Politecnico di Milano

School of Industrial and Information Engineering

Master of Science in Management Engineering

RESOLUTION OF A MULTI-PROJECT SCHEDULING PROBLEM WITH RESOURCE AVAILABILITY AND ELIGIBILITY CONSTRAINTS THROUGH OPTIMIZATION TECHNIQUES

Christianne Bezerra Sepulveda

Personal Code **10494909**

Id Number **831485**

Supervisor
Professor Francesca Valeria Fumero

Academic Year: 2015/2016

ABSTRACT

This work was motivated by the request of an important Brazilian bank, which recognized the importance of improving the allocation of the programmers in its software factory projects. The allocation problem aims to optimize the allocation of project activities to the factory programmers. The activities require a specific skill to be performed, have a predefined duration and are interconnected with other project activities by a strong relationship of precedence. The programmers have a set of skills and a schedule that describes their availability. A mixed integer linear programming model to represent this situation was developed by the author. It is also capable of allocating activities to the programmers based on their technical prowess and time availability. The Mixed Linear Integer Programming model was solved using the CPLEX software, and a solution for a reduced version of the real problem was found. Due to the problem's highly complex nature, it can be classified as an NP-hard type, the exact solution to the full real problem could not be found since the software's computational limits were exceeded. Thus, to provide a solution for the complete real problem, a constructive heuristic method based on list scheduling was developed. The model presented a better solution than the one implemented by the bank in terms of the weighted sum of the project completion times. With effective use of these models, the company is able to make well-grounded decisions with respect to its capacity and resource planning.

Keywords: Resource Allocation. Scheduling. Operations Research. Mixed Integer Linear Programming. Constructive Heuristic.

SOMMARIO

Questo lavoro è stato motivato dalla necessità di un'importante banca brasiliana di ottimizzare l'assegnazione ai programmatori delle attività necessarie alla realizzazione di diversi progetti della sua fabbrica di software. Ogni attività richiede una competenza specifica da utilizzare, una durata predefinita ed è interconnessa con le altre attività del progetto di cui fa parte da un forte rapporto di precedenza. I programmatori hanno un insieme di competenze e un orario di lavoro che determinano la loro disponibilità. È stato sviluppato un modello di programmazione lineare intera mista per rappresentare questa situazione. Tale modello è anche in grado di assegnare le attività ai programmatori in base alla loro abilità e al tempo tecnico richiesto a seconda della loro disponibilità. Il modello di programmazione lineare intera mista è stato risolto utilizzando il software CPLEX, ed è stata trovata una soluzione ad una versione ridotta del vero problema. A causa della sua natura altamente complessa, la soluzione esatta del problema completo può essere classificata come di tipo NP-hard e non è stato possibile trovarla a causa dei limiti di calcolo dei software. A tal punto, per fornire una soluzione al problema reale, è stato sviluppato un metodo euristico-costruttivo basato sulla lista di pianificazione. Valutando la somma ponderata delle date di conclusione dei progetti, si è constatato che il modello sviluppato costituisce una soluzione migliore rispetto a quella già esistente nella banca. L'uso adeguato di tali modelli permetterà alla banca di prendere importanti decisioni riguardanti la capacità produttiva e la distribuzione adeguata di funzionari.

Parole Chiave: Allocazione delle Risorse. Pianificazione. Ricerche Operativa. Programmazione Lineare Intera Mista. Euristica Costruttiva.

LIST OF FIGURES

Figure 1 Participation of the Financial Sector in the Country's Total IT Spending 9

Figure 2 Quantity of Bank Operations per Service Channel 11

Figure 3 Investments and Expenditures in Technology by Brazilian Banks 12

Figure 4 Bar Chart to PERT Chart conversion 20

Figure 5 PERT Duration Calculation for a Single Activity 21

Figure 6 CHDSP for machine environments 28

Figure 7 CHDSP for processing restrictions and constraints 28

Figure 8 CHDSP for objective functions 28

Figure 9 Scheduling Algorithms Classification 54

Figure 10 Flow Chart for the heuristic algorithm 59

LIST OF TABLES

Table 1 Gantt Chart for Project X 15

Table 2 Table of activities 15

Table 3 Map of Skills 16

Table 4 Programmer Schedules 16

Table 5 Quantity of restrictions for each set of restrictions 38

Table 6 Example Problem Project A Gantt Chart 39

Table 7 Example Problem Project B Gantt Chart 39

Table 8 Description of Activities of the Example Problem 39

Table 9 Map of Skills of the Example Problem 40

Table 10 Programmers’ Schedule of the Example Problem 40

Table 11 Description of the Programmers’ Lack of Availability of the Example Problem ... 40

Table 12 Relation between the Events and the Programmers of the Example Problem 41

Table 13 Projects’ Priority 41

Table 14 Xijk values for the preliminary test 45

Table 15 Ci values for the preliminary test 45

Table 16 CPl values for the preliminary test 46

Table 17 Result of Preliminary Test 46

Table 18 Gantt Chart of 2 Real Projects 47

Table 19 Map of Skills for 8 programmers and 7 skills 48

Table 20 Schedules of 6 programmers for the first working days of December 49

Table 21 Importance of the projects 50

Table 22 Comparison of the resolution times for the MILP when varying the inputs 52

Table 23 Main Priority Rules 55

Table 24 Heuristic Method Step by Step (1) 61

Table 25 Heuristic Method Step by Step (2) 61

Table 26 Heuristic Method Step by Step (3) 62

Table 27 Heuristic Method Step by Step (4) 62

Table 28 Heuristic Method Step by Step (5) 62

Table 29 Heuristic Method Step by Step (6) 63

Table 30 Heuristic Method Step by Step (7) 63

Table 31 Comparison between MILP and Heuristic Method 64

Table 32 Comparison between the results of the heuristic and the real problem 67

Table 33 Comparison between the heuristic and the real problem objective functions 67

Table 34 Other heuristic success factors 68

Table 35 Programmers Occupation Rate after Heuristic Result 69

Table 36 Programmers Occupation Rate – Second Allocation 70

Table 37 Success Factors – Reduced Number of Resources 71

SUMMARY

1. INTRODUCTION 9
1.1 Context 10
1.2 Problem Definition 14
1.3 Objective 17
1.4 Relevance of the work 17
1.5 Outline of the work 18
2. LITERATURE REVIEW 19
2.1 Project Management Scheduling Techniques (PMST) 19
2.1.1 Gantt Chart, Milestone Chart and Line of Balance 19
2.1.2 Networking Scheduling Techniques 21
2.1.3 Heuristics for Scheduling 22
2.1.4 Application of PMST in the Software Factory 23
2.2 Scheduling Problems 24
2.2.1 Machine Environment 24
2.2.2 Processing Characteristics and Constraints 25
2.2.3 Optimal Criteria 26
2.2.4 Complexity Hierarchy in Scheduling Problems 27
2.2.5 Single Stage Multi Machine Problems 28
2.3 Related works 31
3. MILP MODEL 34
3.1 The Mathematical Formulation 34
3.2 Example 38
3.2.1 Data 38
3.2.2 Model 41
3.2.3 Results 44
4. RESOLUTION METHODS AND NUMERICAL EEXPERIMENTS 47

4.1 Data Collection 47

4.2 MILP Model Resolution 51

4.3 Constructive Heuristic 52

4.3.1 Literature Review 52

4.3.2 Heuristic Algorithm 55

4.3.3 Example 60

5. DISCUSSION OF RESULTS 66

5.1 Sensibility Analysis of the Resource Constraints 68

6. CONCLUSIONS 72

REFERENCES 74

APPENDIX A 77

APPENDIX B 81

1. INTRODUCTION

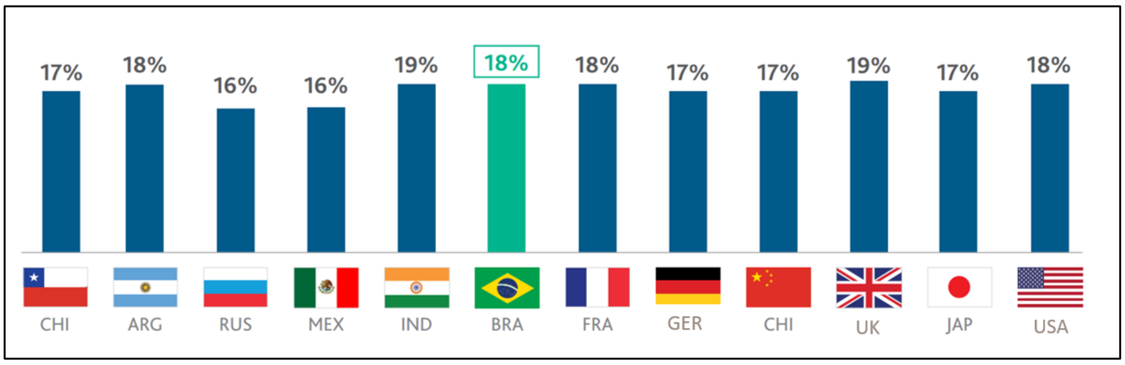
The Brazilian banking industry has undergone significant changes since the late 1960s, and even more so since the early 1990s. These changes involve innovations in products, management and governance practices, market strategies, pricing rules for services and operations (spreads), asset and liability management, mergers and acquisitions and the entry of foreign banks.

