BRUNO HENRIQUE MIRANDA DOS SANTOS

Orientador: Marco Antonio Moreira de Carvalho

UM MÉTODO HEURÍSTICO APLICADO À PRODUÇÃO NA INDÚSTRIA AUTOMOTIVA

Ouro Preto

Setembro de 2016

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

UM MÉTODO HEURÍSTICO APLICADO À PRODUÇÃO NA INDÚSTRIA AUTOMOTIVA

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

BRUNO HENRIQUE MIRANDA DOS SANTOS

Ouro Preto
Setembro de 2016

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas
Graduate Program in Computing Science

A HEURISTIC METHOD APPLIED TO PRODUCTION IN THE AUTOMOTIVE INDUSTRY

Monograph presented to the Undergraduate Program in Computing Science of the Universidade Federal de Ouro Preto in partial fulfillment of the requirements for the degree of in Computing Science.

BRUNO HENRIQUE MIRANDA DOS SANTOS

Ouro Preto
September 2016

UNIVERSIDADE FEDERAL DE OURO PRETO

FOLHA DE APROVAÇÃO

Um Método Heurístico Aplicado à Produção na Indústria Automotiva

BRUNO HENRIQUE MIRANDA DOS SANTOS

Monografia defendida e aprovada pela banca examinadora constituída por:

Dr. Marco Antonio Moreira de Carvalho – Orientador
Universidade Federal de Ouro Preto

Msc. Reinaldo Silva Fortes
Universidade Federal de Ouro Preto

Dr. Tiago Garcia de Senna Carneiro
Universidade Federal de Ouro Preto

Ouro Preto, Setembro de 2016

Resumo

Este trabalho estuda o desafio de 2005 da ROADEF, o problema de sequenciamento de veículos. O Problema de sequenciamento de veículos consiste em determinar a melhor ordem de produção de veículos dado os carros produzidos no dia anterior, os carros a serem produzidos e os itens opcionais a serem instalados nestes carros e as respectivas restrições de capacidade de instalação. Foi realizada uma revisão minuciosa da literatura entre os anos de 1986 a 2016 a fim de gerar o embasamento teórico necessário e também posicionar adequadamente este trabalho na literatura. Este trabalho utiliza os métodos *Depth-First Search*, *Best Insertion* e *Iterated Local Search* para resolver o problema de sequenciamento de carros. *Depth-First Search* provou ser um método de rápida execução e bom em gerar soluções iniciais, *Best Insertion* é capaz de obter os melhores resultados, dentre os três métodos e *Best Insertion* demonstrou ser uma boa estratégia para ser combinada com o *Iterated Local Search*.

Abstract

This work studies the ROADEF 2005 challange problem, the car sequencing. The Car Sequencing Problem is to determine the best order of production given some cars from the previous day, the cars that should be produced, the optional items to be installed in each car and the related capacity restrictions. A thorough literature review is made from the year 1986 to 2016, in order to generate the theoretical basis and to properly position this work on the literature. This work uses Depth-First Search, Best Insertion and Iterated Local Search to solve this problem. Depth-First Search shown to be the fastest method and a good strategy to determine a initial solution, Iterated Local Search is able to obtain the best results, amongst the three methods and Best Insertion is demonstrated to be a good strategy to be combined with the Iterated Local Search.

Dedico este trabalho aos meus pais que me apoiaram ao longo desses cinco anos de curso.

Agradecimentos

Quero agradecer, em primeiro lugar, a Deus, pela força e coragem durante toda esta longa caminhada. Ao professor Marco. Pela paciência na orientação e incentivo que tornaram possível a conclusão desta monografia.

A República DNA onde moro e conheci tantos amigos ao longo dos anos.

Agradeço também ao Eric Bourreau por solucionar os problemas que tive com o site da ROADEF.

Ao colega Kayran dos Santos pelas contribuições com o código e com o LaTeX.

Aos amigos e colegas, pelo incentivo e pelo apoio constantes. A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ACO Ant Colony Optimization. 8

BI Best Insertion. i, ii, 10, 15, 23, 25, 34, 36

CC Color Change. 27

CHIP Constraint Handling in Prolog. 7

CS Car Sequencing Problem. ii, 3, 36

DFS Depth-First Search. i, ii, 10, 14, 15, 21, 23, 26, 30, 34–36

DFS+BI Depth-First Search with Best Insertion. 30, 34–36

GAwHC Genetic Algorithm with Hill Climbing. 7, 8

HPRC High Priority Ratio Constraint. 27, 29

ILS Iterated Local Search. i, ii, 10, 16, 19, 30, 34–36

JIS *Just-in-Sequence*. 1

JIT *Just-in-Time*. 1

LPRC Low Priority Ratio Constraint. 27, 29

ROADEF French Society of Operations Research and Decision Analysis. i, ii, 2, 4–6, 8, 10, 25, 30, 36

Chapter 1

Introduction

Henry Ford developed a new way of producing cars in 1903, known as "Fordism", where each worker would have a very specific job to perform. This innovation greatly reduced the production time and cost taken per each unit to be produced. Due to the reduced cost and production time, it was possible to begin the mass production of items. Later on, it led to mixed production lines where a single line could produce more than one type of product, for instance, an automobile industry could build various types of cars from the same car body. According to Golle (2011), mass customization was created after that, leading to highly differentiate products. For instance, in terms of customization, Mercedes produced 1.1 million of cars in the plant Rasstatt in Germany between the years 2003 and 2005. From all those cars, only two were identical.

According to Boysen et al. (2005), at the beginning of each working day a sequence of cars need to be decided. It often requires that the parts needed are in stock and the time of each work station to be taken in consideration. For the parts, to be delivered in the production line, there are two options, they can be brought from the suppliers, which are *Just-in-Time* (JIT) and *Just-in-Sequence* (JIS). JIT brings the parts in the moment they are required, whereas JIS will pre order them, or change the order when they arrive, so that they will be on the order as they are needed on the production line. Another point is that JIS would require temporary storing, either in the factory or in the suppliers, also the time taken to sort the parts could increase the delivery time.

A work station is a worker or a machine that performs a job by installing an optional item. An optional item is something that the buyer selects when he is deciding which car is the best for his needs. Some examples of optional items are leather seating, sun-roof, and air conditioning. The production line can have the work station moving in two ways, *physical* and *virtual*. In the *physical* mode, the cars are sequenced in order to follow

the working times of the stations. Conversely, in the *virtual* mode the work stations move along the production line and the cars do not need to be changed from the original sequence.

Events such as lack of parts or broken machines could lead to a scenario where the sequencing of cars need to be changed, or *resequenced*. This could lead to great monetary losses and should be avoided at all cost. Lack of parts can be avoided by creating a better sequence of cars to be produced by taking into consideration the inventory. Machine failure can be prevented by doing regular maintenance and by introducing redundancy in the production line, that is, having more than one machine, whenever possible, that can produce or do the job. This is a line of research known as production forecast, that tries to keep every equipment up and running and taking care of parts suply.

The French Society of Operations Research and Decision Analysis (ROADEF) is an international society that, amongst other attributions, organizes an international competition, bi yearly, focused on interacting the academy with companies in order to make the improvements made by the academy appear in companies for better production results. They also formalize the problem description and create sets of instances along the competing year. It also develops solution checkers so that the participants can see the final cost of their solution, whether it is in the right format or not and also validate constraints. This society often offers prizes, in the form of money for its participants, varying up to eleven thousand euros given in ROADEF 2016 conference. ROADEF began developing competitions on 1999 and repeat this process every two years. Each competition is focused on a different type of industry:

- ROADEF'99: Inventory Management Problem;
- ROADEF'01: Frequency Assignment Problem with Polarization constraints;
- ROADEF'03: Management of the Missions of the Earth Observation Satellites;
- ROADEF'05: Car Sequencing problem;
- ROADEF'07: Technicians and Interventions Scheduling for Telecommunications;
- ROADEF'09: Disruption Management for Commercial Aviation;
- ROADEF'10: A Large-Scale Energy Management Problem with varied constraints;
- ROADEF'12: Machine Reassignment;
- ROADEF'14: Rolling Stock Unit Management on Railways Sites; and

- ROADEF'16: Inventory Routing Problem.

This study will focus especifically on the ROADEF'05 with the subject Car Sequencing Problem.

Estellon and Gardi (2013) and Kis (2004) state that the Car Sequencing Problem (CS) is a problem where the company has to produce a determined amount of cars in a working day and the cars of this group have optional items to be added (e.g. radio, sun-roof, air-conditioning, etc.). When dealing with this kind of problem, the goal is to reduce the workload of each working station. The workload of these stations are often defined by the ratio of cars that they can install the optional item within the total amount of cars. For instance, a given station could install a particular option on three out of seven cars. It means that no sequence of cars can have more than three cars with that option to be installed, in a total amount of seven cars. Therefore, the cost of CS is usually associated with how many times this type of constraint is broken.

The other objective of the Car Sequencing Problem is to determine the sequence of cars to be produced in order to reduce the costs of painting, as each change of colors costs a different amount of money. According to Boysen et al. (2012), it can vary from $27 on black up to $122 on amber. Another concern is that each customization can be done to only a few cars for each batch. Note that in this case some breaks might be required in order to remove the paint that have been accumulating in the nozzle tip.

