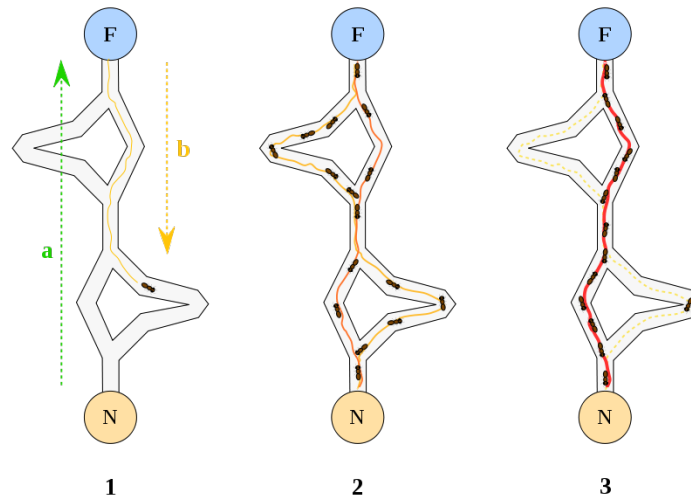


Fig. 4.1 Demonstration of ants behaviour

As shown in figure 4.1, at the beginning (1) the ant will take a random path until it will reach the food. Once the ant has taken it, will spread the pheromone during the way back to the nest (described by the line b). The other ants will smell this pheromone and the random election will grant with more

probability the zones with more pheromone trail (see step 2). After keep doing it for a while, all the ants will take the shortest path (step 3). The aim of the ACO is to develop a procedure able to do a simulate the pheromone trail used by real ants.

As a first step towards an algorithm for discrete optimization it will be shown the simplified pheromone model. Let be considered a graph $G = (V, E)$, where V consists of two nodes, named v_s (representing the nest), and v_d (representing the food source). Furthermore, let E be two links, e_1 and e_2 , between v_s and v_d . To e_1 we assign the length of l_1 , and to e_2 a length l_2 such $l_2 > l_1$. This example is explained in Blum (2005) [7], and will help to understand the main principle for the ant behaviour model. In other words e_1 is the shortest path between the food and the nest, and e_2 represents the longest path. While the ants are moving they are placing pheromone trails. Let be introduced an artificial pheromone value τ_i for each of the two links e_i with $i = 1, 2$. Such a value indicates the strength of the pheromone trail on the corresponding path. Introducing n_a artificial ants, starting from the v_s (the nest) an ant chooses a path with the following probability:

$$p_i = \frac{\tau_i}{\tau_1 + \tau_2}, \quad i = 1, 2 \quad (4.3.1.1)$$

If $\tau_1 > \tau_2$, the probability of choosing e_1 is higher, and vice versa. The trip back from the node v_d , an ant uses the same path as it chose to reach v_d , and it changes the artificial pheromone value associated to the used edge depending on different variables. More specifically, picking edge e_i an ant changes the artificial value τ_i as follows:

$$\tau_i \leftarrow \tau_i + \frac{Q}{l_i} \quad (4.3.1.2)$$

Let Q be a positive constant which is a parameter of the model. This will indicate the amount of artificial pheromone that is added in each explored arch, and as shorter is the distance of the path (denoted with l_i), the higher the amount of added pheromone. Q value is settled in all the project with the value of 1. In order to simulate the evaporation, the whole amount of pheromone is submitted to a pheromone evaporation in each iteration. The concept of iteration means in the ACO framework each time that the desired amount of ants is released. Increasing iterations the results are getting better until the stagnation. The evaporation is modelled as follows:

$$\tau_i \leftarrow (1 - \rho) \cdot \tau_i, \quad i = 1, 2 \quad (4.3.1.3)$$

The parameter $\rho \in (0, 1]$ is a parameter that regulates the pheromone evaporation. Finally, all ants conduct their return trip and reinforce their chosen path as outlined above. Meanwhile the ACO is running, the pheromone will be updated every iteration, contributing to the problem resolution. Let be reminded that all these definitions are the simpler ones and are explained to clarify the ACO concepts. Enhancing the formulation to general terms between two nodes, the definition for the added pheromones trails is:

$$\tau_{ij} \leftarrow \tau_{ij} + \frac{Q}{d_{ij}} \quad (4.3.1.4)$$

The value of Q is the same as outlined before 1 and d_{ij} represent the distance between node i and j . For the pheromone evaporations is used:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} \quad (4.3.1.5)$$

Let be explained the main differences between the real ants behaviour and the behaviour of the artificial ones. These differences are pointed in the publication of Blum (2005) [7] as follows:

- Real ants move in their environment in an asynchronous way, the artificial ants are synchronized. Every iteration simulated in the system will affect to all the ants at the same time.
- While real ants leave pheromone on the ground whenever they move, artificial ants only deposit artificial pheromone on their way back to the nest.
- The foraging behaviour of real ants is based on an implicit evaluation of a solution, shorter paths will be completed earlier than longer ones, and therefore they will receive pheromone reinforcement more quickly. In contrast, the artificial ants behaviour evaluate a solution with respect to some quality measure which is used to determine the strength of the pheromone reinforcement that the ants perform during their return trip to the nest.

The last parameter used in the ant colony optimization is the random proportional rule. With the aim to establish a relationship between the random probability and the real costs it is defined a weight function with the variables α and β . For example, the probability with which ant k , currently at city i , chooses to go to city j is:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \text{ if } j \in N_i^k \quad (4.3.1.6)$$

Where $\eta_{ij} = 1/d_{ij}$ is a heuristic value which is available a priori, in the case of the SSP will be defined as the cost between one job and the next one in the added sequence calculated with the *Keep Tool Needed Soonest*. α and β are two parameters which determine the relative influence of the pheromone trail (τ_{ij}) and the heuristic information (η_{ij}). N_i^k is defined as a feasible neighbourhood of ant k when being at city i , the set of cities which an ant k has not visited yet (the probability of choosing a city outside the neighbourhood is 0). The role of the parameter α and β is the following: if $\alpha = 0$ the closest cities are more likely to be selected and with $\beta = 0$ only the pheromone values are used, without any heuristic bias. The value of $\alpha = 0$ corresponds to a classic stochastic greedy algorithm, shared with the GRASP metaheuristic. Using $\beta = 0$ generally leads to rather poor results and, in particular, for values of $\alpha > 1$ it leads to the rapid emergence of a stagnation situation, a situation in which all the ants follow the same path and construct the same tour. In Dorigo & Stützle (2010) [19] are shared the recommended values for the ACO metaheuristic development, extracted after the analysis of the influence of the different parameters.

In terms of finishing the procedure, in the ant colony optimization it is used the iterations idea, which will define the limits of the running times. Also it is defined the amount of ants released in each

iteration, which will help to develop the optimal path. These parameters are identified as n_a (number of ants) and n_{it} (number of iterations), and as a stopping condition of the algorithm it can be either established as a time limit or an iterations limit. In the next subsection it will be explained the procedure to run the ACO algorithm, and the values of parameters already explained will be defined.

4.3.2 Ant colony optimization scheme

Once analysed the main definitions for the ant colony optimization metaheuristic it will be explained the way to combine all of them to achieve a new way to solve the *job Sequencing and tool Switching Problem*. In the figure 4.2 can be seen in a quick view the elements which will be used for the ACO implementation. Taking into account the different variables the reader will be able to understand the SSP treatment in the ACO framework.

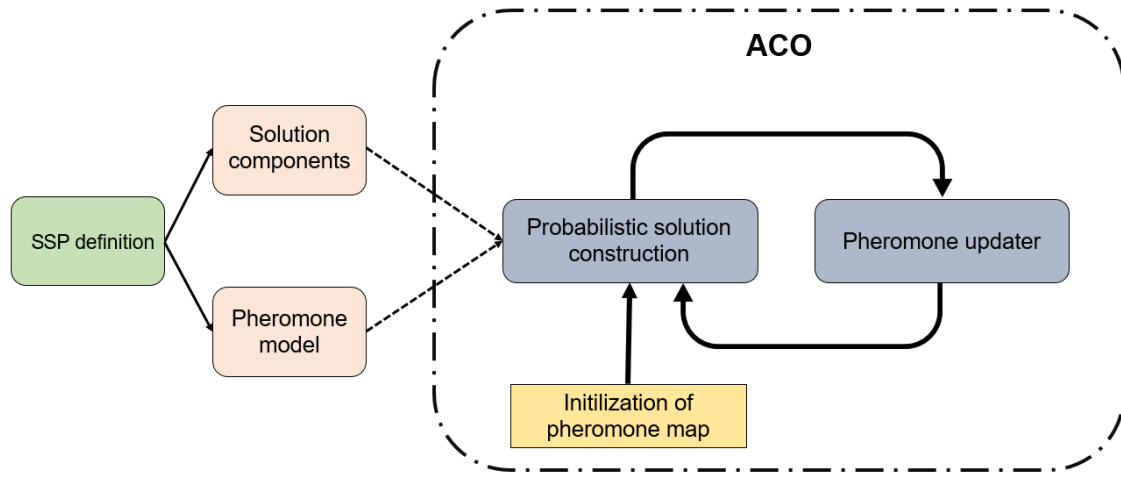


Fig. 4.2 ACO algorithm scheme

Starting from the SSP definition the ACO formulation treats all the jobs as nodes (as solutions presented in chapter 2) and will define a travelling salesman problem towards a graph with the different jobs as nodes and the calculation of the costs being made with the amount of tool switches for a giving sequence. The right tool distribution is extracted using the KTNS algorithm, defined theoretically in the subsection 4.2.1. The pheromone model is defined using a matrix with $N \times N$ dimension, where N is the number of jobs of the instance to be solved. The notation of the subsection 2.1.3 in chapter 2 will be kept during this ACO metaheuristic development. The matrix of the pheromone stores all the pheromone information and it is updated every iteration, being used to determine the random probability for the ants behaviour. In Dorigo & Stützle (2010) [19] is proven that for the ACO development the pheromone matrix can be initialized either with zero or non-zero values. Starting with a non-zero value helps to amplify the search area and to improve the results. If initial pheromone