In this process, two transformations stand out. First, is the transformation of banking in providing services and facilities to customers, such as online payment accounts and debit transactions. The other is the specialization and the strengthening of financial intermediation activity between lenders and borrowers in the short-term credit segment.

In the 2014 Bank Technology Survey held by the Brazilian Federation of Banks (FEBRABAN, 2014), it was found that Brazilian banks invested R$ 21.5 billion in information technology. According to the president of FEBRAN, Murilo Portugal, "Developments in the banking market would not have been possible without IT, making banks accessible for millions of people in remote locations and at any time of day."

Figure 1 shows the participation of the financial sector as a percentage of the country's total IT spending. The financial services industry is the largest investor in information technology among industries globally. In Brazil, the total IT spending in 2014 was USD 59 billion, of which the financial institutions accounted for approximately 18%. This percentage of investment is similar to that seen in major emerging and developed countries.

Figure 1 Participation of the Financial Sector in the Country's Total IT Spending
(% Of Total IT Spending - 2014)



Source: FEBRABAN Bank Technology Survey 2014

The president of FEBRABAN also pointed out that IT improves agility, security and has contributed greatly to increase banking services, with the number of people with access to these services jumping from 50 million in 2012 to 108 million in 2014. This increase allowed for a significant growth in the popular access to bank services and credit.

In an environment where technological advancement is crucial in enabling internal and external processes to function in the most secure and agile way, the optimization of the IT development process is mandatory in order to improve the cost effectiveness.

This present work describes the development and the results of a project aimed for optimizing the allocation of programmers in the software factory of an international bank, which has operations in the Americas, Europe and Asia, providing services in a wide range of business segments. The bank has preferred to remain unidentified for competitive reasons.

With the intent to provide an overview for this work, the following sections present the context, the problem definition, the objectives, and the relevance for the company.

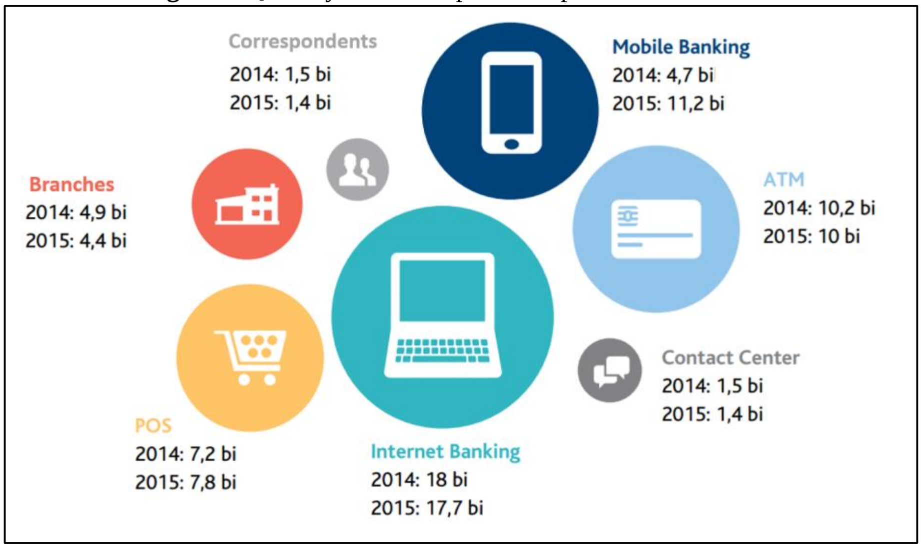
1.1 Context

Brazilian banks are at the country's technological forefront with the launch of solutions, innovative products and services since the mid-1960s, when the first computers arrived in the financial sector. Since then, financial institutions have gone through an intense automation process and digital transformation, which included the deployment of real-time processing systems, ATM network expansion and the development of online and mobile banking, currently used on a large scale.

The launch of the first internet banking platforms occurred in the second half of the 1990s. Prior to that, the main relationship channels between banks and customers were the agencies, ATMs and via telephone.

Mobile banking, which has attracted significant investments from banks so as to ensure comfort and safety to the customer, has evolved into an important gateway to the financial inclusion of millions of Brazilians. According to Gustavo Fosse, FEBRABAN's sector director of banking technology and automation, the use of mobile banking has grown exponentially since 2010, when only 780,000 accounts (less than 1% of total current accounts) were enabled to use this feature. In the first half of 2015, the share of mobile banking among the service channels, which had reached 11% in 2014, rose to 21% of total operations, being placed second among consumer preference. It is an important milestone that reflects not only the advancement of a new channel, but also the beginning of a new relationship between consumers and their banks. **Figure 2** shows the amount of bank operations per service channel in 2014 and 2015.

Figure 2 Quantity of Bank Operations per Service Channel

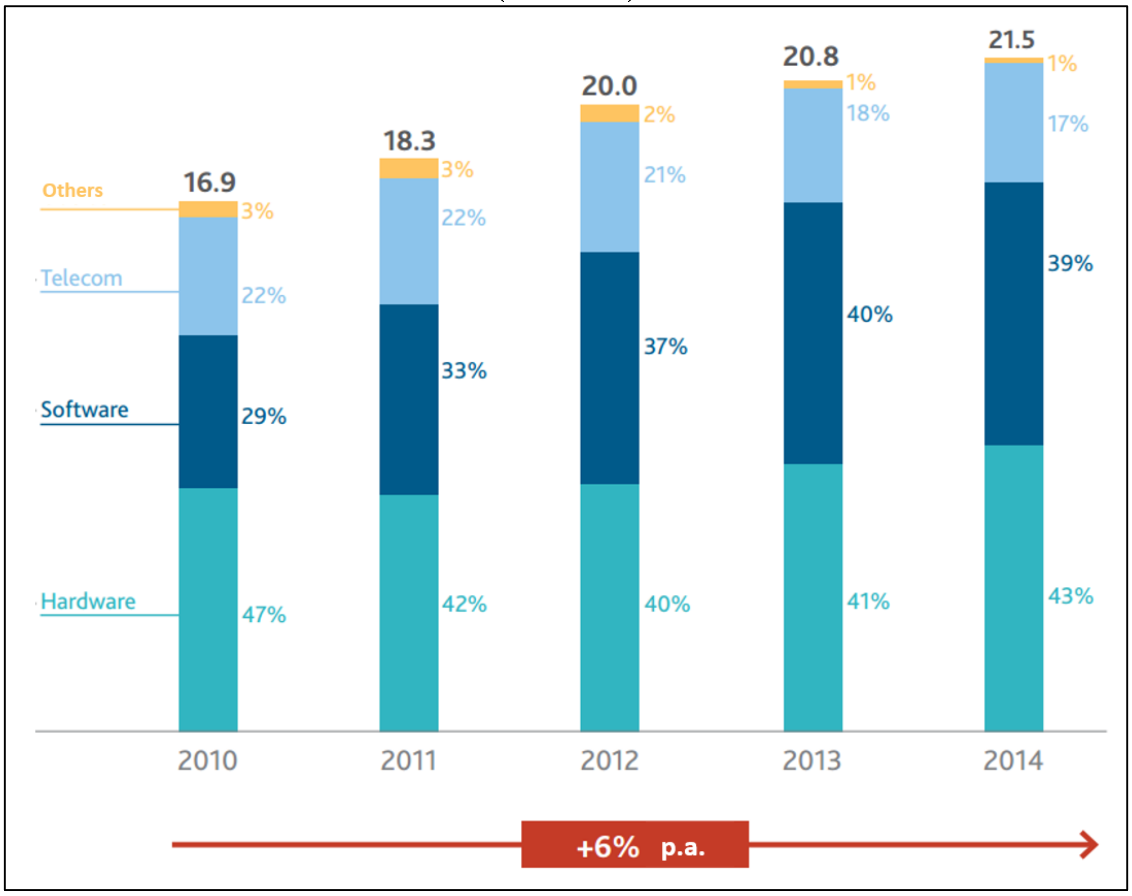


Source: FEBRABAN Bank Technology Survey 2015

A survey provided by the IBGE (Brazilian Institute of Geography and Statistics) presented a demonstration of the potential of the digital medium as a financial inclusion mechanism: According to the survey, internet access via mobile phone in Brazilian households exceeded, for the first time, access via personal computer. From 2013 to 2014 the percentage of households that accessed the internet via microcomputer fell from 88.4% to 76.7%, while the proportion of households that accessed the mobile internet has grown from 53.6% to 80.4%.