In the case of ROADEF'05 there are three soft constraints, described in Chapter 3. ROADEF'05 established a partnership with car manufacturer company Renault to get the data from real factories in order to generate a more realistic approach.

Estellon and Gardi (2013) and Kis (2004) show that the Car Sequencing Problem is a $\mathcal{NP}$-Hard problem. This problem is being researched over the last twenty years in order to cut costs in the automobile industry. The following nine methods, amongst others, have been used to solve this problem in the past:

- Constraint Programming;
- Greedy;
- Integer Programming;
- Backtracking;
- Branch and Bound;
- Large Neighbourhood Search;

- Local Search;
- Genetic algorithms; and
- Ant Colony Optimization.

Figure 1.1 shows the number of articles published and registered on web of science[1] have been increasing since this problem was proposed firstly by Parrello et al. (1986) as a job-shop schedule problem. The peak on 2008 could be due to the fact that the ROADEF challenge on 2005 have drawn the academic community towards it.

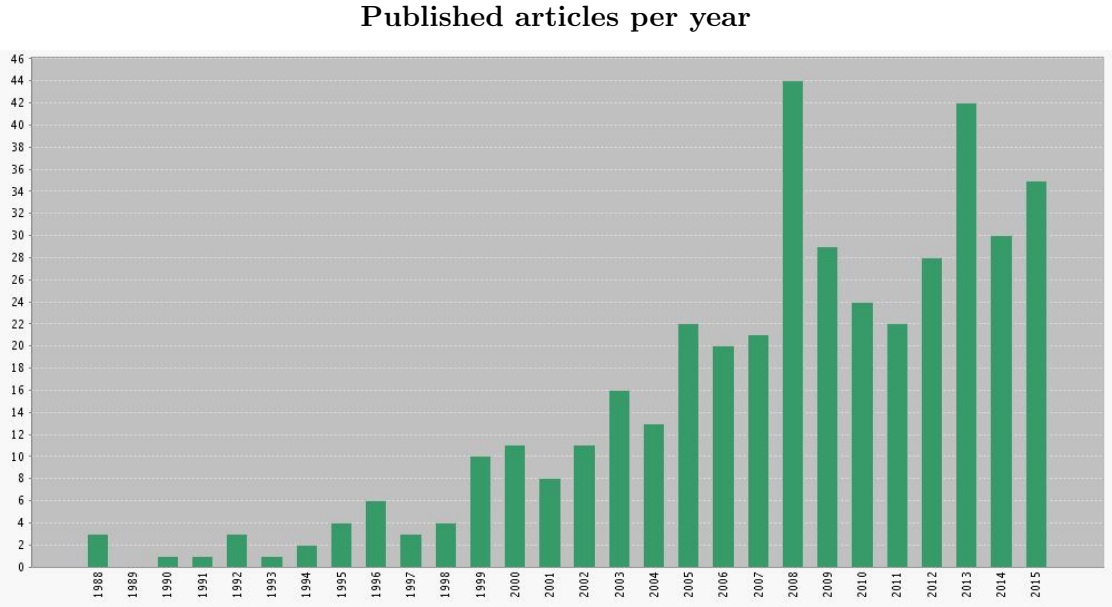

Figure 1.1: Published articles on car sequencing subject after 1987.

1.1 Motivation

There are three main motivations for this work. First, it is a $\mathcal{NP}$-Hard problem. Second, it has been the subject of research by a great Society, ROADEF. Third, it has a big impact on the economy of Minas Gerais state and Brazil.

As mentioned before, it has been shown by Kis (2004) and Estellon and Gardi (2013) that this problem belongs to the class of $\mathcal{NP}$-Hard Problems.

[1] https://webofknowledge.com/ accessed on 26/01/2016

As stated before, on 2005 the French Society of Operations Research and Decision Analysis alongside with car maker Renault, selected this subject as its competition subject on 2005. ROADEF paid up to seven thousand euros as prize for the contestants. It counted with participants from numerous countries such as France, Brazil, Netherlands, Canada, Germany, Switzerland, Poland and Bosnia and Herzegovina.

According to ANFAVEA[2] the automotive industry participation on the overall brazilian gross domestic product was 5% and 23% on the particular industrial sector on 2013. he automotive industry participation on the Minas Gerais gross domestic product was 25.8%. It generated around R$ 178,5 billions in taxes. Brazil is currently seventh largest car producer in the world.

1.2 Goals

This study will devise a heuristic in order to try to outperform the best known results of the literature. This main goal will be achieved by following the steps described below.

- Review the literature on car sequencing;
- Devise a heuristic optimization method that fulfills all problem requirements;
- Implement the previously mentioned method;
- Research for polishing techniques;
- Evaluate the method with real data and publicly available test sets; and
- Compare the method with state of the art methods from the literature.

1.3 Work Organization

The remainder of this work is organized as follows. Chapter 2 presents the literature review from years 1986 to 2016. Chapter 3 presents the basis of the problem with a mathematical description as well as the methods that are going to be used. Chapter 4 presents the main issues and insights regards the implementation. Chpater 5 presents the experiments made as well as the results. Finally, chapter 6 draws the conclusions regarding this version and points the future work.

[2]http://www.anfavea.com.br/50anos/60.pdf Acessed on 09/12/2015

Chapter 2

Literature Review

In this chapter the work done through the years 1986 to 2016 will be described. Parrello et al. (1986) were the first to describe the car sequencing problem on 1986. On 2005 ROADEF used the car sequencing as the problem of its biannual challenge, constituting a milestone on the problem history.

Parrello et al. (1986) described it as a case study of the job-shop scheduling problem. They described step by step how to reduce the job-shop to it showing what is the schedule, the sequence of cars, and the attributes, the constraints for each option. They also showed that it might not have a solution, (i.e., more cars requiring the option than the factory can handle). For these cases the authors created the idea of penalties to change the cost of a solution. Their main goal was to develop an environment to allow other researchers to tackle this problem using automated reasoning, where they state a set of rules and the software would decide the sequence of cars.

For instance, consider that an option has the ratio constraint $1 : 3$ (i.e., for each 3 cars within a sequence, only one can have the optional item installed), and that the input requires 2 cars with that option. Table 2.1 shows, on the left hand side, that regardless of the sequence the penalty would be 1. Another kind of penalty, applied when there are constraint violating cars close to each other, is shown on third and fourth lines of the right hand side of the table. The table represents a part of a car sequence, where Y means that a car has the option and N otherwise.

Table 2.1: Both types of penalties.

car	Y	N	Y
penalty	0	0	1
car	N	Y	Y
penalty	0	0	1

car	Y	N	Y
penalty	0	0	1
car	N	Y	Y
penalty	0	0	3

2.1 90's: Constraint Programming

In the early 90's the effort taken to solve the car sequencing problem focused mainly on Constraint Satisfaction methods. Hentenryck et al. (1991), Hentenryck et al. (1992), A. Davenport and Zhu (1994), Lee et al. (1998) and Davenport and Tsang (1999) developed new algorithms and techniques to improve the current constraint satisfaction solvers, such as the Constraint Handling in Prolog (CHIP). Also, operators to speed up the programming were developed. Due to hardware constraints of that time, most of the experiments conducted considered only small instances up to 400 cars, usually containing only 100 of them, and it would take up to 10 hours to process. The instances used were all randomly generated. Regin and Puget (1997) proposed preprocessing operations in order to remove possible bad values among the data.

After that, Smith (1996) started developing more advanced constraint satisfaction techniques in order to solve the car sequencing problem, but realised that it was not such a good idea. Nevertheless, this work posed interesting ideas:

- Cars with the same optionals would be counted as a class of cars, instead of a single car. If the number of optionals would grow this idea would be infeasible, because the number of possible cars grows exponentially to the number of options (2^O, where O is the number of options);
- The order of cars in the input could be changed so that the backtracking done by the solver would be less frequent and less heavy; and
- A technique for removing possible bad values before assigning them to a variable. A bad value is a value that might lead to a high cost solution.

Instances with up to five options were considered in the reported experiments, but as mentioned above, the results were discouraging.

2.2 00's: Metaheuristc, Heuristic and Exact Methods

Warwick and Tsang (1995) were the first to use a Genetic Algorithm with Hill Climbing (GAwHC). The GAwHC was compared to a Tabu Search and the reported results showed that the GAwHC performed better. However, the GAwHC presented some limitations, such as the maximum size of the input and the time taken to solve an instance.

In the early 00's much more diversity of methods was employed to address the car sequencing problem. Bergen et al. (2001) used multiple types of greedy, local search, backtracking, branch and bound methods, also combining them. Gottlieb et al. (2003) mixed various methods as well, but included the Ant Colony Optimization (ACO) metaheuristic. Perron and Shaw (2004) started using Large Neighbourhood Search to solve it. Neveu et al. (2004) developed IDWALK a local search that would do some changes to the solution and see which solution generated after that was the best one. Drexl et al. (2006) combined branch and bound with tabu search. It is not possible to compare them as they used different test cases of the problem.