value denoted τ_0 takes lower values the search is quickly biased by the first tours generated by the ants, which in general leads toward the exploration of inferior zones of the search space. On the other side, using higher initial pheromone values it will be need to do many iterations in order to wait for the pheromone evaporation. Later the τ_0 values will be defined as well the pheromone update, which correspond to the box "Pheromone updater" in figure 4.2. Depending on the pheromone model the initialization will be in one way or other one. Another feature defined in the initialization is the symmetry property of the pheromone map. Taking into account the characteristics of the SSP, the pheromone map matrix won't be symmetric due to the fact that $i \rightarrow j$ cost is not the same as $j \rightarrow i$ cost.

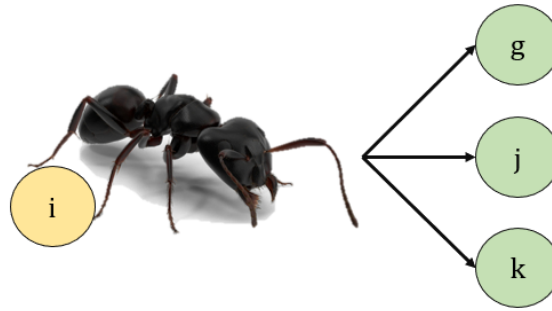


Fig. 4.3 Ant decision scheme

For the ACO explanation, in figure 4.3 it's shown an ant placed in node i and which have three different alternatives as next step of the sequence: g , j and k . In order to achieve the results the ant will smell the pheromone from the pheromone map and with the different weights α and β will establish a probability for selecting each candidate in the next step. Through a random selection will define the next step and the programming will update the pheromone map once finished the route from the nest to the food. The selection probability of the next node will be calculated using the random proportional rule 4.3.1.6, the pheromone global evaporation will be executed through the equation 4.3.1.5. Furthermore, the added pheromone values of the new ants will be updated with the equation 4.3.1.4 and will be included in the pheromone map. All these definitions correspond to the most basic ant colony optimization algorithm, the Ant System. The ant system can be summarized as:

Algorithm 3 Ant System

while termination conditions not met **do**

Schedule Activities

 AntSolutionConstruction(); { See Algorithm 4 }

 PheromoneUpdate()

 LocalSearchProcedure() { Optional }

end Schedule Activities

end while

In all the ACO variants the running times are controlled by a principal while-loops. These loops incorporate the finishing conditions, which can be either time or iterations for instance. In each iteration will be executed three algorithm components: *AntSolutionConstruction()*, *PheromoneUpdate()* and *LocalSearchProcedure()*. In the following lines each algorithm will be explained in detail, but first let be detailed in algorithm 4 the *AntSolutionConstruction()* procedure:

Algorithm 4 Procedure *AntSolutionConstruction()*

```

 $s = \langle \rangle$ 
Determine  $N(s)$ 
while  $N(s) \neq \emptyset$  do
     $c \leftarrow \text{Choose from}(N(s))$ 
     $s \leftarrow \text{extend } s \text{ by appending solution component } c$ 
    Determine  $N(s)$ 
end while

```

Running the procedures detailed in algorithm 3 the artificial ants are created and through the random probability will start to create the routes smelling the pheromone of the current pheromone map and depositing the new values of pheromone. As it has been explained previously, this random based probability adds heuristic elements as the pheromones and stochastic elements as the distance between nodes. Each of them will be weighted with the α and β , taking values for the ant system of $\alpha = 1$ and $\beta = 3$. Once the $N(s)$ set is empty, it will executed the *PheromoneUpdate()* algorithm. This procedure can be implemented in different kind of ways, establishing sub-types for the ACO metaheuristic. Depending of the pheromone treatment (in terms of evaporation and new values) we will face to one type of ACO metaheuristic or other one. For the evaporation coefficient of the ant system the proposed value of Dorigo & Stützle (2010) [19] is $\rho = 0,5$. Finally the last algorithm of the loop is the *LocalSearchProcedure()*, which allows to improve the best solution found using local search procedures. This step is optional for the ACO implementation, but including it the results obtained can be improved without using the metaheuristic, only applying local search methods. Let be noted that this explanation of the ant colony optimization correspond to the first designed metaheuristic, the ant system. This ant system was initialized with zero values in the pheromone map. The final version of the ACO metaheuristic will be presented in the next subsection, including the α , β and ρ used values.

4.3.3 Ant Colony System

In order to improve the most basic ACO model, the ant system, it was decided to improve the formulation developing a more complex type of metaheuristic. Dorigo & Stützle presented in 2010 the different types of metaheuristic which incorporated the ACO behaviour. Due the characteristics of the problem, it was decided to select a metaheuristic which could simplify the complexity, establishing as alternatives the elitist ant system and the ant colony system. After a study of the characteristics of the

problem it was decided to adopt the ant colony system.

First introduced by Dorigo & Gambardella (1997) [18], this metaheuristic differs from ant system in three main points. First, it exploits the search experience accumulated by the ants more strongly as the AS does through the use of a more aggressive action choice rule. Secondly, the pheromone deposit take place only on the arcs belonging to the best-so-far tour. The pheromone evaporation also can be modelled in the same way, as explained in Dorigo & Stützle (2010), and it can be updated only in the best-so-far-tour. For the SSP was proved that the pheromone should be updated only in the best-so-far tours, following the available literature. Updating only the best-so-far tour provides a simplification of the running procedure of the ACS in terms of complexity. As the final difference with the ant system metaheuristic, the ant colony system specifies the usage of local pheromone update. Each time an ant uses an arc (i, j) to move from city i to city j , it removes some pheromone from the arc to increase the exploration of alternative paths. During this subsection the three main changes will be explained for the attention of the reader. In terms of α , β and ρ values, the ant colony system adopted $\alpha = 1$, $\beta = 2$ and $\rho = 0, 1$.

In the ant colony system, when located at city i , ant k moves to a city j chosen according to the pseudo-random proportional rule given by:

$$j = \begin{cases} \arg \max_{l \in N_i^k} \{ \tau_{il} \cdot [\eta_{il}]^\beta \} & \text{if } q \leq q_0; \\ J & \text{otherwise.} \end{cases} \quad (4.3.3.1)$$

Where q is a random variable uniformly distributed in $[0, 1]$, q_0 ($0 \leq q_0 \leq 1$) is a parameter, and J is a random variable selected according to the probability distribution given by equation 4.3.1.6 with the α value presented above. In other words, with probability q_0 the ants makes the best possible move as indicated by the pheromone trails and the heuristic information, while with probability $(1 - q_0)$ it performs a biased exploration of the arcs. In Dorigo & Stützle (2010) is presented as a good value for the q_0 parameter the $q_0 = 0,9$, and for the metaheuristic development it was decided to use it.

Regarding the global pheromone trail update, in the ACS only one ant is allowed to add pheromone after each iteration. Thus, the update of the pheromone trails is done by the best-so-far ant using the following equation:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \rho\Delta\tau_{ij}^{bs} \quad \forall (i, j) \in T^{bs} \quad (4.3.3.2)$$

Where $\Delta\tau_{ij}^{bs} = 1/C^{bs}$. It is important to note that in ant colony system the pheromone trail update, both evaporation and new pheromone deposit, only applies to the arcs of T^{bs} , not to all the arcs as in ant system. Working with the best ant in each iterations provides a reduction in the computational complexity, reducing from $O(N^2)$ to $O(N)$, where N is the size of the instance being solved. The parameter ρ represents the pheromone evaporation and new pheromone value is weighted through the current value and the $\Delta\tau_{ij}^{bs}$.