The customer base has grown and become much more demanding; the internal and external demands of the institutions have also become far more complex and diverse, requiring increasingly well prepared frameworks. Achieving competitive advantage requires investment in technological updating and constant development of innovative processes, resulting in new or improved products and services for the banking customer. Banks are now preparing for new technological challenges, strengthening its pioneering search for innovations that result in improved convenience and customer experience. Given such a scenario, it is easy to understand the importance of software factories inside the Brazilian financial institutions.

Figure 3 Investments and Expenditures in Technology by Brazilian Banks
(R$ Billions)



Source: FEBRABAN Bank Technology Survey 2014

As **Figure 3** shows, investments in software are growing at higher rates than in hardware or telecom, indicating that banks are increasingly more concerned with customer service, consumer experience and efficiency. The total expenses are stabilizing, highlighting the banks' concern for efficiency measures. The growth in hardware investments between 2013 and 2014 occurred in some banks due to an increase in capacity and the modernization of data storage – if we expunge these exceptional investments, software would account for 42% of total expenses and investments. Application development with internal resources was the type of expenditure that grew the most (47% p.a.) over the last five years, showing that banks are increasingly more concerned with creating skills and differentiated offerings inside their business.

The expression "software factory" was used for the first time in the 60s, but only in the 90s did it become more widespread. The term expresses the idea of applying industrial concepts to the software development environment, pursuing an increase in productivity and quality as well as cost and time reduction.

According to FERNANDES & TEIXEIRA (2004), the basic attributes of a software factory are:

- A defined and standardized process;
- Controlled interaction with the client;
- Standard service requests;
- Estimates of cost and time;
- Strict control of resources involved in each factory demand;
- Control and storage of items of software libraries;
- Control of the status and implementation of all demands;
- Generated products according to the standards set by the organization;
- Trained and qualified staff in organizational and production processes;
- Quality control of the product;
- Customer service processes;
- Defined metrics and control of service level agreements defined with the customer ■

MEDEIROS et al. (2004) provide the description of a hypothetical functional team that would be required for a software factory as well as their activities:

- Business Manager: market research and sales services;
- Project Manager: risk management and managing activities under development, scaling and allocating resources, customer interaction and business manager;
- Systems Analyst: survey requirements analysis, architecture definition and documentation system to be developed;
- Quality Assurance Analyst: review of the generated artefacts, control of possible changes, definition and validation of quality and accuracy of the process used;
- Software Engineer: system implementation according to the specifications and documentation, following the defined developing process;
- Test Engineer: development, validation and implementation of software testing in order to ensure the quality and accuracy of the software produced;
- Team Leader: coordination and tasks assignment within a specific group, reporting periodically the progress of the activities to the project. ■

The software factory of the problem in question is one focused on construction and maintenance of software used in different areas of the bank. It supports development systems divisions in the creation and maintenance of programs that make up the systems that support the bank's operations.

The internal software factory model implemented by the bank was a pioneer in financial institutions in Brazil. According to the bank’s systems superintendent in 2010, "This model allowed for a better understanding of the operation and allowed a more seamless transition from the traditional model, where the roles of analysis and programming were confusing, to a collaborative model where competence centres can be observed. As a result, we have teams that are better trained, more productive, and more assertive when it comes to deadlines and the costs negotiated”.

1.2 Problem Definition

The software factory considered in this work faces a resource allocation problem in the projects planned for the next project cycles (approximately 30 projects/cycle).

Each project has its own timeline and set of skills needed for each of its phases. These aspects are summarized in a Gantt Chart for each project. In the Gantt Chart, each row represents one activity of the project and each column is the day number with respect to the first day of the project.

In **Table 1**, a reduced and simplified version of Gantt Chart for project X can be observed, the first input of the problem being considered. The Gantt Chart shows that activity A1 needs to be performed in the 1st, 2nd and 3rd day of the project. Activity A3, on the other hand, needs to be performed during the course of the entire project, while skill A7 needs to be performed in the 1st and 2nd, and from day 9th to day 12th of project X and so on and so forth.

The link between the activities that belong to the same project are also fixed. The Gantt chart must be followed once the project has started. Accordingly, the completion time of each activity is related to the completion time of the other activities in the same project. For example, the completion time of A1 needs to be 9 days before than the completion time of activity A3.

Once one of the activities of the project is allocated, the allocation for all the others is already known and has to be followed.

Table 1 Gantt Chart for Project X

		Days of the Project											
		1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th
Activities	A1	▨	▨	▨									
	A2		▨	▨	▨	▨	▨	▨	▨	▨	▨		
	A3	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	A4						▨	▨	▨	▨			
	A5			▨	▨	▨							
	A6	▨	▨										
	A7	▨	▨							▨	▨	▨	▨
	A8											▨	▨

Table 2 presents the skill needed to perform each of the activities.

Table 2 Table of activities

Activity	Skill needed
A1	S1
A2	S2
A3	S3
A4	S4
A5	S5
A6	S6
A7	S7
A8	S8

The software factory has approximately 2000 programmers, each with different skills. This is summarized in a Map of Skills. Each row of the Map of Skills is the representation of a programmer and each column is the representation of a skill. The shaded cells indicate that the programmer has the skill, whereas the blank cells indicate that they do not possess the skill in question.

In **Table 3**, it is possible to observe a reduced and simplified version of the Map of Skills. The names of the programmers and their abilities are kept unidentified due to competitive reasons. This is the second input of our problem. The Map of Skills says that programmer 1 possesses skills S1 and S9, programmer 2 has skills S4, S5 and S6, while programmer 8 is capable of performing skills S4 and S9 and so forth.

Table 3 Map of Skills

		Skills											
		S1	S2	S3	S4	S5	S6	S7	S8	S9	S9	S10	S11
Programmers	P1		▨							▨			
	P2				▨	▨	▨						
	P3	▨										▨	▨
	P4		▨	▨									
	P5									▨	▨		
	P6						▨	▨	▨				
	P7	▨	▨			▨						▨	
	P8				▨						▨		

Each programmer has their own schedule and working hours. This is summarized in a Schedule. In **Table 4**, a simplified and reduced version of the Programmer Schedules is shown, the third input of our problem. Each row represents a programmer and each column represents a fixed day in the calendar. **Table 4** gives the schedule for the twelve first working days of the month. Shaded cells indicate that the programmer is not available, blank cells show that they are free. From the Schedule, it can be observed that programmer 1 has a lack of availability from day 4 to day 7, meaning that they have a day off for some reason, or that they have another project already scheduled, training etc. Consequently, the programmer cannot work on a new activity that day.

Table 4 Programmer Schedules

		Days											
		1	2	3	4	5	6	7	8	9	10	11	12
Programmer	P1				▨	▨	▨	▨					
	P2												
	P3												
	P4		▨	▨								▨	▨
	P5	▨	▨	▨	▨	▨							
	P6				▨	▨	▨	▨					
	P7												
	P8		▨	▨			▨	▨	▨				

Each project has a different priority level and technical features, which must be adhered to. The project can only be initiated when all the resources are allocated with the required times and skills. If there are no employees available with the requisite skills and time needed for the project, then it must be delayed. The programmers can perform a set of activities, but only the ones that they possess the skill to be able to perform. The expected solution is the allocation of the activity to a skilled programmer on an available day.

The restriction that links the activity completion times in the same project is the technique used to uphold the Gantt chart of the project.

The completion time of each project is equal to the completion time of the last activity needed to complete the project. In other words, the duration of a project starts when the first activity needed to complete the project and ends when the final activity needed to complete the project ends. Thus, the project’s time of completion is the time at which the final activity is finished. Once the completion time of the project is calculated, the objective of the model is to minimize the completion time of the projects weighted by their importance.

1.3 Objective

This works strives to assist the software factory in making its decisional processes more robust, systematic and formal. The best allocation is the one that minimizes the total weighted completion time of the projects.

In order to achieve the full objective, the resolution process should not only generate the proper allocation of the project activities to programmers, but also achieve better results than the current ones.

The data set provided by the company is going to be applied in the method developed in this work, and its outcomes are going to be compared with the actual results. This comparison aims to check if the model fulfils its purpose.

1.4 Relevance of the work

In a market as competitive as the financial sector, operational efficiency can result in cost reduction and an increase in the availability of programmers, providing a competitive edge and creating value for the bank. With an optimized allocation of its resources, the software factory can improve their service level, reduce the project delays and reduce costs.