On 2005 ROADEF used car sequencing as its biannual contest. Solnon et al. (2007) wrote an article to summarize the contest and the results. ROADEF made an explicit list of what was the problem, how a solution would be evaluated. It has also shown the problem difficulty, it's complexity, as well as some methods the participants used to solve it. Explained how the results were calculated and on the website [1] there is the possibility of sending new results for evaluation and comparison with old ones.

After 2005 much effort was taken in using ACO. The book Dorigo and Stützle (2005) shows various ways of implementing it as well as problems in which it has been used in the past. Some authors such as Solnon (2008) used random greedy initial solution and after that he used ACO to improve the results.

Gravel et al. (2005) compared three methods for solving the car sequencing problem, Constraint Satisfaction programming, Integer programming, and ACO. He used CSPLib test case and proposed a new way of calculating the penalties to remove double counting of cars. However this new calculation has not been found in any other paper.

Boysen et al. (2005) used dynamic programming and simulated annealing.

In recent years, 2009 and after, there was a slight difference in the formulation of the problem. Authors started using more than one line of production or removing the color change costs from the optimization, as it is a separated problem and sometimes it could be solved separately. Boysen et al. (2009), Golle et al. (2010) and Yazgan et al. (2011) all followed this line of research.

[1] http://roadef.proj.info-ufr.univ-montp2.fr/2005/ Assessed on 27/01/2016

Boysen et al. (2012) surveyed various past publications that addressed the car sequencing problem and pointed what he thought to be the future research agenda of this area. This agenda involves tackling multi-objective optimization, resequencing the production line.

As sugested by Boysen et al. (2012) articles found after 2012 showed a much more specific and local definition of the car sequecing problem. Such as Mazur and Niederliński (2015a) and Mazur and Niederliński (2015b) where they take in account the makespam taken in order to produce the veihicles, or Yazgan et al. (2011) were they optimize more than one production line at the same time.

2.3 Problem Difficulty

Regarding the difficulty of the problem, Gent (1998) showed that it is $\mathcal{NP}$-complete by reducing it to the Halmitonian Path. Kis (2004) and Estellon and Gardi (2013) later on showed it is a $\mathcal{NP}$-hard problem.

Chapter 3

Theoretical Foundation

In this chapter is the description of the problem. The details of the input files, and an example to illustrate it. The description of how the methods DFS, BI and ILS work.

The Car Sequencing Problem is usually defined by a tuple (*C, O, K, p, q, r*) where C is the set of cars to be produced, O is the set of different optional items that can be added in the current factory, K is the set of cars from the previous day, p limits the maximum amount of cars that can have a specific optional item installed within a sequence of size q (referred to as *p:q* constraints from now on). The last element r determines whether or not a car has an optional item installed:

$$r(c_i, o_j) = \begin{cases} 1 \text{ if car } i \text{ has option } j \\ 0 \text{ otherwise} \end{cases}$$

The ROADEF proposed a formula for calculating the cost of a solution which is a simplified version of the one proposed by Parrello et al. (1986). For each optional item $o_i \in O$ every subsequence of size q_i is going to be checked to see if there are more than p_i cars in that sub sequence that requires this optional item o_i, according to Equation 3.1.

$$p{:}q \text{ penalties} = \sum_{i=1}^{O} \sum_{j=|K|-q_i+1}^{N} \begin{cases} (\text{count}(j, j+q_i) - p_i) * w_i \text{ if count}(j, j+q_i) - p_i > 0 \\ 0 \text{ otherwise} \end{cases} \tag{3.1}$$

In Equation 3.1:

- O is the number of options;
- N is the number of cars in the sequence;

- K is the number of cars from the previous day;
- w_i is the given weight associated with option o_i;
- q_i is size of the sub sequence to be analyzed and p_i is the maximum numbers of cars that can be in that sequence without having penalties; and
- *count* is a function that counts how many cars in the sub sequence j to $j + q_i$ have option i installed.

When evaluating the number of color changes, only the last car from the previous day is taken into account. Equation 3.2 defines the cost of color changes in a solution, where w_c is the weight associated with changing colors.

$$\text{color change cost} = \sum_{j=|K|+1}^{N} \begin{cases} w_c \text{ if color}(j) \neq \text{color}(j-1) \\ 0 \text{ otherwise} \end{cases} \tag{3.2}$$

Then the final cost of the solution is given by Equation 3.3.

$$Z = (3.1) + (3.2) \tag{3.3}$$

3.1 Input

The input is composed of four files:

optimization_objectives: describes the constraint's priorities for this instance, e.g., if the cost of changing colors is higher than the cost of violating priority *p:q* constraint;

paint_batch_limit: describes how many cars can be painted at maximum using the same color in a feasible solution;

ratios: defines the *p:q* constraints; and

vehicles: describes each vehicle required in the current planning period.

Regarding the penalties for this problem, ROADEF'05 defined three categories of penalties. Constraints of type *p:q* constraints are divided into two groups: *priority* constraints and *non-priority* constraints. The difference between them is that a violation of a priority *p:q* constraint always implies a higher penalty. This difference was added due to how the Renault working plant actually works. Each color change also result in a

penalty. The value associated with a penalty is defined in the input and represented by a permutation of $\{10^0, 10^3, 10^6\}$.

Regarding the vehicles file it is important to mention that there is a set K with $|K| = max(q) - 1$, cars from the previous day. This is the starting point for counting the constrained vehicles for each *p:q* constraint.

3.1.1 Input and Output Examples

Table 3.1 exemplifies how an input could be. In the example, four types of cars need to be produced and may have up to five optional items. The number of cars required of each type is in the bottom of the table. Each option has its *p:q* constraint, that limits the maximum amount of cars, p, that can have the option in any given sequence of size q. The colors were added from the original table in order to be more similar to the ROADEF'05 problem and only two colors were added. A car type is defined by which optionals it has and its color. This could be the cars requested to be built in a day in a factory.

In this example the cost of violating an priority constraint is assumed to be 10^6, a non priority constraint is 10^3 and the cost of changing color is 10^0. The color batch for this example is 4. The last column of Table 3.1 is the priority of each *p:q* constraint. Values 1 and 0 respectively, mean weights 10^6 and 10^3 for the constraints, i.e., define priority and non-priority constraints.

Table 3.1: Example input, adapted from Kis (2004).

Opt.	Car types				Constr.	
	1	2	3	4	$p:q$	Priority
1	✓			✓	3:5	1
2		✓			1:3	0
3	✓	✓			2:5	0
4			✓	✓	2:3	1
5			✓		1:4	1
Color	Black	Blue	Black	Blue		
Req.#	2	2	2	2		

As mentioned before, this instance considers $|K| = 4$ cars from the previous day. The sequence of these cars given in the input is important and must be kept. Table 3.2 is the description of the last four cars from the previous day.

A solution for this problem is a sequence of cars that does not violate the color batch. Solutions may have different costs depending on which set of $p:q$ were violated and the

Table 3.2: Last four cars from the previous day.

Opt.	Car types			
	1	2	3	4
1	✓			✓
2		✓		
3	✓	✓		
4			✓	
5			✓	✓
Color	Black	Blue	Black	Blue

weight of each *p:q* constraints that were violated.

Table 3.3: Optimal detailed result previously cited

Opt.	Car												Constr.
	1	2	3	4	2	4	3	1	4	2	3	1	$p:q$
1	✓	✓			✓			✓	✓			✓	3:5
2			✓				✓			✓			1:3
3		✓	✓		✓		✓			✓		✓	2:5
4	✓			✓		✓		✓	✓		✓		2:3
5				✓		✓					✓		1:3

As can be seen on Table 3.3, the sub-sequence [1, 2, 3, 4, 2] does not follow the rules for the third constraint. In this sub-sequence of size 5, there should be at maximum two cars that require the third optional item, however there are three [2, 2, 3]. The other constraint violation in this example is in the sub-sequence [3, 4, 2, 4, 3] and also violates the third constraint. The cost associated with violating constraints in this case is 2×10^3.

Table 3.4: Color changes

Car	4	2	4	3	1	4	2	3	1
Color	Blue	Blue	Blue	Black	Black	Blue	Blue	Black	Black

Table 3.4 displays how the cost of color changes is calculated. Note that only the cars involved in this calculation are shown, omitting the first three cars [1, 2, 3] from the previous day. The cost associated with color changes in this case is 3×10^0. Therefore the final cost for this solution is 2003.

3.2 Depth-First Search

The Depth-First Search (DFS) algorithm has been widely used when dealing with graphs. Cormen (2009) defines the strategy of the DFS as follows "The strategy followed by depth-first search is, as its name implies, to search "deeper" in the graph whenever possible". It has $O(V + E)$ order of complexity, where V is the number of nodes in the graph and E is the number of edges.

The DFS use a color scheme to define visited vertices:

White: means that the vertex have not yet been reached;

Gray: means that this vertex have been found, and its neighbors are being explore; and

Black: means that the vertex and all its neighbor have been explored.

Algorithm 1 is the initialization of variables for the DFS take place. Method $G.V$ returns the list of all vertices contained on graph G. Method $v.color$ set, or get, the color of vertex v. The second *foreach* in Algorithm 1 is to start the DFS from all vertices of the graph. This is necessary in case the graph is disconnected.