Finally as third main change relative to the ant system, the ACS adds a local pheromone trail update, which will be applied immediately after having crossed the arc (i, j) during the tour construction of

every ant. This local update is modelled as follows:

$$\tau_{ij} \leftarrow (1 - \varepsilon)\tau_{ij} + \varepsilon\tau_0 \quad (4.3.3.3)$$

Where ε is a value $\varepsilon \in (0, 1)$, and τ_0 is the initial value of the pheromone map. The ε experimentally has been defined as 0,1, presented in Dorigo & Stützle (2010). Unlikely the ant system, in ant colony system the pheromone map is initialized with a τ_0 value, equal to $1/nC^m$. The n is the number of cities in SSP instance (in our case the number of jobs) and C^m is the length of a nearest-neighbour tour. For the formulation of the SSP it was determined for the C^m a value of the multiple start greedy algorithm presented in the subsection 4.2. Through the execution from an initial random point to achieve one first for the τ_0 definition. The effect of the local updating rule allows that each time an ant uses an arc (i, j) its pheromone trail τ_{ij} is reduced, so that the arc becomes less desirable for the following ants. In other words, allows an increase in the exploration of arcs that have not visited yet and, in practice, has the effect that the algorithm does not show a stagnation behaviour. In order to avoid disruptions during the calculations of the η value achieving $d_{ij} = 0$, it was decided to replace d_{ij} for a lower value of 0,00001. Let be reminded that all the distances between the jobs represented by the nodes are calculated using the *Keep Tool Needed Soonest* algorithm, presented in subchapter 4.2.1.

4.3.4 Local search for ACS

Once explained the ant colony system formulation towards the SSP solution it will be defined local search procedures to improve previous results. As detailed in algorithm 3, the local search procedures are optional and they allow to improve the best solution. The fact of adding local search procedure enhance the quality of the solutions and for the case of the SSP can establish an improvement.

Local search starts from some initial assignment and repeatedly tries to improve the current assignment by local changes. If in the neighbourhood of the current tour T a better tour T' is found, it replaces the current tour and the procedure continues from T' . The most widely known iterative improvement algorithms for TSP, which can be easily applied to the SSP modelling, are certainly 2-opt and 3-opt. Due to the similarities between SSP and TSP (in terms of problem modelling), the local search procedures of TSP can be adapted to the SSP. 2-opt and 3-opt proceed by systematically testing whether the current tour can be improved by replacing 2 or at most 3 arcs, respectively. Both local search algorithms are widely studied in the literature [14, 33], and have been shown empirically to yield good quality solutions. Local search algorithms using $k > 3$ arcs to be exchanged are not used commonly, due to the high computation times involved and the low additional increase in solution quality. From 2-opt to 3-opt a straightforward implementation would require $O(n^2)$ or $O(n^3)$ exchanges to be examined. Depending on the time availability the local search will be applied in the metaheuristic or not.

Fortunately, there exist few speed-up techniques, which use only a small part of the whole neighbourhood. These techniques allow to enhance times for the local search procedure, as commented in

Johnson & McGeoch (1997) [30]. One example which would increase the quality of the solution is the restriction of the movements only for the nearest neighbours. Towards a candidate list ordered according to non-decreasing distances, for a given node i we only consider moves which add a new arc between i and one of the nodes in its candidate list. This technique would improve the previously obtained results. An alternative procedure would be the addition of a radius of nearest neighbour search. This radius would define a local neighbourhood and would allow to reduce the times needed to look for a better solution. Different strategies have been developed in order to improve the times in the local search procedures, and for the SSP characteristics the fact of designate a local search procedure would be a good adding if the available time allows the execution of the local search. In the computational results of chapter 5 the application of the local search for the ACS will be detailed.

Chapter 5

Computational results

5.1 Introduction

In this chapter the computational results obtained with the metaheuristics explained in chapter 4 will be analysed in order to prove that can suppose feasible alternatives to previous solutions of the SSP. Analysing the behaviour of each metaheuristic (GRASP and ACS) it will be proven how closer are the procedures with the best solutions known.

In order to test the performance of the different formulations, it was considered to use the datasets described in Laporte et al 2004 [32], downloadable **clicking here**⁷. In the appendix B the different datasets are defined in terms of amount of jobs/tools, capacity of the magazine and some other features. All the formulations were executed in the same machine, a laptop *HP 15-ay117ns* with core i7 and 8 gigabyte of ram. Regarding the running procedure, all the metaheuristics will be executed the same amount of time. Once obtained all the results they will be compared each other in order to submit a metaheuristic in a feasible time for the *job Sequencing and tool Switching Problem*. In order to settle a mutual aspect for the comparison of the two metaheuristics developed, it was decided to develop a random search procedure. Each of the three alternatives were executed the same amount of time, determined as:

- **A datasets:** 5 seconds.
- **B datasets:** 15 seconds.
- **C datasets:** 100 seconds.
- **D datasets:** 250 seconds.

During the next section will be defined the random search strategy. Let be explained that the GRASP strategy was adopted in early beginnings in the project, trying to achieve good quality solutions in a feasible time. The initial *Keep Tool Needed Soonest* algorithm was optimized during the project

⁷<https://bit.ly/2LMcY1s>

development, achieving the fastest results in the last stages. For the comparison has been used for the two metaheuristics the same KTNS, which runs forty times faster than the first version used for GRASP first results. Comparing the GRASP and the ACS results with a random based strategy it will be noticed how the metaheuristics work in the SSP framework.

5.2 Random Search Procedure

Setting a random basis it can be compared how the metaheuristics manage heuristic information in order to stay closer to the optimal solution. Comparing the results of the developed metaheuristics with a random search based on the use of the KTNS algorithm it can be noted how the solutions are relative to the others procedures. The random search algorithm was developed with the following statements:

- Generate a random sequence based on the jobs available for the dataset.
- Compute the cost for the random sequence generated with the KTNS procedure.
- Respect the amount of time established as basis for the comparison.

The obtained results for the different datasets are presented in the following table:

Dataset	Time (s)	Average solution obtained	Average best known solution	Average deviation
A1	5	12,50	12,5	0,00%
A2	5	10,80	10,8	0,00%
A3	5	10,10	10,1	0,00%
A4	5	10,00	10	0,00%
B1	15	29,53	26,9	9,79%
B2	15	23,27	22	5,76%
B3	15	20,13	19,8	1,68%
B4	15	19,23	19,2	0,17%
C1	100	126,70	102	24,22%
C2	100	107,50	85,9	25,15%
C3	100	85,60	69,4	23,34%
C4	100	62,67	53,6	16,92%
D1	250	256,97	203,2	26,46%
D2	250	228,27	179	27,52%
D3	250	193,47	152,5	26,86%
D4	250	151,90	120,9	25,64%

Table 5.1 Random search simulations results for the different datasets

Let be defined the columns of the table 5.1 as:

- Time. Amount of time used for the simulation of each instance of the dataset. Defined in advance.
- Average solution obtained. Average solution obtained for each dataset and contained instances.
- Average best solution known. Average of the bests solutions known taking into account all the instances inside each dataset.
- Average deviation. Average deviation between the average solution obtained and the best solution known. The best solution known is the one presented by Laporte et al. in 2004 publication.

The results obtained showed how for the smaller datasets the averages deviations obtained were small, even achieving best average solution in the dataset B4. Taking into account the average of the best solutions can be seen how in all the cases the difference regarding with the best solution known is very small. For bigger instances, instead, the average deviation increases to values which are not assumable.

As it can be seen the fact of developing a random search procedure towards the solution of the SSP doesn't provide good results. This facts leads to the idea of applying heuristic information towards the improvement of the usage of the resources. Getting an average deviation of 25% in the D datasets relative with the best solutions known, the random search results were categorized as a bad alternative for the SSP solution. It was known beforehand that applying a random search procedure bests results couldn't be achieved, but for the metaheuristic comparison it was needed.

5.3 GRASP results

During the first attempt towards the resolution of the *job Sequencing and tool Switching Problem* it was tried to solve the SSP through the GRASP metaheuristic. Getting good quality solutions in a feasible time was mandatory, and to know the scope of the problem it was defined a strategy which wanted to measure the time used to carry out one iteration for each dataset.

Let be explained that the most part of the time used in the GRASP metaheuristic correspond to the calculation of the costs for a sequence, hence the improvement the KTNS algorithm leads to improve the running times of the metaheuristic. In order to know the times used to the GRASP iteration, in the table 5.2 is summarised the simulated times needed for one iteration in each dataset:

Dataset	Average time (ms)
A1	27
A2	35
A3	41
A4	31
B1	108
B2	182
B3	151
B4	73
C1	2.491
C2	2.207
C3	2.287
C4	1.770
D1	12.052
D2	14.900
D3	8.471
D4	7.705

Table 5.2 GRASP averages times for one iteration

As it can be seen for the biggest datasets (D) the running times are quite big relative the A datasets. This is due to the fact of executing the KTNS algorithm lots of times and trying all the possible combinations, as explains the multiple start greedy procedure. For instance, one iteration for the GRASP metaheuristic in the A datasets implies 405 executions of the KTNS, a low value if it's compared with the number of executions for a instance of the D dataset: 30.420. Dealing with an amount of ten jobs and ten tools in A datasets less resources are needed than for the D datasets: based on forty jobs and sixty tools. Due to the available time it was considered to simulate as much iteration as possible, and compare the solutions obtained with the best solutions known with the instances defined in appendix B. In the following table 5.3 are presented the results:

Dataset	Time (s)	Average solution obtained	Average best known solution	Average deviation
A1	5	12,50	12,5	0,00%
A2	5	10,80	10,8	0,00%
A3	5	10,10	10,1	0,00%
A4	5	10,00	10	0,00%
B1	15	27,70	26,9	2,97%
B2	15	22,07	22	0,30%
B3	15	19,80	19,8	0,00%
B4	15	19,20	19,2	0,00%
C1	100	112,67	102	10,46%
C2	100	94,87	85,9	10,44%
C3	100	74,77	69,4	7,73%
C4	100	55,47	53,6	3,48%
D1	250	223,47	203,2	9,97%
D2	250	196,40	179	9,72%
D3	250	165,47	152,5	8,50%
D4	250	129,37	120,9	7,00%

Table 5.3 GRASP simulations results for the different datasets

The GRASP metaheuristic simulation results were better than the obtained within the random search methodology. For the datasets C and D the results improved to average deviations of less than 10 % in the majority of the cases.