Once the software factory is operating within the best allocation scenario, it can receive other requests for the improvement or development of software in departments that are not currently prioritized in that bank sector. The financial benefits of the increase of the potential internal clients are difficult to measure, but the reduction of mechanical and repetitive activities can improve employee motivation, availability to perform intellectual activities, the operational time of routine activities as well as reducing operational errors.

1.5 Outline of the work

The rest of this report is organized as follows.

Chapter 2 presents a literature review about project management scheduling techniques and scheduling problems. A significant range of definitions useful for the full understanding of the project is presented. A brief review of similar works of scheduling is also discussed.

Chapter 3 presents a mixed integer linear programming (MILP) model developed to represent the problem of the software factory. First, the complete model is introduced and presented, including indexes, variables, constants, parameters, objective function and constraints. An explanation is provided for each of the model constraints. After the description of the model, a small example is presented, with a subsequent application of the MILP model and finally the preliminary results are discussed.

Chapter 4 begins with the presentation of the data collected. The following sub section shows the results of the MILP model for the complete and partial sets of the real problem data. Subsequently, constructive heuristic theory is briefly presented and a possible heuristic for the problem resolution is proposed. The suggested heuristic is applied to the same reduced problem of the MILP example. The chapter ends with a comparison between the results of the MILP model and the heuristic model in order to validate it.

Chapter 5 compares the solution found by the proposed heuristic and the current solution used by the company. Some improvement suggestions for the model are presented and other success factors for a good allocation solution in addition to the weighted completion time.

The final chapter contains the conclusion, highlighting the importance of this work for the company and for academic literature. Some suggestions for future studies are provided.

2. LITERATURE REVIEW

In this chapter a literature review is presented about project management scheduling techniques and scheduling problems. A significant range of definitions useful for the full comprehension of the work is presented. A brief review of similar works of scheduling is also discussed.

2.1 Project Management Scheduling Techniques (PMST)

According to KERZNER (2009), management is continually seeking new and better control techniques to cope with the complexities, masses of data, and tight deadlines that are characteristic of highly competitive industries. Managers also seek better methods for presenting technical and cost data to customers. Scheduling techniques help achieve these goals. The most common techniques are:

- Gantt or bar charts
- Milestone charts
- Line of balance
- Networks
  - Program Evaluation and Review Technique (PERT)
  - Arrow Diagram Method (ADM) – also called the Critical Path Method (CPM)
  - Precedence Diagram Method (PDM)
  - Graphical Evaluation and Review Technique (GERT)
- Heuristic for Scheduling

2.1.1 Gantt Chart, Milestone Chart and Line of Balance

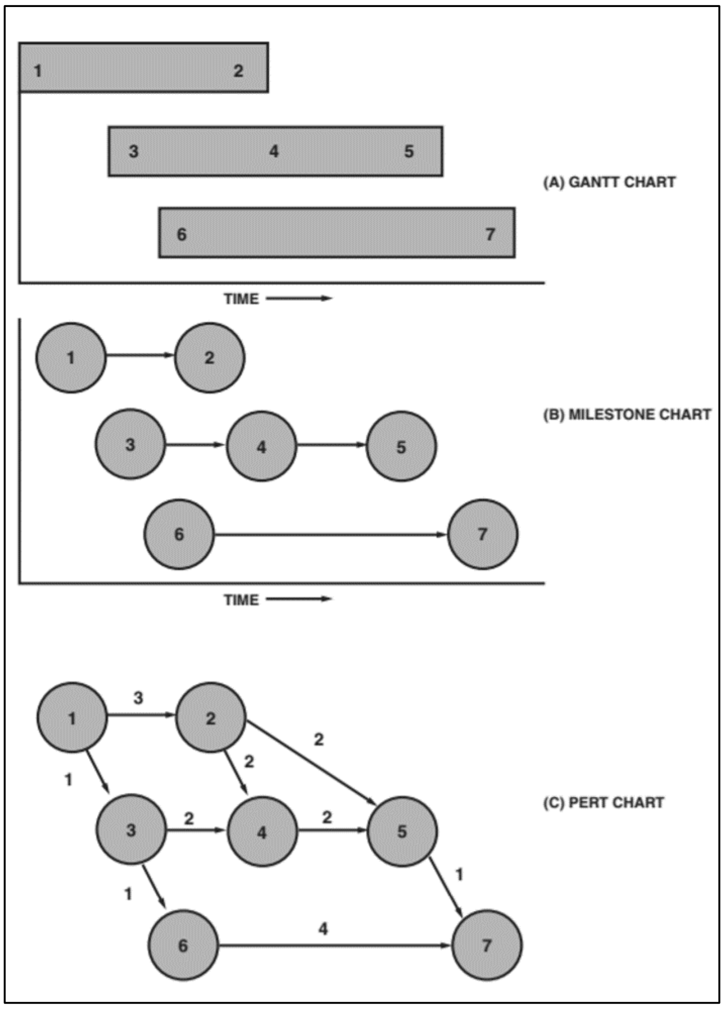
The **Gantt Chart** is probably the most popular form of scheduling. It is the usual horizontal bar chart, with the horizontal axis representing the time and the vertical axis the various machines. A color and/or pattern code may be used to indicate a characteristic or an attribute of the corresponding job, (PINEDO, 2008). **Figure 4** (A) presents a small Gantt Chart with three activities and their precedence order.

The **Milestone Chart** tool shows milestones against a time scale in order to highlight key project events and to draw stakeholder attention to them. A milestone is defined as a point in time or event whose importance lies in it being the climactic point for many converging activities. It helps to strengthen the emphasis on goal orientation while reducing focus on activity orientation, (RUSS & DRANGAN 2016). **Figure 4** (B) shows a milestone chart with seven milestones.

The **Line of Balance** (LOB) is a method of showing the repetitive work that may exist in a project as a single line on a graph. Unlike a Bar Chart, which shows the duration of a particular activity, a LOB Chart shows the rate at which the work that makes up all of the activities has to be undertaken to stay on schedule, the relationship of one trade or process to the subsequent trade or process is defined by the space between the lines (PAI; VERGUESE & RAI, 2013).

Gantt and milestone charts can be used to develop the PERT network, as shown in **Figure 4**. The Gantt chart in **Figure 4** (A) can be converted to the milestone chart in **Figure 4** (B) by defining the milestones of each one of the events. By then determining the relationship between the events on different bars in the milestone chart, we can construct the PERT chart in **Figure 4** (C).

Figure 4 Bar Chart to PERT Chart conversion



Source: KERZNER (2009)

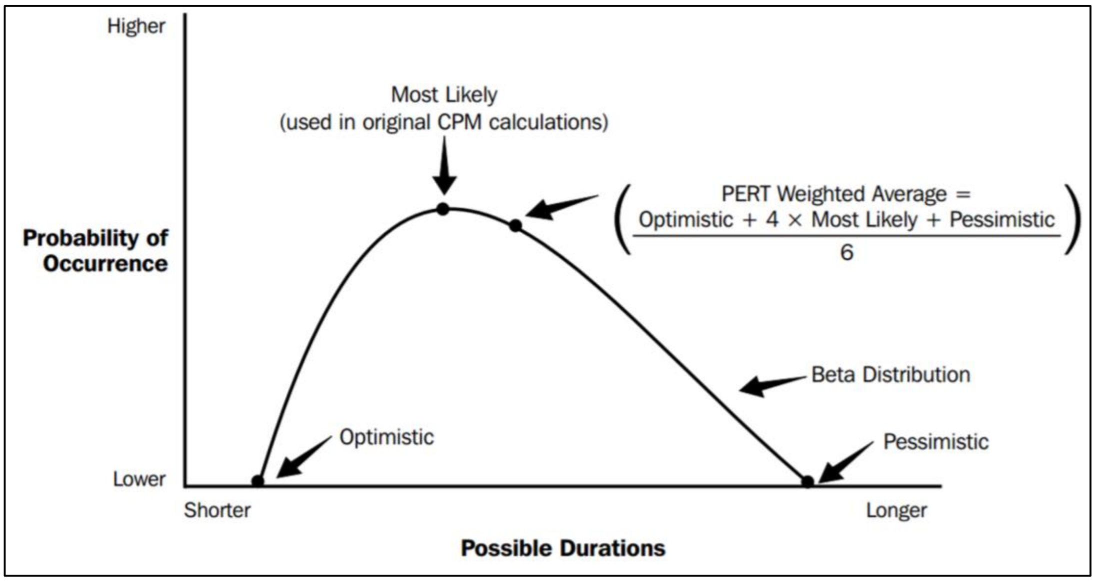
2.1.2 Networking Scheduling Techniques

Network models can be used as an aid in scheduling large complex projects that consist of many activities. If the duration of each activity is known with certainty, then the **Critical Path Method** (CPM) can be used to determine the length of time required to complete a project. CPM can also be used to determine how long each activity in the project can be delayed without delaying the completion time of the project. This amount of time is called Total Float. Free Float is the amount of time that an activity can be delayed without delaying the early start date of any posterior activity, (WINSTON, 2004).