Algorithm 1: DFS Adapted from Cormen (2009).

```
   Data: Graph G
1  foreach v ∈ G.V do
2  |  v.color ← White;
3  |  v.π ← Null;
4  end
5  foreach v ∈ G.V do
6  |  if v.color = White then
7  |  |  DFS-Visit(G, v);
8  |  end
9  end
```

Algorithm 2 marks vertex v as gray, as it begins the procedure. Afterwards for each vertex that exists an edge from v and are still marked as white, the DFS-Visit is called recursively. After all neighborhood vertices of vertex v have been visited, vertex v is then marked as black.

Figure 3.1 exemplifies an DFS execution, the graph has 5 vertices and 5 edges. As described before, a vertex marked in black means that all the neighbors of the vertex have been visited. A thicker edge means that edge have been already transversed. Starting at

Algorithm 2: DFS-Visit Adapted from Cormen (2009).

```
   Data: Graph G, Vertex v
1  v.color ← Gray;
2  foreach u ∈ G.ADJ[v] do
3  |  if u.color = White then
4  |  |  u.π ← v;
5  |  |  DFS-Visit(G, u);
6  |  end
7  end
8  v.color ← Black;
```

vertex 1 the DFS then goes to vertex 2, and so on. Vertex 1 only becomes black after all other vertices have been visited.

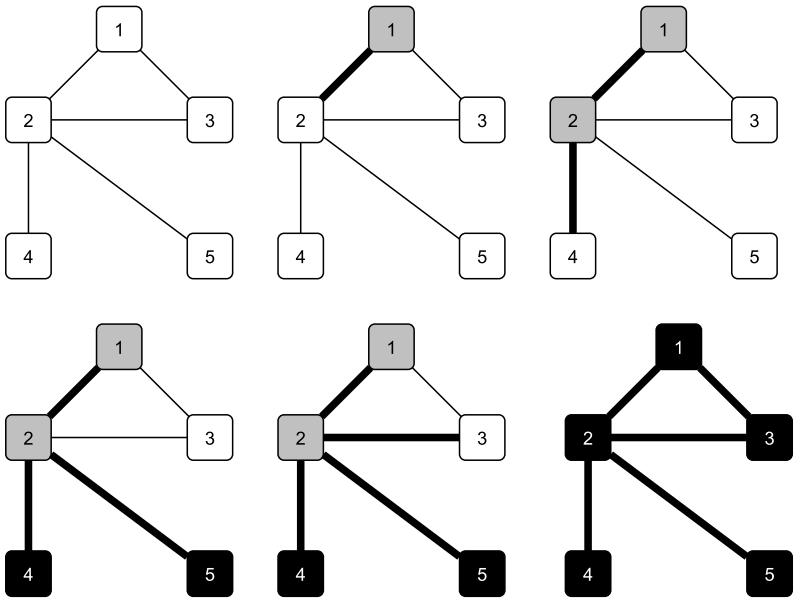

Figure 3.1: DFS Execution.

3.2.1 Best Insertion

A Best Insertion (BI) algorithm is a straightforward greedy technique for permutational problems that usually picks a solution element at random and insert it in another position of the solution. This new position is determined by the best feasible solution value

associated, amongst all possible. Thus if the solution value is improved, the movement is performed.

In the car sequencing context the best insertion removes a car from a postion and has to compute how the ratios would be in the other positions that this car may be inserted. This is a heavy process and should be optimized.

3.3 Iterated Local Search

Lourenço et al. (2003) described the main steps of the Iterated Local Search (ILS). This technique ILS operates over an initial solution, usually a greedy generated one. Afterwards, ILS execute a local search procedure in order to find a local optimum. This first result is considered as the best initially. Then, the ILS applies a perturbation (i.e., a random modification) to the current solution and store it as a candidate solution. This candidate solution undergo different local search procedures in order to achieve local optimum in each one of them. The resulting candidate solution is evaluated and, in case that the acceptance criterion accepts this candidate, the ILS updates the best solution. This process is repeated until a stop criterion is met, returning the best solution found so far.

Algorithm 3 presents the basic operation of the ILS, where S_{best} is the best solution found so far, $S_{candidate}$ is the solution that is currently being changed in order to try to find a better solution and α is a number that indicates how much of the solution is going to be changed during perturbation.

Algorithm 3: ILS

```
   Data: double α
 1 S_best ← InitialSolution();
 2 S_best ← LocalSearch(S_best);
 3 while stop criterion has not been met do
 4 |   S_candidate ← Perturbation(S_best, α);
 5 |   S_candidate ← LocalSearch(S_candidate);
 6 |   if AcceptanceCriterion(S_candidate, S_best) then
 7 |   |   S_best ← S_candidate;
 8 |   end
 9 end
10 return S_best
```

The local search, perturbation and acceptance criterion are often treated as black box algorithms and problem dependent. Also, the strength of the perturbation requires

testing in order to define what is best for the problem.

Algorithm 4 depicts a sample local search. Until the stop criterion is met, it saves the value of the current solution, in variable s', and apply a movement, i.e. swap the position of two elements in the solution, if the result is at least equal to the previous result the movement is accepted, the movement is undone otherwise. At the end it returns a local minima.

Algorithm 4: Local Search

```
   Data: Solution s
1  while stop criterion has not been met do
2  |  s' ← s;
3  |  Movement(s);
4  |  if evaluate(s) > evaluate(s') then
5  |  |  UndoMovement(s);
6  |  end
7  end
8  return s
```

Figure 3.2 shows what the perturbation algorithm should do in a minimization sample problem. Perturbation should differentiate the solution enough so that a different local optimum may be explored by the local search procedures. From a solution s^* and in order to find a different local minimum s^{*}', the perturbation may alter the current solution and jump to another region of the solution space, say, s'. Then, a local search procedure is responsible for refining this new solution until s^{*}' is found.

The Acceptance criterion in a *minimization* problem may be just testing if $S_{candidate}$ is valid and comparing the costs of both, $S_{candidate}$ and S_{best}, although many other criteria may be adopted to this end.

Analogously, the stop criterion adopted may vary. For example it may be maximum number of iterations, maximum running time, etc.

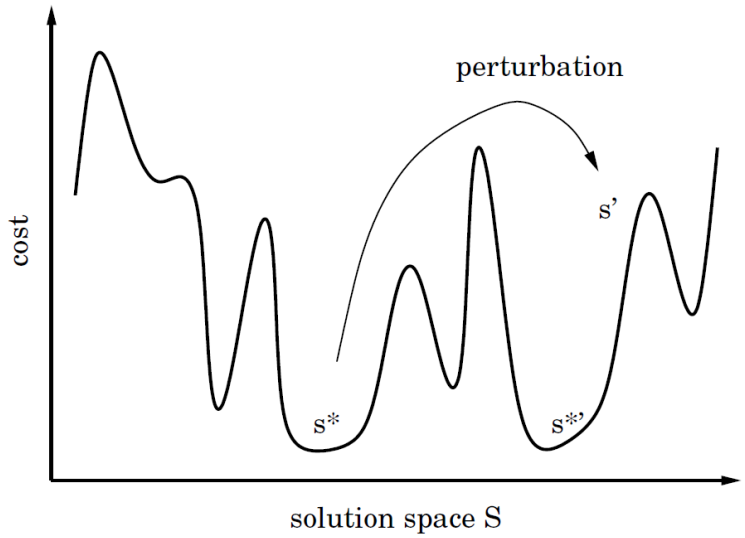

Figure 3.2: Perturbation graph, taken from Lourenço et al. (2003).

Chapter 4

Development

This chapter describes the implementations made, in details, such as the computational modeling and representation, the new constructive heuristic and ILS.

4.1 Computational Representation

There are two main structures for the computational representation of this problem. The first being the data representation and the second being the solution representation. There are also auxiliary classes to represent small entities of the problem.

The data representation is trivial. The solution representation uses three arrays to store the list of cars, which car has been sequenced, and the values associated with each $p : q$ constraint.

The array of cars contains the indexes of the cars of the input data currently in the solution. The array of sequenced cars visited contains the information regarding if a car has been already included in the solution or not. The third array stores the values of each $p : q$ constraint after the last car was added to the solution. Because every car can be sequenced after the others, there is no need to actually store the graph, the array of cars and the array of car visited are enough to understand it as a graph.

The solution representation also has three matrices used to evaluate swap movements faster. Ribeiro et al. (2008) developed a technique to evaluate a swap movement in time $O(m)$, where m is the number of $p : q$ constraints. This technique uses three matrices, M_1, M_2 and M_3. These matrices represents $p : q$ constraints as lines and the cars as columns.

Matrix M_1: stores at row i and column j, the number of cars, starting at position j in the solution sequence ending at position $j + q$, having the optional item i.

Matrix M_2: stores at row i and column j, the number of sub-sequences starting at column 1 on matrix M_1 i^{th} row and ending at column j that have more than p cars in a sub-sequence of size q. That is, the number of each $p:q$ constraints violating sub-sequences.