The first attempt towards the metaheuristic development for the *job Sequencing and tool Switching Problem* was considered as insufficient due to not being able to endure with the bigger size instances. The times needed to achieve lower percentage of deviation relative to the best solutions known in the datasets A and B were enough to achieve low percentages of deviations, but for C and D they weren't enough.

5.4 ACS results

After analysing the obtained results for the random search and for the GRASP metaheuristic, the results of the ant colony system will be shared. As commented in chapter 4, two variables must be settled for the ACS execution: the number of ants and the number of iterations. Due to the fact that the time is limited, it can be decided to set the ants or the iterations. It was decided to establish the number of ants, setting an amount of ten ants. The number of iterations was defined for the top bound of time, doing as much as possible iterations accomplishing the time constraint. In the subsection 5.4.1 it is shown how the number of iterations allows to an ant achieving better solutions. In the table 5.4 can be seen the summary of the results obtained for the different running times:

Dataset	Time (s)	Average solution obtained	Average best known solution	Average deviation
A1	5	12,50	12,5	0,00%
A2	5	10,80	10,8	0,00%
A3	5	10,10	10,1	0,00%
A4	5	10,00	10	0,00%
B1	15	27,80	26,9	3,35%
B2	15	22,00	22	0,00%
B3	15	19,80	19,8	0,00%
B4	15	19,20	19,2	0,00%
C1	100	109,40	102	7,25%
C2	100	92,30	85,9	7,45%
C3	100	73,20	69,4	5,48%
C4	100	54,60	53,6	1,87%
D1	250	223,40	203,2	9,94%
D2	250	196,80	179	9,94%
D3	250	168,70	152,5	10,62%
D4	250	131,30	120,9	8,60%

Table 5.4 ACS simulations results for the different datasets

In nearly all datasets the obtained average deviations are smaller than a 10 %, and in some of the cases the results obtained are better than the best solution known. The list of the improved datasets is:

- Dataset B1. The instance five improved from a best known value of 27 to 26.
- Dataset B2. The instance four improved from 21 to 20.
- Dataset C4. The instance five improved from 54 to 52. The instance number nine improved from 56 to 55.

In order to improve the results obtained for C and D datasets it was decided to incorporate some more characteristics to the ACS. These could guide to the usage of more time resources but instead the results could be improved. The first feature was resetting the pheromone map with the initial value every fifty iterations. This allows the ants to increase the quality of the solutions exploring other arcs and deleting all the previous information. The other feature added was the local search procedure in the ACS, described theoretically in subsection 4.3.4. The local search could guide to use more time resources but instead the solutions could improve and get more closer to the optimal ones. This local search it was decided to be applied every time that an ant achieved the best solution for the route. Based on 2-opt, the procedure tries to exchange two jobs of the sequence in order to enhance results. The number of times applied was settled as twenty-five. The last feature is about the default value of the pheromone map. In the first steps of the ant colony system it was decided to run a multiple

start greedy in order to set the value. This multiple start greedy was executed only one time from one random chose initial job. After checking the velocity of the GRASP metaheuristic to execute one iteration (see table 5.2), it was decided to include this feature to the design of the ACS metaheuristic. After include the three commented features the results obtained with the ACS were:

Dataset	Time (s)	Average solution obtained	Average best known solution	Average deviation
A1	5	12,50	12,5	0,00%
A2	5	10,80	10,8	0,00%
A3	5	10,10	10,1	0,00%
A4	5	10,00	10	0,00%
B1	15	27,37	26,9	1,73%
B2	15	22,00	22	0,00%
B3	15	19,80	19,8	0,00%
B4	15	19,20	19,2	0,00%
C1	100	108,50	102	6,37%
C2	100	90,77	85,9	5,67%
C3	100	72,50	69,4	4,47%
C4	100	54,17	53,6	1,06%
D1	250	217,80	203,2	7,19%
D2	250	190,97	179	6,69%
D3	250	161,03	152,5	5,60%
D4	250	126,63	120,9	4,74%

Table 5.5 ACS simulations results with improvements

It can be seen how the addition of new features as the local search or the reset of the pheromone map allow to enhance the results shown in the previous table 5.4. The table 5.5 shows a notable improvement to the results obtained with the random search and sets a good quality solution in a feasible time for the *job Sequencing and tool Switching Problem*.

5.4.1 ACS behaviour

The top difficulty in the ant colony system framework is to predict how many ants or iterations are needed for each instance in order to achieve good solutions without stagnation. In this subsection it will explained with a real case how the ants work to get the optimal value for the sequence and the distribution of the tools. This simulation was done using only one ant per iteration, and through the usage of different amounts of iterations it will be shown how increasing the number of iterations the pheromone map is modified. Using only one ant per iteration allows to perceive the movement of the ant through the different nodes while the pheromone values are smelled. The instance solved is

the number 1 of the dataset *B1*, which currently best value known of tool switches is twenty-five. In the last part of the subsection can be seen the final distribution of the tools for the sequence obtained using the ACS metaheuristic.

Starting with one iteration, the results obtained were:

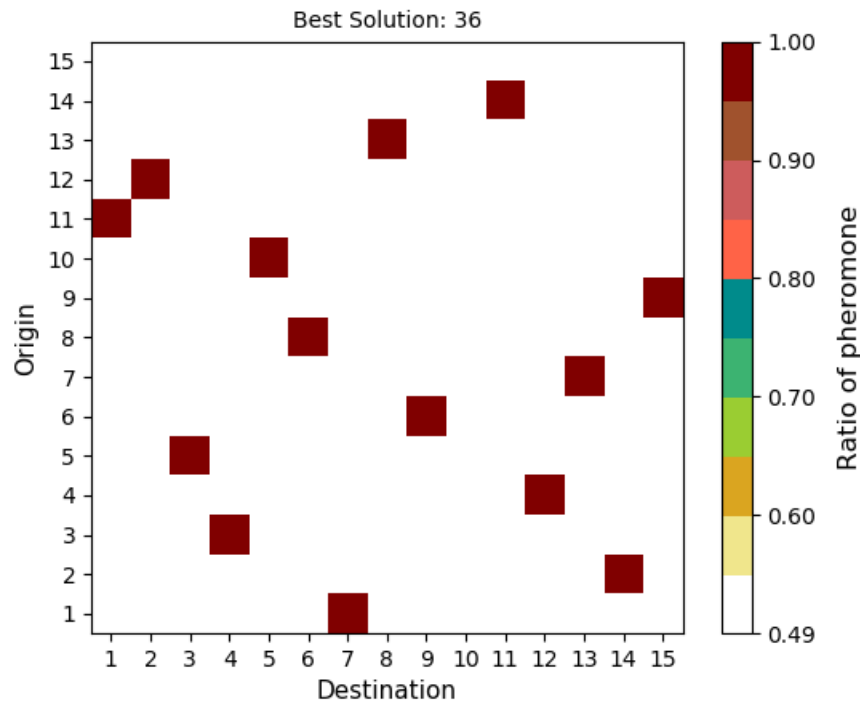


Fig. 5.1 Pheromone map obtained with one iteration

The best solution achieved (also the unique) was 36, as it is shown on the top of figure 5.1. In the right part of the heat map the legend shows the ratio of the pheromone values. This ratio is calculated through the pheromone map obtained in the number of iterations calculated divided by the most larger value of the pheromone matrix. Establishing the differences between nodes can be easily seen. In the y axis it's represented the origin nodes, and in the x axis is represented the destination nodes. Viewing the relationship in terms of ratios of pheromone between origin and destination can be seen how the balance between the heuristic and stochastic information guides to quality solutions. Remember that this balance was established using the α and β variables.