If, however, the duration of the activities is not known with certainty, the **Program Evaluation and Review Technique** (PERT) can be used to estimate the probability that the project will be completed by a given deadline (WINSTON, 2004).

To apply CPM and PERT, we need a list of the activities that make up the project. The project is considered to be completed when all activities have been completed. For each activity, there is a set of activities (called predecessors of the activity) that must be completed before the activity begins. A project network is used to represent the precedence relationship between activities, (WINSTON, 2004). **Figure 5** presents the PERT duration calculation for a single activity.

Figure 5 PERT Duration Calculation for a Single Activity



Source: KERZNER (2009)

The **Graphical Evaluation and Review Technique** (GERT) allows for a probabilistic treatment of both network logic and activity duration estimates (i.e., some activities may not be performed at all, while some may be performed only in part, and others may be performed more than once), (PMBOK GUIDE, 2000).

The **Precedence Diagram Method** (PDM) is a method of constructing a project network diagram that uses boxes or rectangles (nodes) to represent the activities, connecting them with arrows that indicate the dependencies, (PMBOK GUIDE, 2000).

Advantages of network scheduling techniques include, (KERZNER, 2009):

- They form the basis for all planning and predicting and help management decide how to use its resources to achieve time and cost goals.
- They provide visibility and enable management to control “one-of-a-kind” programs.
- They help management evaluate alternatives by answering questions such as how time delays will influence project completion, where slack exists between elements, and what elements are crucial to meet the completion date.
- They provide a basis for obtaining facts for decision-making.
- They utilize a so-called time network analysis as the basic method to determine manpower, material, and capital requirements, as well as to provide a means for checking progress.
- They provide the basic structure for reporting information.
- They reveal interdependencies of activities.
- They facilitate “what if” exercises.
- They identify the longest path or critical paths.
- They aid in scheduling risk analysis

2.1.3 Heuristics for Scheduling

Traditionally, the resource-constrained project-scheduling problem is tackled using the following methods (WEI; LIU & TSAI, 2002):

- **Visual Inspection**: arrange the resource utilization sequence by visually inspecting the project network so as to provide limited resources to the activities in the critical path. The main objective is to complete the project as soon as possible. This method is only suitable for small and non-complicated projects requiring few resources.
- **Graphical Method**: this method provides limited resources to the activities in the critical path. Here, two rules must be followed:

    - Limited sources should be first assigned to the activity with less total float when several activities are undertaken at the same time;
    - Resources should be first assigned to the activity with less duration if the total floats of activities are identical.
- **Heuristic method:** this method is suitable for large and complex projects. It seeks the near-optimal project schedule. One possibility is to use a scheduling list, as the resources become available, activities are released using such criteria as:
    - The activity with the longest duration first (LAF);
    - The activity with the shortest duration first (SJF);
    - First come first serve (FCFS);
    - The activity with the latest finish time first (LFT)
    - The activity with the smallest earliest completion time first (MEF)
    - The activity with the minimum slack time first (MSF)
    - The activity with the largest resource over activity time ratio first (ROT)

2.1.4 Application of PMST in the Software Factory

The software factory problem has some characteristics that do not allow the direct application of project management scheduling techniques to the software factory problem. These are:

- The CPM method is not ideal in this case because it seeks to find the project's critical activities. Here, however, all of the projects are deemed critical since none of them can be delayed without risking a delay in the entire project. There is a fixed relationship of precedence between the activities as established in the Gantt Chart. Consequently, the total float for all of the activities is zero.
- The Gantt chart of the projects is already given, and the due date for these projects is as soon as possible.
- The main objective of the problem is to not only provide the best allocation of the activities of a project, but also the best allocation for all the projects. Inside one project, all activities have the same priority, because one can only start an activity when it is possible for all the others to start as well.

According to PINEDO (2008), one of the most common types of problems in project planning in the construction industry is $P\infty|prec|C_{max}$. This problem denotes a scheduling problem with n jobs subject to precedence constraints and an unlimited number of machines (or resources) working in parallel. The total time of the entire project must be minimized. Considering the PINEDO (2008) approximation of the project planning problems to parallel machines, it is therefore reasonable to review classic scheduling problems.

2.2 Scheduling Problems

As stated by POTTS; CHEN & WOEGINGER (1998), the machine scheduling problems can be described as follows: There are m machines that are used to process n jobs. A schedule specifies, for each machine $i = 1 \dots . m$ and each job $j = 1 \dots n$, one or more time intervals throughout which processing is performed on j by i . A schedule is feasible if there is:

- No overlapping of time intervals corresponding to the same job (so that a job cannot be processed by two machines at once);
- No overlapping of time intervals corresponding to the same machine (so that a machine cannot process two jobs at the same time);
- And if it satisfies various requirements relating to the specific problem type.

The machine environment, the job characteristics and an optimality criterion specify the problem type, (POTTS; CHEN & WOEGINGER, 1998).

2.2.1 Machine Environment

In a **single stage production system**, each job requires one operation. Single stage systems involve either a **single machine** (1), or m machines operating in parallel. In the case of parallel machines, each machine has the same function. Three cases deserve special attention, as per POTTS; CHEN & WOEGINGER (1998):

- **Identical parallel machines** (Pm) in which each processing time is independent of the machine performing the job. There are m identical machines in parallel;
- **Uniform parallel machines** (Qm) in which the machines operate at different speeds but are otherwise identical. There are m machines in parallel with different speeds;
- **Unrelated parallel machines** (Rm) in which the processing time of a job depends on the machine assignment. There are m different machines in parallel.

In a **multi-stage production system,** the jobs require operations at different stages. There are three main types of multi-stage systems. All such systems that we consider comprise s stages, each having a different function (POTTS; CHEN & WOEGINGER, 1998).

- In a **flow shop** (Fm) with s stages, the processing of each job goes through the stages $1 \dots s$ in that order. There are m machines in series;
- In an **open shop** (Om), the processing of each job also goes once through each stage, but the routing (that species the sequence of stages through which a job must pass) can differ between jobs and forms part of the decision process. There are m machines.
- In a **job shop** (Jm), each job has a prescribed routing through the stages, and the routing may differ from job to job.

2.2.2 Processing Characteristics and Constraints

The processing requirements of each job j are given (POTTS; CHEN & WOEGINGER, 1998):

- For a single machine and identical parallel machines, p_j is the processing time;
- For uniform parallel machines, the processing time on machine i may be expressed as ${}^{p_j}/_{s_i}$, where s_i is the speed of machine i;
- For unrelated parallel machines, a flow shop and an open shop, p_{ij} is the processing time on machine/stage i;
- And for a job shop, p_{ij} denotes the processing time of the ith operation (which is not necessarily performed at stage i).

We assume that all p_i and p_{ij} are non-negative integers.

According to POTTS; CHEN & WOEGINGER (1998) and PINEDO (2008), in addition to its processing requirements, a job is characterized by the following characteristics:

- **Availability for processing** - each job j may be restricted by its **release date** r_j that determines when it becomes available for processing, and/or by its integer **deadline date** d_j that specifies the time by which it must be completed;
- **Precedence Constraints** ($prec$) - If job j has precedence over job k, then k cannot start its processing until j is completed;
- Whether **interruptions in the processing** of its operations are allowed - Some scheduling models allow **pre-emption** ($prmp$), the processing of any operation may be interrupted and resumed at a later time on the same or on a different machine;
- **Machine availability constraints** ($brkdwn$) imply that a machine may not be continuously available. The periods that a machine is not available are assumed to be

fixed (e.g., due to shifts or scheduled maintenance). Also referred to as machine breakdowns;

- **Machine eligibility restrictions** (Mj) means that not all m machines are capable of processing job j;
- Other processing restrictions and constraints are sequence dependent setup times (s_{jk}), job families ($fmls$), batch processing ($batch(b)$), permutation ($prmu$), blocking ($block$), no-wait (nwt) and recirculation. ($rcrc$).

2.2.3 Optimal Criteria

For each job j, an integer $due\ date\ d_j$ and a positive integer $weight\ w_j$ may be specified and some statistics can be computed, (POTTS; CHEN & WOEGINGER, 1998) and (PINEDO, 2008):

- The $completion\ time\ C_j$
- The $flow\ time\ \ F_j\ =\ C_j - r_j$
- The $lateness\ L_j\ =\ C_j - d_j$
- The $earliness\ E_j = \max\{d_j - C_j, 0\}$
- The $tardiness\ T_j =\ \max\{C_j - d_j, 0\}$
- And the $unit\ penalty\ U_j = 1\ if\ C_j > d_j, U_j = 0$ otherwise.