Matrix M_3: which is similar to M_2, stores at row i, column j, the number of sub-sequences starting at column 1 on matrix M_1 i^{th} row and ending at column j that have at least p cars in a sub-sequence of size q.

For example, consider there are ten cars to be produced and only one *p:q* constraint with $p = 1$ and $q = 4$. Given that five out of ten cars require this option Figure 4.1 is how the cars are sequenced and what the matrices would be. The first position of matrix M_1 counts the cars from position 1 to position 4, totalizing three cars. The other positions in matrix M_1 just slide the window of four cars. Because there is more than $p = 1$ car on the first position of M_1, the first position of M_2 counts one. Again, because there is at least $p = 1$ car on the first position of M_1, the first position of M_3 counts one.

1	2	3	4	5	6	7	8	9	10
X	_	X	X	_	_	_	X	_	X

M_1 = [3 2 2 1 1 1 2 2 1 1]
M_2 = [1 2 3 3 3 3 4 5 5 5]
M_3 = [1 2 3 4 5 6 7 8 9 10]

Figure 4.1: Sample solution and matrices M_1, M_2 and M_3. Source: Ribeiro et al. (2008)

Let us illustrate the use of these matrices in a swap movement, as described in Ribeiro et al. (2008). Suppose that cars at positions a and b were to swap positions in a solution. By using matrices M_2 and M_3 it is possible to fast evaluate the cost of the movement. Equation 4.1 is the decrease in the number of violations associated with ratio constraint q_j.

$$\Delta_j^1 = \begin{cases} M_2[a][j] - M_2[a - q_j][j], & \text{if } a - q_j > 0 \\ M_2[a][j], & \text{otherwise} \end{cases} \tag{4.1}$$

Equation 4.2 is the increase in the number of violations associated with ratio con-

straint q_j.

$$\Delta_j^2 = \begin{cases} M_2[b][j] - M_2[b - q_j][j], & \text{if } b - q_j > 0 \\ M_2[b][j], & \text{otherwise} \end{cases} \tag{4.2}$$

Both equations, 4.1 and 4.2, evaluate only one optional item. The cost of the swap movement is defined by $\Delta_j = \Delta_j^2 - \Delta_j^1$. The total cost movements (i.e., considering the ratio constraints for all optional items) is then computed by Equation 4.3, where m is the number of optional items.

$$\sum_{j=1}^{m} \Delta_j \tag{4.3}$$

4.2 Problem Modeling

In this work, we model the problem using graphs, where each vertex v represents a car and an edge connects two cars v and w if car w could be sequenced after car v, as can be seen in Figure 4.2. This graph corresponds to the example given in Table 3.1. The yellow node represents the cars from the previous day, while blue nodes represent the cars currently being sequenced.

In order to sequence the cars using the graph, to determine a Hamiltonian path would be required, since every car needs to be on the sequence. However, the associated decision problem is $\mathcal{NP}$-Complete adding weights to the edges models the problem more realistically. For example, it can represent the cost of color changing between cars. Figure 4.3 is an illustration of a hamiltonian path detection algorithm. In this case vertex 1 is the initial. The hamiltonian path found is [1, 2, 4, 3].

A quicker way, albeit not optimal, to find a solution would be to perform a DFS, inserting the vertices in the solution as they are explored. This does not grant to have a good, or even feasible solution, since the edges take in consideration only if the next car can be sequenced after the current one. However, it can be a good start for a heuristic method, giving good insights on the solution.

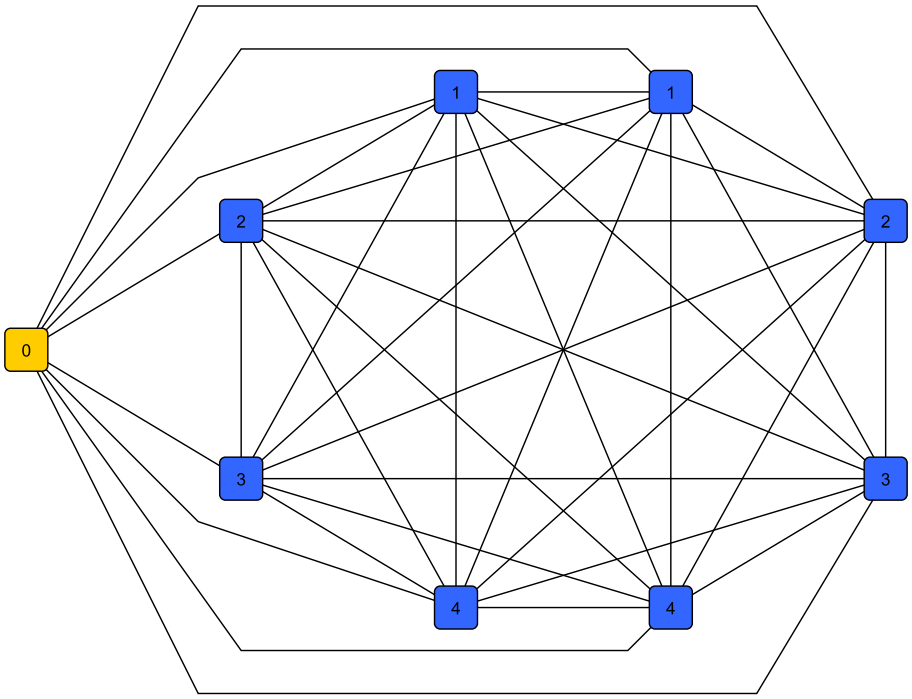

Figure 4.2: Example graph regarding the example in Table3.1.

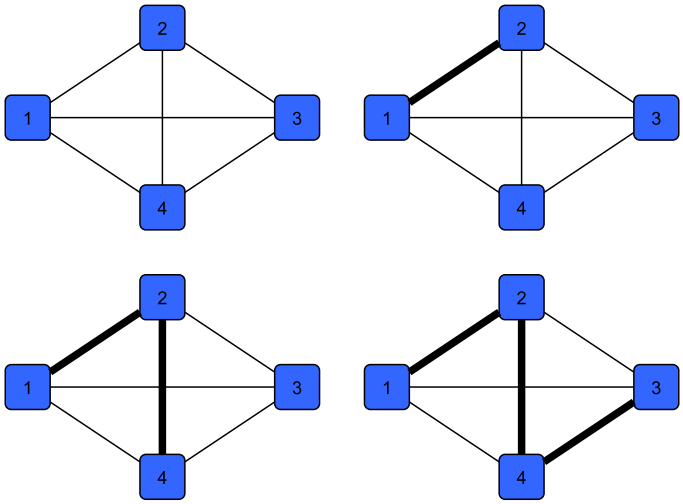

Figure 4.3: Hamiltonian Path.

4.3 A New Greedy Heuristic Method

The heuristic method developed is a combination of Depth-First Search (DFS) and a Best Insertion (BI), both described earlier in Chapter 3.

The DFS firstly produces a list of candidates by iterating through all non-explored cars. After the list is generated the DFS selects a candidate randomly amongst the cars with the lowest cost. The algorithm iterates until no possible candidate is available: either all cars are in the solution or there is no current car that is possible to be inserted.

The calculation of weights and candidate selection of the DFS is described next. Weights of the edges are dynamic in this problem. Figure 4.4 is a snapshot of how this concept works. Node 0 has been included in the solution, it represents the cars from the previous day. The candidate list has four cars: adding car 1 to the solution would increase a cost of 1000 to the solution value, adding car 2 would increase a cost of 1000 and so on. When there is a tie amongst a number of cars the algorithm randomly chooses one.

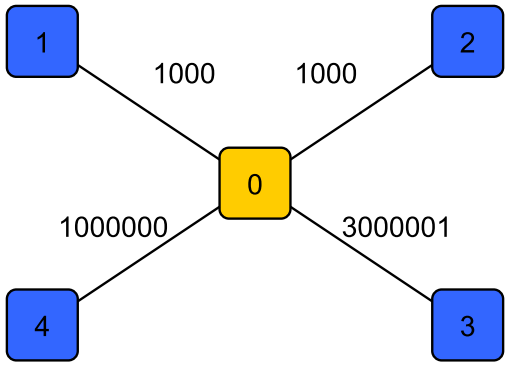

Figure 4.4: Weights before first insertion.

Because it implies in the smallest solution value increase, car 2 is randomly chosen and added to the solution. Therefore, the weights associated to sequencing the remaining cars change as can be seen on Figure 4.5. The cost to add car 4 to the solution now is 0, the cost to add car 3 is 1000001, and so on. The DFS would continue this process until there no more cars to be inserted in the solution.

DFS is a straight up algorithm, as it does not backtrack. Sometimes it is possible that there are still remaining nodes, but the solution would not be feasible if any of those nodes were inserted. Another problem that might occur is the associated penalty

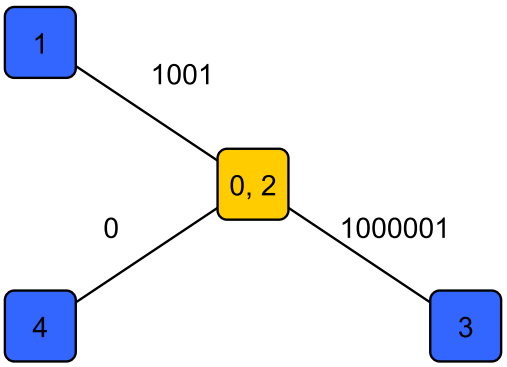

Figure 4.5: Weights after first insertion.

to insert a node in the solution is too big and, in order to avoid this increase in the cost, this item should be inserted in another position in the solution. In these cases is interesting to use a Best Insertion algorithm, as described in Chapter 3.