Let be reminded that for the SSP formulation the jobs are considered as jobs. Taking into account that the pheromone map is initialized with default values τ_0 can be seen how one iteration modify the map in terms of ratios. Increasing the iterations the ant tries to populate the pheromone in order to achieve better quality results than the ones already achieved. In the following pictures can be seen how the pheromone map evolves meanwhile the iterations are modified:

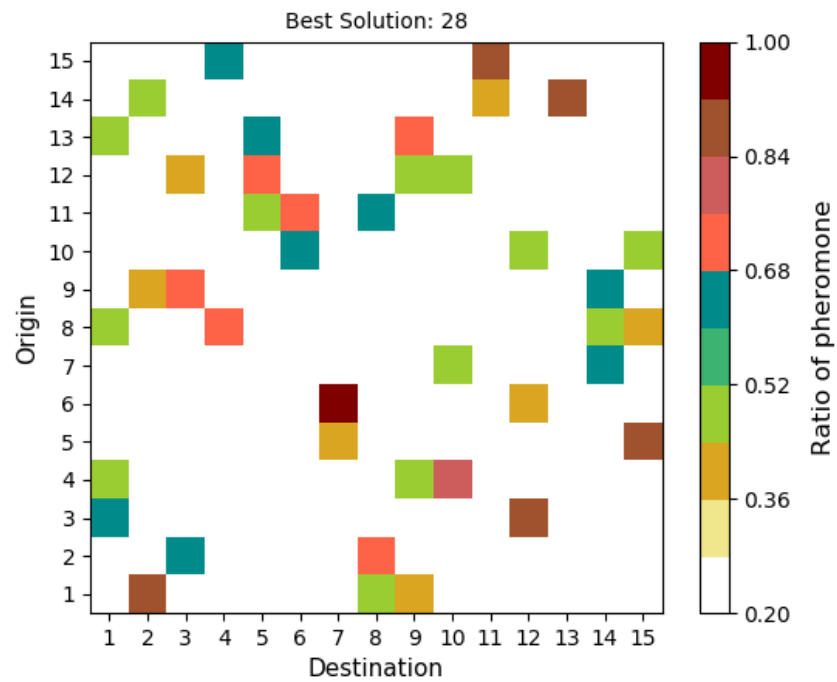


Fig. 5.2 Pheromone map obtained with five iterations

As it can be seen the results improved only using five iterations and while the ACS is running more iterations the results are getting better, due to the previous pheromone trails. Increasing the iterations to fifteen it is obtained:

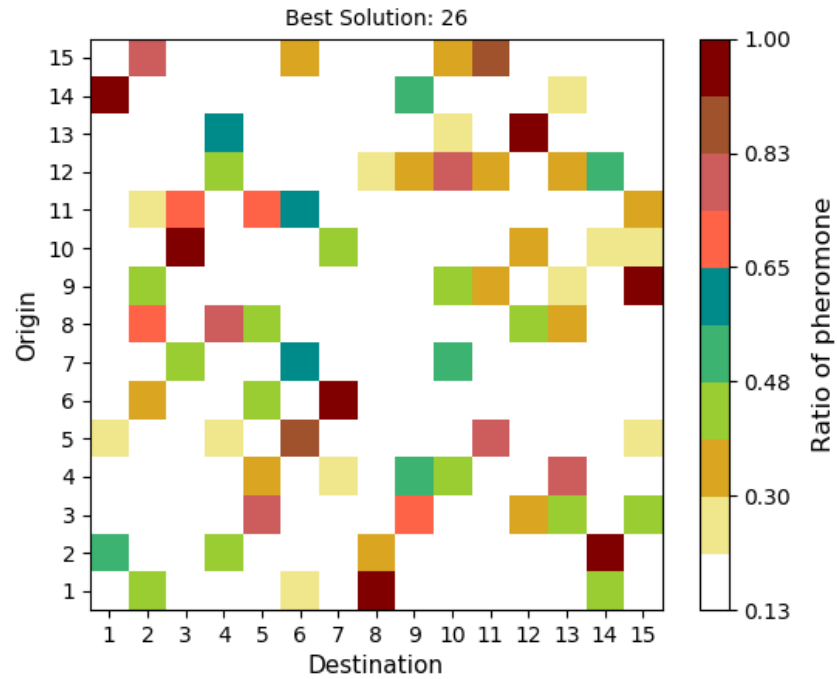


Fig. 5.3 Pheromone map obtained with fifteen iterations

The ant is close to the best known value and finally, increasing the number of iterations to fifty, the result obtained is the best value known. Although the value of twenty-five starts to appear from the thirty iteration it was determined to show the pheromone map of the fifty iteration to see the ant effect, trying to improve the previous obtained results. The obtained pheromone map is:

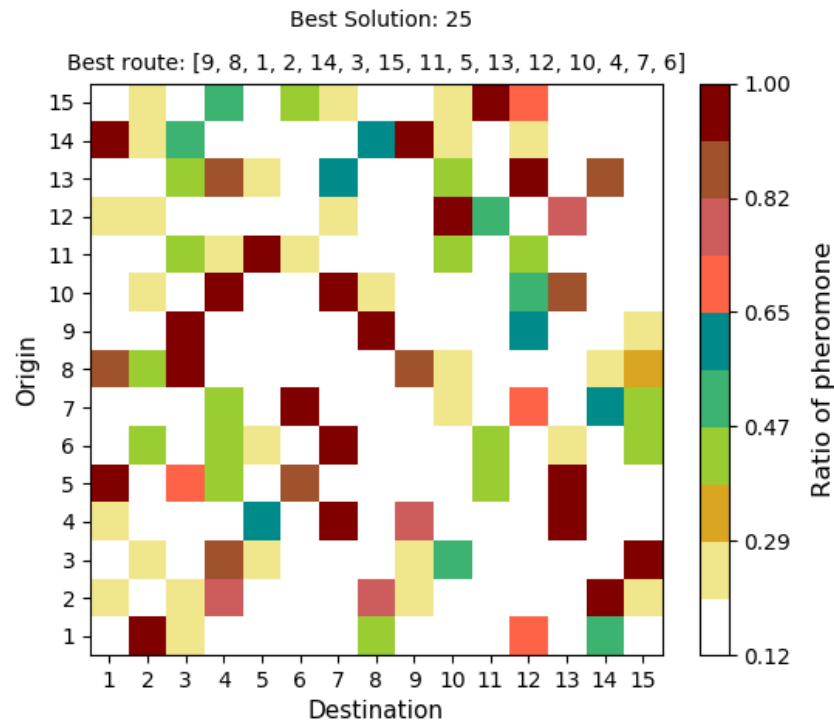
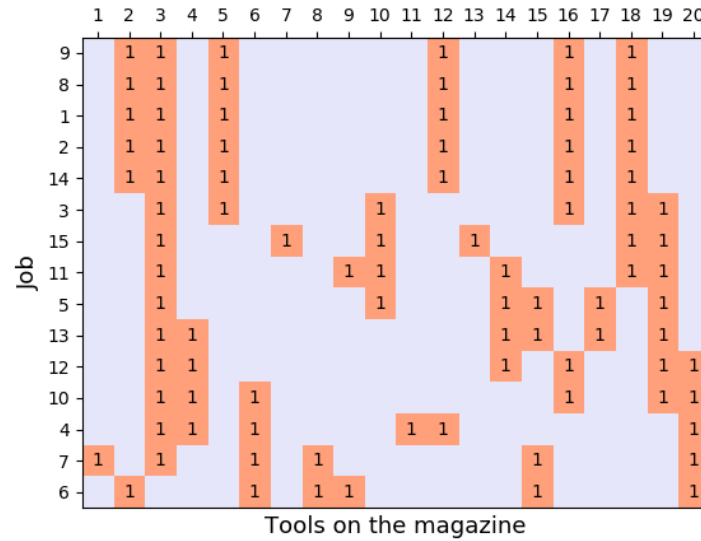


Fig. 5.4 Pheromone map obtained with fifty iterations

Executing ant colony system with one ant and fifty iterations took a total amount of running time of 0,33 seconds, achieving the best value known of tool switches for this dataset. The behaviour of the metaheuristic allows to achieve the most likely best route using heuristic information in less time than the others. Due to a lower complexity value the calculation procedure needs less time to calculate more possible solutions.

On the top part of the figure 5.4 can be seen the best sequence found, which guides to the achievement of the value of twenty-five switches. The capacity of the magazine is included in the total amount of changes, as explained in subsection 2.1.2. During all the procedure all the calculations of the cost and tool distributions are calculated with the *Keep Tool Needed Soonest* policy, explained in subsection 4.2.1. Once obtained the best route, the tool distribution map which minimize the tool switches is:

Fig. 5.5 Distribution of the tools for the σ sequence found

The y axis indicates from up to down the best sequence found for the different jobs, in order to minimize the tool switches. The x axis indicates the different tools available for the dataset proposed and the ones inside the matrix indicate when the tools should be charged on the magazine. It can be seen how the KTNS algorithm keeps as much as possible the vertical alignment of the tool usage, in order to minimize the switches between one job and the next one. For this dataset the capacity of the magazine is five, providing space for five tools in each job. All the capacity is always used as stated in the assumptions subsection 2.1.2, guiding in the matrix to get five ones per row.