Moreover, if f_j is a regular objective function, i.e., a non-decreasing cost function, then the cost of job j is defined $f_j = f_j(\ C_j)$.

As presented by POTTS; CHEN & WOEGINGER (1998) and PINEDO (2008), some commonly used optimality criteria involve the minimization of:

- The maximum completion time, or makespan, $C_{max} = max_jC_j$. It is equivalent to the completion time of the last job to leave the system. A minimum makespan usually implies a good utilization of the machine(s);
- The maximum lateness $L_{max} = max_jL_j$. It measures the worst violation of the due dates;
- The maximum cost $f_{max} = max_jf_j$;
- The maximum earliness $E_{max} = max_jE_j$;

- The total (weighted) completion time $\sum_j (w_j) C_j$;
- The total (weighted) flow time $\sum_j (w_j) F_j$;
- The total (weighted) earliness $\sum_j (w_j) E_j$;
- The total (weighted) tardiness $\sum_j (w_j) T_j$;
- The (weighted) number of late jobs $\sum_j (w_j) U_j$;
- The total cost $\sum_j f_j$, where each maximization and each summation is taken over all jobs j.

Also, in some situations more than one of these criteria must be considered.

The allocation problem of the software factory can be classified as a $Pm|prec, Mj, brkdwn|\sum w_l C_l$ problem following PINEDO (2008) notation. This delineates a scheduling problem of parallel identical machines subject to precedence constraints, machine eligibility restrictions and machine availability constraints with the objective function of the minimization of the total weighted completion time of the projects.

A set of programmers with the same skill can be approximated to parallel identical machines from the point of view of an activity that needs a specific skill to be performed. Each activity is going to have a set of parallel programmers to be allocated.

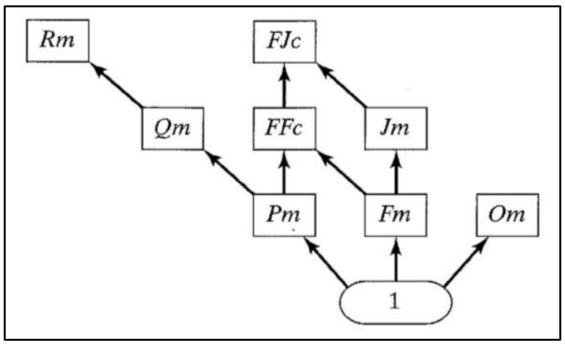
2.2.4 Complexity Hierarchy in Scheduling Problems

A significant amount of research in deterministic scheduling has been devoted to finding efficient, so-called polynomial time algorithms for scheduling problems. However, many scheduling problems do not have a polynomial time algorithm. These problems are called *NP-hard* problems (PINEDO, 2008).

In complexity terminology, $Pm||\sum w_j C_j$ reduces $Pm|prec, Mj, brkdwn|\sum w_j C_j$, indicating that the first is only a special case of the second. According to PINEDO (2008), a problem $Pm||\sum w_j C_j$ is NP-hard, which implies that $Pm|prec, Mj, brkdwn|\sum w_l C_l$, the formulation of the software factory problem, is also NP-hard.

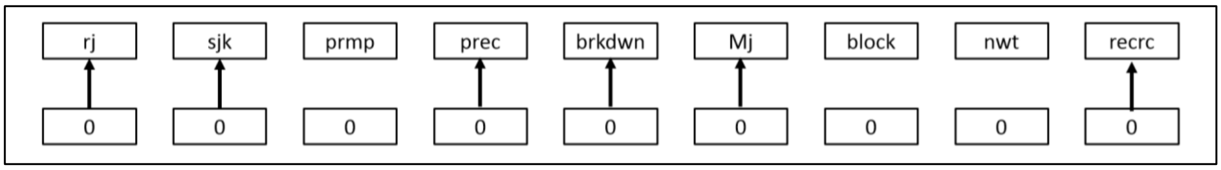
The Complexity Hierarchies of Deterministic Scheduling problems (CHDSP) is presented in the next figures. As **Figure 7** shows, the addition of precedence constraints ($prec$) and machine eligibility restrictions (Mj) affects the complexity of the model, making it more complex. The box with “zero” show that the scheduling does not have that restriction. **Figure 6** and **Figure 8** show the CHDSP for machine environments and for the objective function, respectively.

Figure 6 CHDSP for machine environments



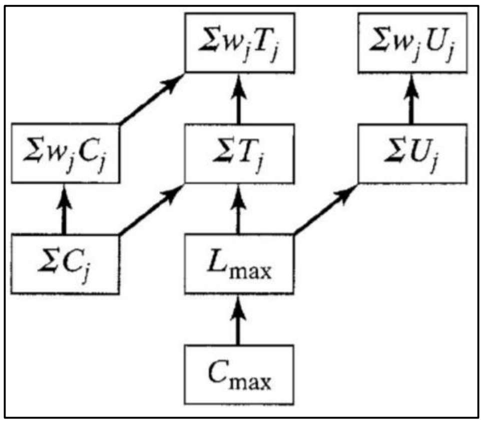
Source: PINEDO (2008)

Figure 7 CHDSP for processing restrictions and constraints



Source: PINEDO (2008)

Figure 8 CHDSP for objective functions



Source: PINEDO (2008)

2.2.5 Single Stage Multi Machine Problems

The model that most closely resembles the problem of the allocation of programmers is the Single Stage Multi Machine Problem. A classical formulation for the minimization of total earliness and tardiness for the Single Stage Multi Machine Problem is presented in ARENALES et al (2015).

Parameters

The integer and non-negative parameters:

p_{ik}	Processing time of job i in machine k
s_{ijk}	Set up time of machine k to processes job i immediately after job j
d_i	Due date of job i
M	A large number

Variables

X_{ijk}	1 if job i precedes immediately job j on machine k, 0 otherwise
C_{ik}	Completion time of job i on machine k

$E_j = \max\{d_j - C_j, 0\}$ = Earliness of job i

$T_j = \max\{C_j - d_j, 0\}$ = Tardiness of job i

Model

In the following formulation:

$$\min \sum_{i=1}^{n} (T_i + E_i) \qquad (2.1)$$

$Subject\ to$

$$\sum_{k=1}^{m} \sum_{i=0}^{n} X_{ijk} = 1\,, \qquad j = 1, \ldots, n \qquad (2.2)$$

$$\sum_{j=1}^{n} X_{0jk} \le 1, \qquad k = 1, \ldots, m \qquad (2.3)$$

$$\sum_{\substack{i=0 \\ i\neq h}}^{n} X_{ihk} - \sum_{\substack{j=0 \\ j\neq h}}^{n} X_{hjk} = 0, \qquad h = 1,\dots,n \;\; k = 1,\dots,m \qquad (2.4)$$

$$C_j \geq C_i + \sum_{k=1}^{m} X_{ijk}\left(p_{ik} + s_{ijk}\right) + H \cdot \left(\sum_{k=1}^{m} X_{ijk} - 1\right), \qquad i = 0,\dots,n, j = 1,\dots,n \qquad (2.5)$$

$$T_j \geq C_j - d_j, \qquad i = 1,\dots,n \qquad (2.6)$$

$$E_j \geq d_j - C_j, \qquad i = 1,\dots,n \qquad (2.7)$$

$$X \in B^{m(n+1)(n+1)}, \qquad T \in R_+^n, \quad E \in R_+^n \qquad (2.8)$$

The objective function (2.1) minimizes the total sum of the tardiness and earliness of the jobs. The set of constraints (2.2) imposes that each job j has only one predecessor job in only one machine. The set of constraints (2.3) ensures that each machine k, if allocated, has only one job processing sequence. The set of constraints (2.4) ensures that each job j has only one job immediately before, except job 0, which is the beginning and the end of the processing sequence. If $X_{ijk} = 1$, the set of constraints (2.5) implies that

$$C_j \geq C_i + p_{ik} + s_{ijk}$$

In other words, it means that if the job j succeeds the job i in the machine k schedule, then the completion time of job j, Cj, is after or equal to the completion time of job i, Ci, plus the duration of job i, p_{ik} and the set-up time s_{ijk}. If $X_{ijk} = 0$, the set of constraints (2.5) does not apply:

$$C_j - C_i \geq -M$$

The set of constraints (2.6) defines the tardiness of each job. The set of constraints (2.7) defines the earliness of each job. The constraint (2.8) indicates the type of each decision variable.

The above formulation will be the basis for that of the problem being considered. The subsequent section presents a short description of some related works before the next chapter where the new formulation is presented.