The best insertion technique is executed every time that adding a car to the solution increases the cost of the solution. Figure 4.6 exemplifies how the BI algorithm works. The first four cars (in yellow) are from the previous day. The next three (in blue) are cars from the current day and do not have a penalty associated. When adding the eighth car (red) to the solution a penalty is applied. To avoid the increase on the solution value, the algorithm removes this car from the solution and simulates its insertion between previous cars in the solution, starting from the last car of the previous day (i.e., last yellow car). At each of these insertions, the solution is evaluated and the best insertion is determined as the one implying in the best solution value (i.e., no penalty or lowest penalty). In the given illustration, the best insertion is made between cars 202 and 203.

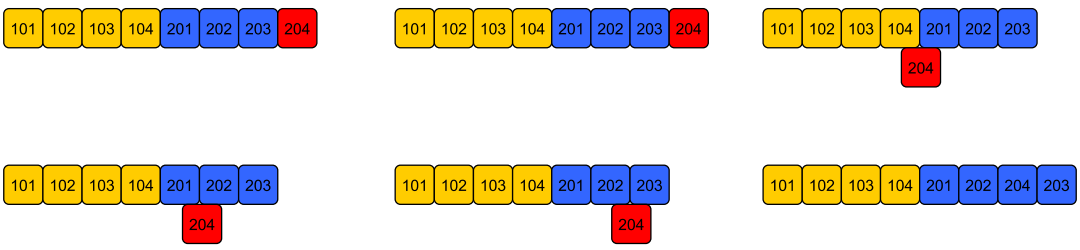

Figure 4.6: Best insertion execution example.

This technique uses the previously described matrix M_1 to jump a few steps ahead and speed up the process. For instance, if a car was to be inserted at position i and this would imply a penalty related to a $p : q$ constraint with $q = 10$, then, that penalty would span until position $i + 10$. There is no need to check positions $i + 1$ to $i + 10$, so the best insertion instead of moving the car one position ahead moves it straight to position $i + 11$.

4.4 Iterated Local Search

Ribeiro et al. (2008) proposed the following local searches for the car sequencing solution, considering only the cars of the current planning day. All cars (or group of cars) are chosen randomly.

Car Exchange: swaps the positions of two cars;

Car Insertion: removes a car from a position and insert it in another position;

Reinsertion: removes a random number of cars, varying from two to eight, and insert them at other positions using the Best Insertion method;

Group exchange: swaps a group of contiguously sequenced cars with another; and

Group inversion: invert the cars sequence inside a group of contiguous cars with the same color.

However after some testing we noticed that only *Car Exchange, Reinsertion* and *Group Inversion* produced good results. So it was decided that to use only these methods as local search.

Group Exchange might not be good due to the fact that it requires groups of cars with the same color, and in the ROADEF instances the number of colors is relatively small if compared to the number of cars. Therefore, it may cause that the exchanged groups were either too different, or too similar. If the groups are too different, this local search procedure could lead to a potential increase in the number of violations in the solution. Otherwise, the solution cost would not be changed.

Car Insertion might not generate good solutions due to the fact that replacing just one car may be not enough to actually improve the solution. Also, the computational time of this movement is quite high when compared to others. *Reinsertion* also has a high computational time, however, since it repositions more cars at a time it can generate better solutions.

Algorithm 5 is shows the implemented Iterated Local Search. The DFS was used as the initial solution generation, due to its processing time. Based on the preliminary tests performed, the best value for the perturbation was found to be $\alpha = 0.2$, i.e., 20% of the solution elements will undergo random swap movements.

Algorithm 5: ILS Implementation

```
   Data: Graph G, double α
 1 S_best ← DFS(G);
 2 while stop criterion has not been achieved do
 3 |   S_candidate ← Perturbation(S_best, α);
 4 |   S_candidate ← GroupInversion(G, S_candidate);
 5 |   S_candidate ← Reinsertion(G, S_candidate);
 6 |   S_candidate ← CarExchange(G, S_candidate);
 7 |   if AcceptanceCriterion(S_candidate, S_best) then
 8 |   |   S_best ← S_candidate;
 9 |   end
10 end
11 return S_best
```

Chapter 5

Experiments

In this chapter is the description of the computational environment, where the experiments were executed. The instances and their main attributes. The experiments made and the results obtained by the methods described on Chapter 4.

5.1 Computational Environment

The computational environment used on the experiments consists of a processor Intel® Core™ i7-4790 CPU @ 3.60 GHz, 16 GB of memory RAM and Ubuntu 14.04.4 LTS operational system.

5.2 Instances

Table 5.1 shows the details for the instance set X taken from the ROADEF'05 challenge site[1]. The table is ordered, increasingly, by the number of cars in each instance. #Cars is the number of cars in the given instance; $\#p : q$ is the number of $p : q$ constraints in the given instance and *Color Batch* is the size of the color batch for the given instance. The next three columns show the penalty values for each instance. On top of these columns is the type of the constraints: High Priority Ratio Constraint (HPRC), Low Priority Ratio Constraint (LPRC) and Color Change (CC). In each row for these columns is the associated weight, chosen from the set $\{10^6, 10^3, 10^0\}$, as mentioned on Chapter 3.

[1] http://challenge.roadef.org/2005/en/sujet.php Accessed on 20/07/2016

Table 5.1: ROADEF instances characteristics.

Name	#Cars	#Color	#p:q	Color Batch	HPRC	LPRC	CC
BATILLY_CH2	78	5	6	100	10^6	10^3	10^0
DACIA_CH1	91	6	1	1000	10^3	10^0	10^6
CORDOBA_CH2	234	7	4	10	10^6	10^0	10^3
CURITIBA_VU	252	7	8	30	10^6	10^0	10^3
CORDOBA_CH1	278	9	5	10	10^6	10^0	10^3
FLINS_CH2	372	12	6	15	10^6	10^0	10^3
DACIA_CH2	377	7	2	1000	10^3	10^0	10^6
BATILLY_CH1	460	15	26	100	10^6	10^3	10^0
SANDOUVILLE_CH2	508	13	20	12	10^6	10^0	10^3
REVOZ	718	13	12	200	10^3	10^0	10^6
SANDOUVILLE_CH1	792	13	22	12	10^6	10^0	10^3
BURSA	822	14	7	60	10^6	10^0	10^3
FLINS_CH1	974	13	11	15	10^6	10^3	10^0
CURITIBA_VP	985	10	8	400	10^6	10^0	10^3
MAUBEUGE	1071	20	20	60	10^6	10^3	10^0
VALLADOLID	1279	13	12	40	10^6	10^0	10^3
DOUAI_CH3	1283	17	12	20	10^6	10^0	10^3
PALENCIA	1338	15	18	10	10^6	10^0	10^3
DOUAI_CH1	1543	15	12	20	10^6	10^0	10^3

Table 5.2 shows the values of the *p:q* constraints in each instance. They are divided in High Priority Ratio Constraint and Low Priority Ratio Constraint. In order to summarize the data, multiple constraints were grouped together. For instance, the LPRC of instance BATILLY_CH2 is 1:(2, 3, 5, 14) meaning that there is at least one constraint of each 1:2, 1:3, 1:5, 1:14. For some instances there maybe more than a constraint with the same value, however, repeated values are not displayed. The symbol '-' denotes the absence of a constraint type.

Table 5.2: Roadef instances characteristics, continued.

Name	HPRC	LPRC
BATILLY_CH2	1:6	1:(2, 3, 5, 14)
DACIA_CH1	1:2	-
CORDOBA_CH2	1:(6, 9, 16), 2:3	-
CURITIBA_VU	1:(2, 4, 17, 22), 3:5	1:(5, 13)
CORDOBA_CH1	1:(2, 15)	1:5
FLINS_CH2	1:(2, 5, 20, 100), 8:10	-
DACIA_CH2	1:2	-
BATILLY_CH1	1:(2, 6, 7, 8, 10) 1:(12, 20, 25)	1:(3, 4, 8, 10, 12, 13, 16, 18, 19), 1:(39, 91, 136), 3:(4, 5), 4:5
SANDOUVILLE_CH2	1:(2, 3, 4, 6, 8, 10)	1:(3, 4, 5, 6, 7, 9, 20, 25, 50)
REVOZ	1:3, 5:6	1:(3, 4, 5, 6, 10), 2:3, 10:15
SANDOUVILLE_CH1	1:(2, 3, 4), 2:3	1:(1, 3, 4, 5, 6, 7, 8, 9, 10), 1:(18, 137, 274), 2:3
BURSA	1:(6, 7), 3:4	1:(2, 3), 39:43
FLINS_CH1	1:(2, 4, 5, 8, 10, 30, 100), 2:3, 4:9	1:(5, 9)
CURITIBA_VP	1:(2, 5, 29)	1:(3, 6,7, 10, 65)
MAUBEUGE	1:(3, 4, 10), 2:3	1:(2, 3, 6, 7, 9, 18, 20, 28, 76), 2:3, 5:8, 17:30
VALLADOLID	1:(6, 7, 10, 20)	1:(2, 3, 5, 8, 10)
DOUAI_CH3	1:(3, 5)	1:(8, 14, 16, 19, 25, 38, 47, 247), 12:60, 41:60
PALENCIA	1:(2, 4, 13), 3:5, 6:7	1:(3, 6, 7, 10, 15, 20)
DOUAI_CH1	1:20	1:(2, 4, 6, 7, 8, 13, 19, 56, 297)

5.3 Results

The next three tables present data in the following standard: Column *BKS* presents the best known solutions, taken from the ROADEF[2] website; column S^* gives the best solution found by the proposed methods, S is the average solution value over all independent runs for each method, σ is the standard deviation among the solution values for all runs for each instance and T is the running time expressed in seconds. Column *gap* presents the percentage distance between the BKS and S^*, calculated as $100 \times (S^* - BKS)/BKS$.