5.5 Comparing metaheuristics

Once explained the obtained results for the different procedures they will be compared to asses which of the developed metaheuristics (GRASP or ACO) provide more quality results in a feasible time. In order to contrast them, the compared feature will be defined as the difference relative with the best solutions known for each instance inside each dataset: A, B, C and D. Taking into account that in the datasets A and B the best results are obtained in most of the cases, the comparison will be carried out mainly in the blocks C and D. The results obtained will be plotted in a box plot, allowing to see in a graphical way the possible divergences between GRASP and ACS. The links to the interactive box plots will be shared for the information of the reader. In the following table the deviations for each procedure are shared together in order to see the differences between each procedure:

Dataset	Time (s)	Random Search	GRASP	ACS
A1	5	0,00%	0,00%	0,00%
A2	5	0,00%	0,00%	0,00%
A3	5	0,00%	0,00%	0,00%
A4	5	0,00%	0,00%	0,00%
B1	15	9,79%	2,97%	1,73%
B2	15	5,76%	0,30%	0,00%
B3	15	1,68%	0,00%	0,00%
B4	15	0,17%	0,00%	0,00%
C1	100	24,22%	10,46%	6,37%
C2	100	25,15%	10,44%	5,67%
C3	100	23,34%	7,73%	4,47%
C4	100	16,92%	3,48%	1,06%
D1	250	26,46%	9,97%	7,19%
D2	250	27,52%	9,72%	6,69%
D3	250	26,86%	8,50%	5,60%
D4	250	25,64%	7,00%	4,74%

Table 5.6 Results of simulations for Random search, GRASP and ACS

Let be explained that the ant colony system results in table 5.6 are the ones belonging to the improved version, with the new features. In the following subsections every block of dataset will be analysed in order to know which solution provides better results.

5.5.1 Dataset A

For all the instances of the dataset A the best value known is achieved and there is no difference between the different types way to solve them. The best known results are not improved in any of the methods and the procedures achieve in the established time the best values known. Due to the equality between methods, conclusions about which metaheuristic perform better can't be done with this dataset.

5.5.2 Dataset B

In the solution of the second dataset the differences between each procedure are more visible than for the first one. The random search achieves nearly all the best solutions (0,17 % deviation) in the fourth block of the dataset and for the first and the second blocks the average deviations are greater than 5 %. In the other hand, GRASP and ACS metaheuristics achieve better results than the random search, even improving the best solutions known, as commented in section 5.4. Through the statistical

comparison it can't be determined which of the two alternatives achieves better solutions due to the similar results obtained. The election will be done using the results for datasets C and D. Through the B dataset, can't be determined which metaheuristic achieves better results, only can be extracted that GRASP and ACS performed better than the random search

5.5.3 Dataset C

For the dataset C the obtained results are plotted in the following figure:

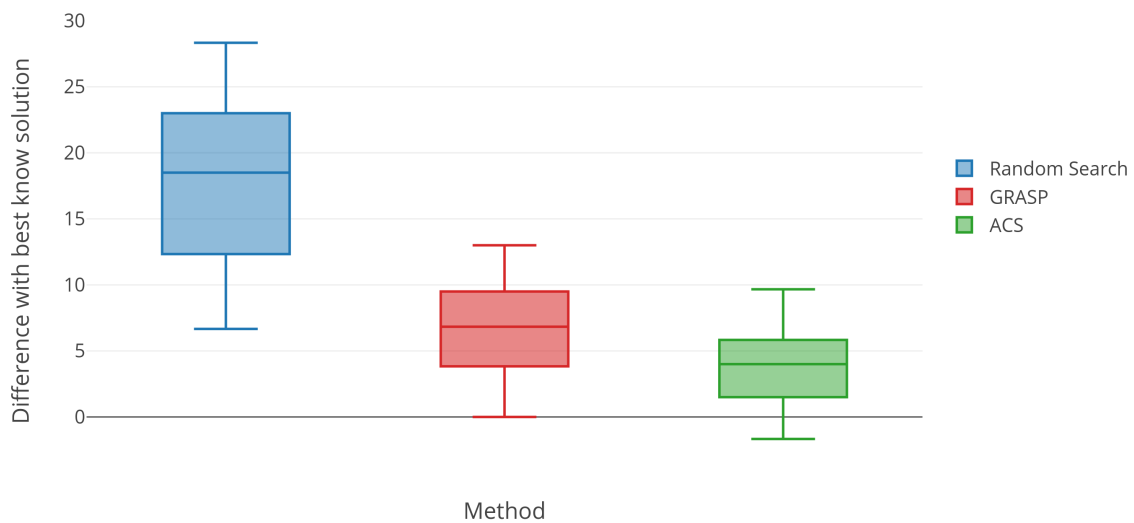


Fig. 5.6 Obtained box plot for the dataset C

The graphic information can be accessed via internet [clicking here](http://bit.ly/2J9Xnvb)⁸. To know which of the two developed metaheuristic obtained better results, it must be done a *Wilcoxon Signed-Rank Test*, because the differences between the two metaheuristic don't provide a clear result of which one obtains better results. Due to the differences between the two metaheuristic results this procedure can be applied, unlike B dataset. This test allows to set which of the two options is clearly better than the other one, and it's based in the ranks of two related samples. The hypothesis sustained by this procedure is that the matched samples have different mean ranks. Using the test was determined that for the dataset C the best solution developed was ACS, in terms of getting a lower median value for the differences with the best solutions known.

⁸<http://bit.ly/2J9Xnvb>

5.5.4 Dataset D

As done for the dataset C, the box plot of the results obtained is the following:

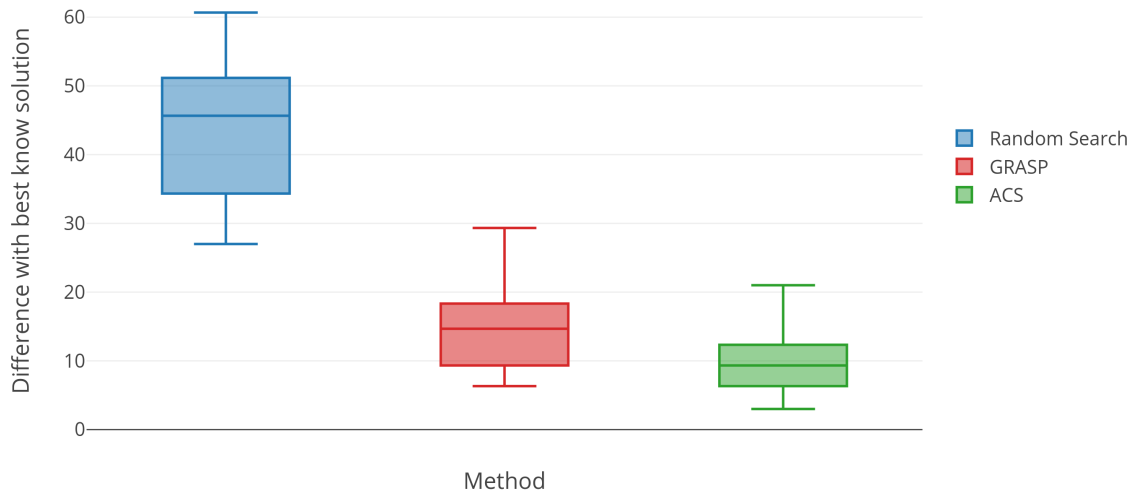


Fig. 5.7 Obtained box plot for the dataset D

The information can be accessed via internet **clicking here**⁹. For this dataset it can be clearly seen how the ant colony system metaheuristic obtains better results than the greedy based metaheuristic. In the same amount of time the results of the ACS are better than the GRASP. The fact which leads to achieve better solutions is the usage of the heuristic information in order to improve the previous obtained results. In terms of complexity, the ant colony system enhanced the calculations done by the GRASP metaheuristic.

5.6 Conclusions computational results

After a comparative view between the two developed metaheuristic the mathematical conclusions will be shared in this section. The GRASP based metaheuristic provided a starting point towards the achievement of a good quality solution in a feasible time. The main issue of this kind of metaheuristic was indeed the fact of iterating from all the jobs and trying to achieve a best sequence, providing poor results for big size instances. The complexity of the metaheuristic is high, and the usage for the *job Sequencing and tool Switching Problem* didn't guarantee good results for all the cases.

In the other hand the ant colony optimization metaheuristic worked with a lower complexity, allowing to get better results in the same amount of time. The way of working of the ant colony system allowed to incorporate new features in order to improve the results, as the reset of the pheromone map, the local search procedures or the addition of the multiple start greedy to set the initial values of the pheromone map. All these features combined improved the results of the initial version of the ant

⁹<http://bit.ly/2LWFgqe>

system, the first known ACO metaheuristic.