2.3 Related works

GOMES; NEVES & SOUZA (2014) addresses the problem of project scheduling with resource and precedence constraints (RCPSPRP). Two conflicting objectives are considered in the problem: the makespan minimization and the minimization of the total weighted start time of the activities. The precedence constraints have the same definition presented in PINEDO (2008), an activity cannot start while its precedent activities have not yet been finished. To solve the problem, five heuristic algorithms were implemented: Multi Objective GRASP (MOG), Multi Objective Variable Neighbourhood Search (MOVNS), MOG using VNS as local search, denominated GMOVNS; MOVNS with intensification procedure denominated MOVNS_I; and Pareto Iterated Local Search (PILS). The authors addressed a similar problem in terms of the characteristic of the projects, with resource and precedence constraints, despite the differing statement of the precedence constraint in the software factory problem.

The objective of the makespan minimization is not addressed in the software factory problem, and the second objective of the minimization of the total weighted start time of the activities looks similar to the minimization of the total weighted completion time at first sight. In GOMES; NEVES & SOUZA (2014), however, the weight factor does not represent the importance of the activity (as is the case in this present work), but an issue that is widely discussed in project management: if it is worth making a large investment to start performing the activities as soon as possible. The objective function is given by $n\ \sum_{i=1}^{n} \frac{w_i}{s_i}$, where n is the number of activities, s_i the start time of execution and w_i is the cost of executing the activity. GOMES; NEVES & SOUZA (2014) address the allocation of only one project, and not a list of projects as the software factory problem.

YAMASHITA (2003) addresses a resource availability cost problem RACP, where the objective is to find the best cost allocation of resources, so that the makespan does not exceed a certain delivery date for the project and the precedence relationships between activities are upheld. The resources are unlimited with a cost of availability given by a non-decreasing function. YAMASHITA (2003) considers situations where there is uncertainty regarding the duration of the activities and where the processing times of the activities are fixed. The method used to find the solution for the problem was the meta heuristic Scatter Search.

LEE (1996) presented an extensive study of single and parallel machine scheduling problems with an availability constraint, with respect to various performance measures. There are two cases considered: resumable and nonresumable cases. In the resumable case, pre-emption is allowed. If an operation cannot be finished before the machine's period of unavailability, then it can continue after the machine is available again. The nonresumable case describes the case where pre-emption is not allowed. The disrupted operation has to totally restart rather than continue.

The MA; CHU & ZUO (2010) paper summarizes the many results involving deterministic scheduling problems with availability constraints motivated by preventive maintenance. The machine may become unavailable due to a breakdown or preventive maintenance during the scheduling period. When unavailability constraints are due to preventive maintenance, two cases are possible. One is the case where maintenance periods are fixed in advance, deemed a deterministic case. The other is that maintenance periods are also decision variables, i.e., scheduling processing of jobs and maintenance of a machine simultaneously. When unavailability is due to machine breakdowns, it is a stochastic case, i.e., the breakdowns and repair processes are random. It is also possible that a machine is unavailable at the beginning of the scheduling period because it continues to process uncompleted jobs scheduled during the previous scheduling period. This case can be classified as a deterministic one. Sometimes, an unavailability interval is referred to as a hole. This paper only deals with the deterministic case. The software factory programmers' availability constraint is a deterministic case, the dates and durations of their lack of availability are already known and fixed as an input data.

LI (2006) considers uniform parallel machine scheduling problems with unit-length jobs where every job is only allowed to be processed on a specified subset of machines. Let $\{J_1, J_2, \ldots, J_n\}$ be a given set of jobs and $\{M_1, M_2, \ldots, M_m M1, M2, \ldots, Mm\}$ be a given set of machines, where $m \leqslant n$. For each $j = 1, 2, \ldots, n$, let $M_j \subseteq \{M_1, M_2, \ldots, M_m\}$ be the set of machines that is capable of processing job J_j. In the software factory problem, the resource eligibility restrictions are due to the fact that programmers are only able to perform activities for which they have the necessary skill.

BROWNING & YASSINE (2010) address the case of a portfolio of concurrent projects with identical starting times. Each project consists of an activity network that draws from common pools of multiple types of resources which are typically not large enough for all the activities to work concurrently. The goal is to prioritize activities so as to optimize an objective function such as minimizing the delay for each project or the overall portfolio. Such is the basic resource-constrained multi-project scheduling problem (RCMPSP). According to BROWNING & YASSINE (2010), both the RCPSP and the RCMPSP are strongly NP-hard, meaning that there are no known algorithms for finding optimal solutions in polynomial time. The method proposed by the author for the resolution was a comprehensive analysis of 20 priority rules on 12,320 test problems.

All the searched works about project scheduling strived to find a solution via a heuristic method. The present work will present a MILP scheduling formulation for the project scheduling problem, a resource-constrained multi-project scheduling with precedence relations, resource availability constraints and resource eligibility restrictions. The present works also presents a developed heuristic method for allocation.

3. MILP MODEL

This chapter presents the mixed integer linear programming (MILP) model developed to represent the problem of the software factory. First, the complete model is introduced and presented, including indexes, variables, constants, parameters, the objective function and constraints. An explanation is provided for each of the model constraints. After the description of the model, a small example is presented, with a subsequent application of the MILP model and finally the preliminary results are discussed.

3.1 The Mathematical Formulation

The allocation problem of the software factory's programmers can be defined as a $Pm|prec, M_j, brkdwn|\sum w_l C_l$ problem, following the PINEDO (2008) notation. This problem denotes a scheduling problem with n jobs to a limited number of machines (or resources) in parallel. The problem is subject to precedence constraints, machine availability constraints and machine eligibility restrictions. The objective function seeks to minimize the total weighted completion time of all the projects.

The definition of the software factory problem as a $Pm|prec, M_j, brkdwn|\sum w_l C_l$ scheduling problem derives from a few considerations:

- The programmers of the software factory are to be represented by the machines of the problem;
- The activities of the projects are the jobs to be processed;
- The lack of availability and the events together are called events;
- The programmers have eligibility restrictions according to their skills;
- The precedence constraints of the activities are not denoted solely by "one or more jobs have to be completed before another job is allowed to start its processing". The precedence constraints of the software factory are denoted by one activity must start/finish an exact number of days before/after another activity in the same project;
- The programmers' lack of availability are the machine availability constraints;
- The completion time considered in the objective function is the one of the projects, not the activities. The completion time of a project is the completion time of the last activity to be completed within the project;

- In the classic formulation of the problem, $Pm|prec, M_j, brkdwn|\sum w_l C_l$, the activities do not belong to a project, as the model perceives them as being independent. The concept of considering the activities of being part of a project is an addition to the problem made by the author;
- The activities are nonresumable and pre-emption is not allowed. In other words, it cannot be interrupted by a lack of availability or by another activity.

The mathematical formulation of the software factory problem is presented below:

Definitions

$\mathcal{P}$ Set of all projects;

$\mathcal{A}^l$ Set of all activities belonging to project l;

$\mathcal{A}$ Set of all activities $\mathcal{A} = \bigcup_{l \in \mathcal{P}} \mathcal{A}^l$;

$\mathcal{B}^k$ Set of all lack of availability of programmer k;

$\mathcal{B}$ Set of all lack of availability $\mathcal{B} = \bigcup_{k \in \mathcal{M}} \mathcal{B}^k$;

$\mathcal{E}$ Set of all events $\mathcal{E} = \mathcal{A} \cup \mathcal{B}$;

$\mathcal{M}$ Set of all programmers available;

$\mathcal{M}^i$ Set of programmers capable of performing the event i. In case event i is a lack of availability, the programmer is capable of performing only its own lack of availability.

Parameters

H A large number;

p_i Duration of event i ;

ω_l Weight of importance of project l;

Π Set of ordered pairs of activities, (i, j) that are associated through “project-dependent” precedence constraints in such a way that if both are in the same project, then the completion time of activity i must precede the completion time of event j;

$prec_{ij}$ Number of days the completion time of event i must precede the completion time of event j, $(i,j) \in \Pi$;

Cb_i Completion time of the lack of availability $i \in \mathcal{B}$.

Variables

X_{ijk} 1 if event i immediately precedes event j for programmer $k \in \mathcal{M}^i \cap \mathcal{M}^j$, 0 otherwise;

C_i Completion time of event i;

CP_l Completion time of project l.