Tables 5.3 and 5.4 show the results of the Depth-First Search (DFS), Depth-First Search with Best Insertion (DFS+BI), for each instance. DFS and DFS+BI were run each 10 independent times. Table 5.5 show the results of the Iterated Local Search (ILS) were run 5 independent times.

[2]http://roadef.proj.info-ufr.univ-montp2.fr/2005/ Accessed on 20/06/2016

Table 5.3: DFS results.

Instance	BKS	gap(%)	S^*	S	σ	T
BATILLY_CH2	3.0	200033.33	6004	5306404.3	3552688.7	0.01
DACIA_CH1	5010000.0	0.22	5021000	5023300.0	1791.6	0.01
CORDOBA_CH2	153034000.0	56.20	239033000	267735000.0	20885487.9	0.06
CURITIBA_VU	55590.0	510859.12	284042175	329841165.3	26988456.6	0.07
CORDOBA_CH1	30000.0	0.00	30000	430000.0	800000.0	0.06
FLINS_CH2	37000.0	113508.11	42035000	60637300.0	12899288.8	0.08
DACIA_CH2	6056000.0	0.45	6083000	6094200.0	8121.6	0.13
BATILLY_CH1	36341495.4	831.23	338424075	400527277.4	30888441.1	0.25
SANDOUVILLE_CH2	31077118.0	913.88	315085345	364580895.2	26965211.7	0.29
REVOZ	12002003.0	0.33	12042122	12047848.2	5994.0	0.56
SANDOUVILLE_CH1	196971.0	126398.10	249164570	299750254.9	28211898.5	0.50
BURSA	110069.0	22.19	134488	31728868.6	32461050.8	0.82
FLINS_CH1	61183030.0	1077.03	720141436	773043309.1	45249267.7	1.00
CURITIBA_VP	8087035.0	3795.49	315029403	340428472.7	19540292.7	0.93
MAUBEUGE	160365.0	489239.55	784729372	866365470.4	60179793.2	2.02
VALLADOLID	189075.0	542058.59	1025086362	1048893996.6	15321570.7	1.91
DOUAI_CH3	231030.0	176534.32	408078274	429278735.8	13990969.8	1.25
PALENCIA	328006.0	261279.42	857340191	949623222.9	63726323.9	2.54
DOUAI_CH1	69237.0	2.41	70904	71052.9	133.7	1.82

Table 5.4: DFS+BI results.

Instance	BKS	gap(%)	S^*	S	σ	T
BATILLY_CH2	3.0	133400.00	4005	5202404.4	2891062.5	0.00
DACIA_CH1	5010000.0	0.22	5021000	5023100.0	1445.7	0.01
CORDOBA_CH2	153034000.0	60.14	245074000	311771100.0	28382555.8	1.82
CURITIBA_VU	55590.0	518211.09	288129137	309630436.8	13529778.4	26.84
CORDOBA_CH1	30000.0	450.00	165000	979400.5	1328088.4	53.93
FLINS_CH2	37000.0	130140.54	48189000	55490200.0	6528131.2	249.74
DACIA_CH2	6056000.0	0.46	6084000	6091700.0	2968.2	0.10
BATILLY_CH1	36341495.4	1001.66	400358062	431977363.9	23728979.7	105.82
SANDOUVILLE_CH2	31077118.0	991.73	339279584	391048178.0	22606966.8	551.51
REVOZ	12002003.0	0.33	12042122	12547747.7	670093.6	0.46
SANDOUVILLE_CH1	196971.0	192019.78	378420254	407408561.5	16225939.1	4420.89
BURSA	110069.0	85784.88	94532631	144135738.4	28539632.9	904.29
FLINS_CH1	61183030.0	2043.79	1311633430	1350933969.7	26348570.9	3640.96
CURITIBA_VP	8087035.0	4475.91	370055478	387166314.0	13108359.2	1084.60
MAUBEUGE	160365.0	604873.25	970165358	1061728486.6	54671658.9	3218.19
VALLADOLID	189075.0	481655.12	910878484	959172872.3	29373981.6	8727.04
DOUAI_CH3	231030.0	157785.02	364761757	376963609.6	8634509.2	13679.40
PALENCIA	328006.0	439207.00	1440953323	1520660372.9	56856421.7	10415.80
DOUAI_CH1	69237.0	1371.27	1018664	1031958.3	8820.6	44145.30

Table 5.5: ILS results.

Instance	BKS	gap(%)	S^*	S	σ	T
BATILLY_CH2	3.0	0.00	3	3.4	0.5	1800.00
DACIA_CH1	5010000.0	0.18	5019000	5023200.0	3059.4	1800.00
CORDOBA_CH2	153034000.0	0.01	153045000	153045600.0	489.9	1807.60
CURITIBA_VU	55590.0	37872.73	21109039	23312651.0	1940354.2	1812.60
CORDOBA_CH1	30000.0	0.00	30000	33800.0	4915.3	1819.00
FLINS_CH2	37000.0	191.89	108000	111200.0	2481.9	1990.00
DACIA_CH2	6056000.0	0.51	6087000	6093600.0	4176.1	1802.00
BATILLY_CH1	36341495.4	290.96	142080131	154500927.8	8269657.9	2030.59
SANDOUVILLE_CH2	31077118.0	187.28	89279583	95673764.0	3437417.2	2097.79
REVOZ	12002003.0	0.29	12037165	12046746.8	6619.1	2062.87
SANDOUVILLE_CH1	196971.0	79304.25	156403340	166027837.2	5758979.3	3914.80
BURSA	110069.0	23.05	135440	306836.6	86097.5	1977.59
FLINS_CH1	61183030.0	379.96	293652290	302445135.2	7138626.2	2604.59
CURITIBA_VP	8087035.0	648.22	60508933	65085308.0	3430398.2	2399.19
MAUBEUGE	160365.0	194055.62	311357658	330927890.6	10745414.1	2858.39
VALLADOLID	189075.0	76469.77	144774302	157970707.0	8631611.3	2808.59
DOUAI_CH3	231030.0	57824.35	133822636	139217630.6	5270829.9	7388.98
PALENCIA	328006.0	141652.35	464956229	477967860.8	7771664.2	3167.39
DOUAI_CH1	69237.0	2.57	71014	71089.4	65.5	12400.80

Based on data presented by Table 5.3, it is possible to see that the DFS is a fast method, usually does not require more than three seconds to run and can be used as a initial solution. However, the value of the solution is quite high and has a large standard deviation. On instance BATILLY_CH2 the BKS is 3 and the best result found by the DFS is 6004. Despite the actual value being 6004, this actually means that only seven additional penalties are taken (six of value 1000 and one of value 1): the objective function is not linear. It is important to notice that, despite being a fast method, the DFS should be run more than once in order to have a more interesting initial solution, due to the fact that it contains randomness.

According to Table 5.4, the best insertion sometimes is not as efficient as expected. It gets closer to the BKS result, however, the running time required to compute the results of a instance may be much greater. For example, on instance SANDOUVILLE_CH1 the DFS run takes only 0.48 seconds on average, whereas DFS+BI takes 4420.89 seconds on average. This is due to the fact that local optima of this problem frequently are worse than the global optimum, i.e., DFS+BI may insert a car in a position that at the local context seems the right decision but in the global context this car should be in a completely different position on the solution.

The low quality of local-based decisions is further explained by the nature of *p:q* constraints. These constraints are cumulative in the solution sequence, however, they are also renewable, i.e., at each q cars in the solution the value p is reset to zero and starts accumulating again. Thus, in a partial solution a *p:q* constraint value gives us no insight on the structure of the complete solution structure. The more different *p:q* constraints, the less accurate the local movements are.

The running time of the ILS is defined to run for 1800 seconds. In some cases (DOUAI_CH1, DOUAI_CH3, PALENCIA, etc.) the running time is higher, this is due to the fact that the BI sometimes takes longer than its given time to process. Regarding the objective function value, the cost still varies from run to run, but less than DFS and DFS+BI. ILS is also able to obtain results closer to the BKS. In some cases, ILS is able to achieve the BKS value. Other cases, such as CURITIBA_VU, MAUBEUGE and PALENCIA may have large gaps due to the fact that it is highly constrained and the ILS was not able to make enough changes in the solution in the given time.