Through the statistical analysis done in section 5.5, it can be seen that the best developed solution for the *job Sequencing and tool Switching Problem* is the ACO based metaheuristic. The values obtained in the times established are the best ones, and in order to try to improve the results obtained it was decided to run the ant colony system with 10 ants and 300 iterations, a big amount of iterations relative with the number of iterations needed to accomplish the times established as the basis of the comparison. It was decided to use an amount of iterations instead an amount of time to keep resetting the same amount of times the pheromone map for each instance. The results obtained can be summarised in the following table:

Dataset	Time (s)	Average solution obtained	Average best known solution	Average deviation
A1	14,32	12,50	12,5	0,00%
A2	11,42	10,80	10,8	0,00%
A3	11,75	10,10	10,1	0,00%
A4	10,23	10,00	10	0,00%
B1	39,93	27,10	26,9	0,74%
B2	35,01	21,80	22	-0,91%
B3	32,59	19,80	19,8	0,00%
B4	30,18	19,20	19,2	0,00%
C1	465,49	106,50	102	4,41%
C2	443,70	89,80	85,9	4,54%
C3	413,62	71,60	69,4	3,17%
C4	357,16	53,60	53,6	0,00%
D1	1732,51	215,40	203,2	6,00%
D2	1685,33	189,20	179	5,70%
D3	1592,26	159,70	152,5	4,72%
D4	1549,72	125,40	120,9	3,72%

Table 5.7 ACS results when extending time

These results shown how the fact of extending times provides the possibility of getting better solutions. A summary of the data is the following:

- 52,5 % of the instances achieved the best value known or improved it.
- For the larger datasets C and D the results were less than 6 % in terms of average deviation.
- Maximum running times of 1732,51 seconds, nearly 29 minutes.
- 7 instances of datasets B and C were improved:

- From B1 dataset the instances 5 and 8.
- From B2 dataset the instances 3 and 4.
- From C4 dataset the instances 5, 7 and 9.

Before closing the computational numerical results let be explained that the sequences obtained through the different methods can be accessed **clicking here**¹⁰, along with the best value of tool switches for the obtained sequence.

5.7 Comparison with current state of the art

Regarding the objectives established for the project, in this section the achieved metaheuristic will be compared with the current state of the art in order to know the quality of the solution. The results for the Laporte et al. instances are shared at the end of Ahmadi et al. (2018) [1] publication. The obtained table is:

Dataset	Current state of the art by Ahmadi et al.	ACS metaheuristic average results	Best known value by Laporte et al.
A1	12,5	12,5	12,5
A2	10,8	10,8	10,8
A3	10,1	10,1	10,1
A4	10,0	10,0	10
B1	26,5	27,1	26,9
B2	21,7	21,8	22
B3	19,7	19,8	19,8
B4	19,2	19,2	19,2
C1	99,1	106,5	102
C2	82,8	89,8	85,9
C3	66,8	71,6	69,4
C4	51,5	53,6	53,6
D1	198,4	215,4	203,2
D2	174,0	189,2	179
D3	146,7	159,7	152,5
D4	115,4	125,4	120,9

Table 5.8 Comparison with current state of the art

As it can be seen the results obtained by the developed metaheuristic don't achieve the same values than the current state of the art presented by Ahmadi et al. In A and B datasets the results

¹⁰<http://bit.ly/2HkGwjC>

obtained are close with the state of the art but for bigger instances the gaps start to grow. The proposed solution can't suppose a statement due to the lower quality solutions relative with the state of the art. Even though the computational results of the ACS metaheuristic are considered successful, they are not enough to suppose an improvement to the current best solution for the SSP. Nevertheless with the development of an alternative solution the ACS can be combined with other procedures, in order to improve the obtained results.

5.8 Plotting solutions

With the aim to develop a tool which could provide a sequence and an optimal distribution for the tools needed in each step it was decided to develop a python based program. This plotting tool can provide the results for the different datasets and instances presented by Laporte et al. (2004) [32]. The instructions and the features of the applications can be seen in the appendix C. **Clicking here**¹¹, it can be downloaded and executed if all the modules required are installed on the machine. Once unzipped all the files, the user can simulate the instances proposed by Laporte et al. With some plots of the solution it will be visible the ACS contribution in the SSP framework.

5.9 Python coding

All the coding developed for the simulations can be downloaded **clicking here**¹². Inside the file there are the three developed programs which provide all the simulated data presented in this chapter. All the algorithms are limited in terms of time for the comparison and the results are stored in external folders. The proposed solutions shared the KTNS coding and the data collector, which processes all the information inside the instances samples.

¹¹<https://bit.ly/2J8OIKy>

¹²<https://bit.ly/2LQvTYT>

Chapter 6

Conclusions

Once the computational results are analysed, obtained by the greedy based (GRASP) and ant colony system (ACS) metaheuristics, the main conclusions of the project can be extracted. Starting with a problem statement and taking the assumptions of the precedents in the literature, it was decided that to solve the *job Sequencing and tool Switching Problem* proposing another strategy. An alternative way would provide better solutions than the ones already existent and could settle a new basis for future works. Taking into account the computational complexity of the problem and the capability to model it as a travelling salesman problem, the calculation of the tool distribution was decided to be done using the *Keep Tool Needed Soonest* algorithm, already existent in precedents solutions.

The first attempt towards the metaheuristic development was done with a greedy one, in order to try all the possible sequences which could guide to good quality solutions. As defined in this document, the main issue of these kind of metaheuristics is the high complexity, which guides to the waste of resources even without achieving top quality results. This issue is shared with most of the precedents in the literature, and the hard part of the project was to manage the bigger instances. For the datasets presented by Laporte et al. (2004) [32], to deal with amounts up to forty jobs and sixty tools wasn't realistic using a greedy metaheuristic. The integer linear programming and the integer non-linear programming couldn't provide a quality solution for all the datasets defined, due to the management of constraints with big amounts of data. In numeric terms, the greedy metaheuristic combined with the *Keep Tool Needed Soonest* algorithm achieved deviations lower than 10 % with the best solutions known in some of the cases, but it was considered to reduce the problem complexity. The reduction was done by using another strategy, based on the ant's behaviour. Between all the possibilities of the ant colony optimization, it was decided to start with the more basic structure: the ant system. Although the initial results weren't better than GRASP, evolving the metaheuristic to the ant colony system provided a lower complexity for the calculations, improving the usage of resources. Taking into account that for most part of the time, the metaheuristic is executing the *Keep Tool Needed Soonest* algorithm, the need to improve the programming to achieve higher velocities became a necessity. The initial version of the KTNS algorithm worked forty times slower than the final one. Doing improvements in KTNS coding, the running times for the ant colony system were reduced,

allowing to add local search procedures and other features. These features were the reset of the pheromone map and the improvement of the initial values of the pheromone map, τ_0 . With all these additions the obtained results were considered better than the GRASP results, for all the datasets. Although the GRASP results in some of the cases were nearly the same as the ACS metaheuristic, the computational results for all the datasets provided the evidence that the ant colony system achieved better values.

Comparing the results with the current state of the art can be seen how the proposed ACS metaheuristic can't establish a new basis, due to the differences in the obtained results. Meanwhile the proposed solution can't improve the instances which belong to C and D datasets, the current state of the art can achieve negative average deviations for most of C and D datasets. Nevertheless, the fact of not establishing a new trend in the SSP framework doesn't mean that the project needs to be catalogued as a failure. Instead, the ACS development allows to open a new scenario in the SSP framework in which the ant colony optimization is introduced as an alternative, even not achieving the best known values. The early next step which should be done from this point would be improving the obtained solutions to get closer to the current state of the art. Combining the ACS with other heuristic tools or changing the local search procedures the proposed metaheuristic will achieve better results than the ones presented in this document. Thinking about future works, another next step would be the adaptation of the problem framework to a real case, with real data. Enabling industries to improve their processes would be the final aim of this project. The main problem of this adaptation is how to deal with real constraints, as different tuning times or order restrictions. By working with constraints and variable costs the problem definition would be more complex but with the presented metaheuristic the more basic model is settled. Another line for future studies is the enhance of the velocity of the ant colony system and local search procedures. The current local search procedure allows the improvement of the results, but making an effort in order to improve it would allow the whole improvement of the metaheuristic. Apart of these improvements, the metaheuristic development for the SSP is considered as successful even though not suppose a new stage in the SSP solution.

Bibliography

- [1] Ahmadi, E., Goldengorin, B., Süer, G. A., and Mosadegh, H. (2018). A hybrid method of 2-tsp and novel learning-based ga for job sequencing and tool switching problem. *Applied Soft Computing*.
- [2] Al-Fawzan, M. and Al-Sultan, K. (2003). A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. *Computers & industrial engineering*, 44(1):35–47.
- [3] Amaya, J. E., Cotta, C., and Fernández, A. J. (2008). A memetic algorithm for the tool switching problem. In *International Workshop on Hybrid Metaheuristics*, pages 190–202. Springer.
- [4] Amaya, J. E., Cotta, C., and Fernandez-Leiva, A. J. (2012). Solving the tool switching problem with memetic algorithms. *AI EDAM*, 26(2):221–235.
- [5] Bard, J. F. (1988). A heuristic for minimizing the number of tool switches on a flexible machine. *IIE transactions*, 20(4):382–391.
- [6] Bertossi, A. (1981). The edge hamiltonian path problem is np-complete. *Information Processing Letters*, 13:157–159.
- [7] Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, 2(4):353–373.
- [8] Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308.
- [9] Boussaïd, I., Lepagnot, J., and Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, 237:82–117.
- [10] Browne, J., Dubois, D., Rathmill, K., Sethi, S. P., Stecke, K. E., et al. (1984). Classification of flexible manufacturing systems. *The FMS magazine*, 2(2):114–117.
- [11] Catanzaro, D., Gouveia, L., and Labbé, M. (2015). Improved integer linear programming formulations for the job sequencing and tool switching problem. *European journal of operational research*, 244(3):766–777.
- [12] Chaves, A. A., Lorena, L. A. N., Senne, E. L. F., and Resende, M. G. (2016). Hybrid method with cs and brkga applied to the minimization of tool switches problem. *Computers & Operations Research*, 67:174–183.
- [13] Crama, Y., Oerlemans, A. G., and Spieksma, F. C. (1994). Minimizing the number of tool switches on a flexible machine. In *Production Planning in Automated Manufacturing*, pages 165–195. Springer.
- [14] Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812.