Table 5.6 shows the best result of each method for each instance and the ranking over all three methods. *Result* is the best result found for each method for each instance and *Ranking* is the ordinal ranking that each method got in each instance, when compared to the other methods. The ordinal ranking assigns a value to each method according to its performance: to the best performing method the value 1 is assigned; to the second best

performing method, the value 2 is assigned and so on. In case of ties, the same number is assigned to the methods. This gives us a way to directly compare the methods' performance.

Table 5.6: Ranking of the three methods, DFS, DFS+BI and ILS.

Instance	Result			Ranking		
	DFS	DFS+BI	ILS	DFS	DFS+BI	ILS
BATILLY_CH2	6004	4005	3	3	2	1
DACIA_CH1	5021000	5021000	5019000	3	3	1
CORDOBA_CH2	239033000	245074000	153045000	2	3	1
CURITIBA_VU	284042175	288129137	21109039	2	3	1
CORDOBA_CH1	30000	165000	30000	1	3	1
FLINS_CH2	42035000	48189000	108000	2	3	1
DACIA_CH2	6083000	6084000	6087000	1	2	3
BATILLY_CH1	338424075	400358062	142080131	2	3	1
SANDOUVILLE_CH2	315085345	339279584	89279583	2	3	1
REVOZ	12042122	12042122	12037165	3	3	1
SANDOUVILLE_CH1	249164570	378420254	156403340	2	3	1
BURSA	134488	94532631	135440	1	3	2
FLINS_CH1	720141436	1311633430	293652290	2	3	1
CURITIBA_VP	315029403	370055478	60508933	2	3	1
MAUBEUGE	784729372	970165358	311357658	2	3	1
VALLADOLID	1025086362	910878484	144774302	3	2	1
DOUAI_CH3	408078274	364761757	133822636	3	2	1
PALENCIA	857340191	1440953323	464956229	2	3	1
DOUAI_CH1	70904	1018664	71014	1	3	2
			Average Ranking:	2.05	2.79	1.21

As can be seen on Table 5.6, ILS clearly performs better than DFS and DFS+BI: on average, ILS got a raking of 1.21. On the instances that ILS could not obtain the best solution, it is close to the best. On instance DACIA_CH2, ILS took three penalties of 10^3 more than the DFS. On instance BURSA, ILS took 48 penalties of 10^0 less than the DFS but took one penalty of 10^3. On instance DOUAI_CH1, ILS took 890 penalties of 10^0 less than the DFS but took one penalty of 10^3.

Chapter 6

Conclusions

The Car Sequencing Problem showed to be an interesting problem, in terms of difficulty and what can be developed. A review of the literature was made from the year 1986, when the problem was first proposed, to 2016. Three main methods were developed to solve this problem: Depth-First Search (DFS), Depth-First Search with Best Insertion (DFS+BI) and Iterated Local Search (ILS). Regarding the ILS five main local searches were implemented, but only three showed to be able to improve the solution value.

A surprise that arises is the DFS+BI performed worse than DFS, both in solution value and running time. Regarding the solution value, that may be caused by the fact that when the BI algorithm is running it inserts a car that seems to be better at a particular position, however, the position that the DFS selected for it is the best at the end. Regarding the running time, the BI is comparably quite slow, because it needs to loop through all so far selected cars from the current day and it is a quite heavy process.

When comparing DFS, DFS+BI and ILS with other authors using the ROADEF website these three methods are not able to achieve a good ranking. With a overall rank of 33 with a mark of -0.3666 whereas the best author has a mark of 1.0122 it is easy to see that there are still improvements to be done with the ILS.

Future work includes considering more test cases, the development of new local search procedures, the use of techniques such as parallelism. Further computational experiments must also be carried out.

Bibliography

C. Wang A. Davenport, E. Tsang and K. Zhu. Genet: a connectionist architecture for solving constraint satisfaction problems by iterative improvement. *In Proceedings of AAAI'94*, pages 325–330, 1994.

M.E. Bergen, P. van Beek, and T. Carchrae. Constraint-based vehicle assembly line sequencing. *Proceedings of the 14th Canadian Conference on Artificial Intelligence*, 2056:88–99, 2001.

N. Boysen, M. Fliedne, and A. Scholl. Sequencing mixed-model assembly lines: Survey, classification and model critique. *Elsevier*, 2009.

N. Boysen, A. Scholl, and N. Wopperer. Resequencing of mixed-model assembly lines: Survey and research agenda. *Elsevier*, pages 594–604, 2012.

Nils Boysen, Malte Fliedner, and Armin Scholl. Level scheduling for batched jit supply. *Flexible services and manufacturing journal*, 21(1-2):31–50, 2005.

Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

Andrew Davenport and Edward Tsang. Solving constraint satisfaction sequencing problems by iterative repair. In *Proceedings of the first international conference on the practical applications of constraint technologies and logic programming (PACLP)*, pages 345–357, 1999.

M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2005.

A. Drexl, A. Kimms, and L. Matthießen. Algorithms for the car sequencing and the level scheduling problem. *Springer Science*, pages 153–176, 2006.

B. Estellon and F. Gardi. Car sequencing is np-hard: a short proof. *Journal of the Operational Research Society*, 64(10):1503–1504, 2013.

I.P. Gent. Two results on car-sequencing problems. *APES*, 1998.

Uli Golle. *On the Car Sequencing Problem: Analysis and Solution Methods*. Phd thesis, Johannes Gutenberg-Universität Mainz, Fachbereichs Rechts- und Wirtschaftswissenschaften, 2011.

Uli Golle, Nils Boysen, and Franz Rothlauf. Analysis and design of sequencing rules for car sequencing. *Elsevier*, 2010.

J. Gottlieb, M. Puchta, and C. Solnon. A study of greedy, local search and ant colony optimization approaches for car sequencing problems. *Applications of evolutionary computing*, 2611:246–257, 2003.

M. Gravel, C. GagnÃ©, and W. L. Price. Review and comparison of three methods for the solution of the car sequencing problem. *Operational Research Society*, 2005.

P. V. Hentenryck, H. Simonis, and M. Dincbas. Performance of a comprehensive and efficient constraint library using local search. *Artificial Intelligence 58(1-3)*, pages 113–159, 1992.

P. V. Hententyck, V. Saraswat, and Y. Deville. The cardinality operator: a new logical connective and its application to constraint logic programming. *ICLP-91*, 1991.

Tam'as Kis. On the complexityof the car sequencing problem. *IEEE Transaction on Consumer Electronics*, 53(3):1186–1194, August 2004.

J.H.M. Lee, H.F. Leung, and H.W. Won. Performance of a comprehensive and efficient constraint library using local search. *11th Australian JCAI, LNAI. Springer Verlag*, 1998.

Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

M. Mazur and A. Niederliński. A two-stage approach for an optimum solution of the car assembly scheduling problem part 1. problem statement, solution outline and tutorial example. *Archives of Control Sciences*, 25:355–365, 2015a.

M. Mazur and A. Niederliński. A two-stage approach for an optimum solution of the car assembly scheduling problem. part 2. clp solution and real-world example. *Archives of Control Sciences*, 25:367–375, 2015b.

B. Neveu, G. Trombettoni, and F. Glover. Id walk: A candidate list strategy with a simple diversification device. *Springer Science*, 2004.

Bruce D Parrello, Waldo C. Kabat, and L. Wos. Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated Reasoning*, 2(10):1–42, 1986.

L. Perron and P. Shaw. Combining forces to solve the car sequencing problem. *Proceedings of CP-AI-OR'2004*, 3011:225–239, 2004.

J.-C. Regin and J.-F. Puget. A filtering algorithm for global sequencing constraints. *LNCS*, 1330(3):32–46, 1997.

Celso C Ribeiro, Daniel Aloise, Thiago F Noronha, Caroline Rocha, and Sebastián Urrutia. A hybrid heuristic for a multi-objective real-life car sequencing problem with painting and assembly line constraints. *European Journal of Operational Research*, 191 (3):981–992, 2008.

B. Smith. Succeed-first or fail-first: A case study in variable and value ordering heuristics. *third Conference on the Practical Applications of Constraint Technology PACT'97*, pages 321–330, 1996.

Christine Solnon. Combining two pheromone structures for solving the car sequencing problem with ant colony optimization, 2008.

Christine Solnon, Van Dat Cung, Alain Nguyen, and Christian Artigues. The car sequencing problem: overview of state-of-the-art methods and industrial case-study of the roadef'2005 challenge problem. *EUROPEAN JOURNAL OF OPERATIONAL RESEARCH (EJOR)*, 2007.

T. Warwick and E. Tsang. Tackling car sequencing problems using a genetic algorithm. *Journal of Evolutionary Computation - MIT Press*, 3:267–298, 1995.

H. R. Yazgan, I. Beypinar, S. Boran, and C. Ocak. A new algorithm and multi-response taguchi method to solve line balancing problem in an automotive industry. *Springer*, 2011.