- [15] Denizel, M. (2003). Minimization of the number of tool magazine setups on automated machines: A lagrangean decomposition approach. *Operations Research*, 51(2):309–320.
- [16] Dorigo, M. (1992). Optimization, learning and natural algorithms. *PhD Thesis, Politecnico di Milano*.
- [17] Dorigo, M. and Birattari, M. (2011). Ant colony optimization. In *Encyclopedia of machine learning*, pages 36–39. Springer.
- [18] Dorigo, M. and Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66.
- [19] Dorigo, M. and Stützle, T. (2010). Ant colony optimization: overview and recent advances. In *Handbook of metaheuristics*, pages 227–263. Springer.
- [20] Furrer, M. and Mütze, T. (2017). An algorithmic framework for tool switching problems with multiple objectives. *European Journal of Operational Research*, 259(3):1003–1016.
- [21] García-Villoria, A., Corominas, A., and Pastor, R. (2013). Pure and hybrid metaheuristics for the response time variability problem. *Meta-Heuristics Optimization Algorithms in Engineering, Business, Economics, and Finance*, pages 275–311.
- [22] Garey, M. R. and Johnson, D. S. (2002). *Computers and intractability*, volume 29. wh freeman New York.
- [23] Ghiani, G., Grieco, A., and Guerriero, E. (2007). An exact solution to the tlp problem in an nc machine. *Robotics and Computer-Integrated Manufacturing*, 23(6):645–649.
- [24] Ghiani, G., Grieco, A., and Guerriero, E. (2010). Solving the job sequencing and tool switching problem as a nonlinear least cost hamiltonian cycle problem. *Networks*, 55(4):379–385.
- [25] Glover, F. (1977). Heuristics for integer programming using surrogate constraints. *Decision sciences*, 8(1):156–166.
- [26] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549.
- [27] Glover, F. and Laguna, M. (2013). Tabu search. In *Handbook of combinatorial optimization*, pages 3261–3362. Springer.
- [28] Hart, J. P. and Shogan, A. W. (1987). Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6(3):107–114.
- [29] Jäger, G. and Molitor, P. (2008). Algorithms and experimental study for the traveling salesman problem of second order. In *International Conference on Combinatorial Optimization and Applications*, pages 211–224. Springer.
- [30] Johnson, D. S. and McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, 1:215–310.
- [31] Karp, R. M. and Steele, J. M. (1985). Probabilistic analysis of heuristics. *The traveling salesman problem*, pages 181–205.
- [32] Laporte, G., Salazar-Gonzalez, J. J., and Semet, F. (2004). Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions*, 36(1):37–45.

- [33] Lin, S. (1965). Computer solutions of the traveling salesman problem. *The Bell system technical journal*, 44(10):2245–2269.
- [34] Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516.
- [35] Mütze, T. (2014). Scheduling with few changes. *European Journal of Operational Research*, 236(1):37–50.
- [36] Paiva, G. S. and Carvalho, M. A. M. (2017). Improved heuristic algorithms for the job sequencing and tool switching problem. *Computers & Operations Research*, 88:208–219.
- [37] Privault, C. and Finke, G. (2000). k-server problems with bulk requests: an application to tool switching in manufacturing. *Annals of Operations Research*, 96(1-4):255–269.
- [38] Starkweather, T., McDaniel, S., Mathias, K. E., Whitley, L. D., and Whitley, C. (1991). A comparison of genetic sequencing operators. In *ICGA*, pages 69–76.
- [39] Stecke, K. E. (1983). Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems. *Management Science*, 29(3):273–288.
- [40] Tang, C. S. and Denardo, E. V. (1988a). Models arising from a flexible manufacturing machine, part i: minimization of the number of tool switches. *Operations research*, 36(5):767–777.
- [41] Tang, C. S. and Denardo, E. V. (1988b). Models arising from a flexible manufacturing machine, part ii: Minimization of the number of switching instants. *Operations Research*, 36(5):778–784.

Appendix A

Graph-theoric definitions

These definitions were presented by Crama [13] in 1994 and they helped to realize the complexity of the travelling salesman problem.

A *graph* G is a triple of the form (V, E, d) , where:

- V is a finite set; the elements of V are nodes of G .
- E is a set of pairs of nodes, called edges.
- d is a function which assigns a non-negative length to each pair of nodes; we assume that $d(u, v) = +\infty$ when $\{u, v\}$ is not an edge.

A path in a graph is a sequence of nodes, i.e., a permutation of a subset of V . A travelling salesman path is a permutation of V . The length of a path $(u_1, \dots, u_k)$ is by definition:

$$d(u_1, u_2) + d(u_2, u_3) + \dots + d(u_{k-1}, u_k)$$

Notice that the length of such a path is infinite if some pair $\{u_i, u_{i+1}\}$ is not an edge of the graph.

The travelling salesman problem on a graph G can be stated as follows: Finding a minimal length path in graph G . With a graph $G = (V, E, d)$, we can associate another graph $H = (E, I, \delta)$, called the edge-graph of G , and defined as follows:

- Each node of H is an edge of G .
- A pair $\{e, f\}$, with $e, f \in E$, is an edge of H if and only if the edges e and f share a common node in G .
- $\delta(e, f) = 1$ if $\{e, f\}$ is an edge of H , and $\delta(e, f) = +\infty$ otherwise.

Observe that, in an edge-graph, every travelling salesman path has length either $|E| - 1$ or $+\infty$. Consider now the restriction of the travelling salesman problem to edge-graphs, that is: *Input*: A graph G .

Problem P2: Find a travelling path of minimal length in edge-graph G .

Equivalently, $P2$ asks whether there exists a travelling salesman path of finite length in the edge-graph of Bertossi (1981) [6] proved that this problem is NP-hard. In the other hand we can find directed graphs, based by (V, U, d) , with V is the same as for a graph and U and d are defined with the following statement:

- U is a set of ordered pairs of nodes, called arcs; i.e; $U \subset V \times V$.
- d is a non-negative length function defined on $V \times V$, with property that $d(u, v) = +\infty$ when (u, v) is not an arc.

So, in a directed graph, $d(u, v)$ may differ from $d(v, u)$. The definitions of a travelling salesman path and of the travelling salesman problem extend in a straightforward way for directed graphs.

Appendix B

Instances definition

The definition of the instances used for the resolution of the job sequencing and tool switching problem is summarized in this appendix. The author of defined the number of jobs/tools and the capacity of the magazine through different random sequences. Datasets are defined in 4 blocks (A,B,C,D), each one with a different amount of tools and jobs. Every block has 10 instances with different distribution of jobs/tools and distinct magazines capacities (C). T_m^* defines the minimum value of tools needed for a job ($T_m^* = \min_{j' \in J} |T_{j'}|$) and T_M^* defines the maximum value of tools needed in a job ($T_M^* = \max_{j' \in J} |T_{j'}|$). The data can be downloaded **clicking here**¹³. In the following table it can be seen a summary of the datasets:

<i>Dataset</i>	<i>Jobs</i>	<i>Tools</i>	T_M^*	T_m^*	C
datA1	10	10	4	2	4
datA2	10	10	4	2	5
datA3	10	10	4	2	6
datA4	10	10	4	2	7
datB1	15	20	6	2	6
datB2	15	20	6	2	8
datB3	15	20	6	2	10
datB4	15	20	6	2	12
datC1	30	40	15	5	15
datC2	30	40	15	5	17
datC3	30	40	15	5	20
datC4	30	40	15	5	25
datD1	40	60	20	7	20
datD2	40	60	20	7	22
datD3	40	60	20	7	25
datD4	40	60	20	7	30

Table B.1 Definition of the used datasets

¹³<https://bit.ly/2LMcY1s>

Appendix C

ACS python based solution

To achieve the solutions of the JSTP towards the ant colony strategy it was decided to provide an informatics solution developed with python. This program provides the best sequence found for a dataset of the instances studied in this project only providing 4 inputs:

- Dataset sequence between *A1* and *D4*.
- Instance number of the dataset. A number between 0 and 10.
- Number of ants for the ant colony system metaheuristic.
- Number of iterations for the ant colony system.

With this four inputs the program will execute the ant colony system and will provide the solution in the terminal, as well a plotted solution as a figure. This plotted solutions consist in:

- Pheromone map matrix. Result of the pass of all the ants and iterations.
- Tool distribution for the sequence. For the best sequence found the tool distribution calculated with KTNS.

The python based solution can be downloaded **clicking here**¹⁴. The instructions for the running procedure are on the "*Readme.txt*" file.

¹⁴<https://bit.ly/2J8OIKy>