Model

$$\min \sum_{l \in \mathcal{P}} \omega_l \cdot CP_l \tag{3.1}$$

$Subject\ to$:

$$\sum_{k \in \mathcal{M}^i \cap \mathcal{M}^j} \sum_{i \in \mathcal{E}} X_{ijk} = 1\,, \qquad \forall\, j \in \mathcal{E}, j \neq 0 \tag{3.2}$$

$$\sum_{\substack{j \in \mathcal{E} \\ j \neq 0}} X_{0jk} \leq 1, \qquad \forall\, k \in \mathcal{M}^j \tag{3.3}$$

$$\sum_{\substack{i \in \mathcal{E} \\ i \neq h}} X_{ihk} - \sum_{\substack{j \in \mathcal{E} \\ j \neq h}} X_{hjk} = 0, \qquad \begin{array}{l} \forall\, h \in \mathcal{E}, h \neq 0 \\ \forall\, k \in \mathcal{M}^i \cap \mathcal{M}^j \cap \mathcal{M}^h \end{array} \tag{3.4}$$

$$C_j \geq C_i + \sum_{k \in \mathcal{M}^i \cap \mathcal{M}^j} X_{ijk} \cdot p_j + H \cdot \left(\sum_{k \in \mathcal{M}^i \cap \mathcal{M}^j} X_{ijk} - 1 \right) \qquad \forall\, i,j \in \mathcal{E}, \; j \neq 0 \qquad (3.5)$$

$$C_j = C_i + prec_{ij}, \qquad (i,j) \in \Pi \qquad (3.6)$$

$$C_i = Cb_i, \qquad \forall\, i \in \mathcal{B} \qquad (3.7)$$

$$CP_l \geq C_i, \qquad \forall\, i \in \mathcal{A}^l, \; \forall\, l \in \mathcal{P} \qquad (3.8)$$

$$X_{ijk} \in \{1,0\} \qquad \forall\, i,j \in \mathcal{E}, \; \forall\, k \in \mathcal{M}^i \cap \mathcal{M}^j \qquad (3.9)$$

$$C_i \in \mathbb{R}_+ \qquad \forall\, i \in \mathcal{E} \qquad (3.10)$$

$$CP_l \in \mathbb{R}_+ \qquad \forall\, l \in \mathcal{P} \qquad (3.11)$$

The objective function (3.1) minimizes the sum of the completion time of each project weighted by the importance of the project. The global optimum for all the projects is the objective. The set of constraints (3.2) imposes that each event j has only one preceding event and it is allocated to only one programmer k. The set of constraints (3.3) ensures that each programmer k, if allocated, has only one event sequence. The set of constraints (3.4) ensures that each event has only one event immediately before, except event 0, which is the beginning and the end of a sequence of events. Event 0 is a fictional event that marks the beginning and the end of the performing order of each programmer. Every programmer, if allocated to some activity, will start and finish the performing order with event 0. If $X_{ijk} = 1$, the set of constraints (3.5) implies that

$C_j \geq C_i + p_j$

In other words, it means that if event j succeeds event i in the schedule of programmer k, the completion time of event j, Cj, is after or equal to the completion time of event i, Ci, plus the duration of event j, p_j. If $X_{ijk} = 0$, the set of constraints (3.5) does not apply:

$C_j - C_i \geq -H$

The set of constraints (3.6) defines that if activity i and j are in the same project, then there is a time relationship between their completion times that must be ensured. The set of constraints (3.7) determines that if activity i is a lack of availability, then the completion time of i, C_i, is known, and it is equal to Cb_i. The set of constraints (3.8) ensures that the completion time of project l is after or equal to the completion time of every activity i that composes this project. The constraints (3.9), (3.10) and (3.11) indicate the type of each decision variable.

Table 5 presents the maximum quantity of restrictions for each set of restrictions.

Table 5 Quantity of restrictions for each set of restrictions

Set of restrictions	Quantity of restrictions
(3.2)	$\lvert\mathcal{E}\rvert^2 \cdot \lvert\mathcal{M}\rvert$
(3.3)	$\lvert\mathcal{M}\rvert$
(3.4)	$\lvert\mathcal{E}\rvert^2 \cdot \lvert\mathcal{M}\rvert$
(3.5)	$\lvert\mathcal{E}\rvert^2 \cdot \lvert\mathcal{M}\rvert$
(3.6)	$\lvert\Pi\rvert$
(3.7)	$\lvert\mathcal{B}\rvert$
(3.8)	$\lvert\mathcal{A}\rvert$

The total quantity of restrictions of the problem is $|\mathcal{B}| + |\mathcal{A}| + |\Pi| + |\mathcal{M}| + 3 \cdot |\mathcal{E}|^2 \cdot |\mathcal{M}|$

3.2 Example

This section presents a pilot problem with the possibility of understanding how the model is making programming decisions for allocation, as well as what is the general behaviour of the variables. Finally, the validity of the model for the pilot problem is discussed.

3.2.1 Data

The pilot problem has two projects, A and B, each one composed of 3 activities. There are three possible skills demanded to execute these activities. Four programmers are available to perform the project's activities. Each one of the programmers has their own schedule and the lack of availability for each programmer is given.

Table 6 and **Table 7** show the Gantt chart for projects A and B. As previously discussed, a project can have its starting time delayed, but once the project has started, it cannot be belated. The duration of the activities is fixed, as is the duration of the project. The project's duration is given by the difference between the first day of the first activity to start and the last day of the last activity to finish. The colored cells indicate that the activity should be done that specific day. If activity A1 starts on December 3rd, A2 must start December 4th and A3 must begin on December 5th, the first, second and third day of the project, respectively.

Table 6 Example Problem Project A Gantt Chart

	Days of the Project				
	1st	2nd	3rd	4th	5th
A1	🟩 **A1**	🟩			
A2		🟩	🟩 **A2**	🟩	
A3			🟩	🟩 **A3**	🟩

Table 7 Example Problem Project B Gantt Chart

	Days of the Project				
	1st	2nd	3rd	4th	5th
A4			🟦	🟦 **A4**	🟦
A5	🟦 **A5**	🟦			
A6		🟦	🟦 **A6**	🟦	

Table 8 provides a description of each activity to be allocated. The characteristics of an activity are the project they belong, the skill they demand and their duration in days.

Table 8 Description of Activities of the Example Problem

Activity	Project	Skill demanded	Duration
A1	A	S1	2
A2	A	S2	3
A3	A	S3	3
A4	B	S1	3
A5	B	S2	2
A6	B	S3	3

Table 9 shows the Map of Skills of each one of the four programmers available. The shaded spaces indicate that programmer P has the skill S. For example, programmer P1 possesses the skills S2 e S3.

Table 9 Map of Skills of the Example Problem

	S1	S2	S3
P1		▨	▨
P2	▨		
P3	▨	▨	
P4		▨	▨

Table 10 shows the Programmers' Schedule for the first 10 days of the current month. The following days of the month are assumed to be free. The coloured spaces are the lack of availability of the programmers, denoted by B#.

Table 10 Programmers' Schedule of the Example Problem

	1	2	3	4	5	6	7	8	9	10
P1				**B1**						
P2		**B2**								
P3						**B3**				
P4			**B4**					**B5**		

Table 11 shows the characteristics of the lack of availability for each programmer. The lack of availability and the activities are called events.

Table 11 Description of the Programmers' Lack of Availability of the Example Problem

Lack of Availability	Programmer	Duration in Days	Completion Time
B1	P1	2	5
B2	P2	2	3
B3	P3	1	6
B4	P4	2	4
B5	P4	1	8

Table 12 shows the relation between the activities and the programmers. The colored spaces indicate that programmer P is capable of performing activity A, because they possess the skill required for that activity. **Table 12** is a result of the information contained in **Table 8** and **Table**

9. Activity 0 is a fictional activity that starts and ends the performing list of each active programmer.

Table 12 Relation between the Events and the Programmers of the Example Problem

	A0	A1	A2	A3	A4	A5	A6	B1	B2	B3	B4	B5
P1	▨		▨	▨		▨	▨	▨				
P2	▨	▨			▨				▨			
P3	▨	▨	▨		▨	▨				▨		
P4	▨		▨	▨		▨	▨				▨	▨

Table 13 shows the priority scale between project A and project B. The higher the value of the priority, the higher the priority of the project.

Table 13 Projects' Priority

Project	**Priority**
A	1
B	2

3.2.2 Model

The following section presents the model for the example problem, presented in the previous section.

Sets

$\mathcal{P}$ Set of all Projects:

Projects 0, A and B

$\mathcal{A}^l$ Set of all activities belonging to Project l:

$\mathcal{A}^0$: A0

$\mathcal{A}^A$: A1, A2 and A3

$\mathcal{A}^B$: A4, A5 and A6

$\mathcal{A}$ Set of all activities $\mathcal{A} = \bigcup_{l \in \mathcal{P}} \mathcal{A}^l$:

Activities A0, A1, A2, A3, A4, A5 and A6

$\mathcal{B}^k$ Set of all lacks of availability for programmer k:

$\mathcal{B}^1$: B1

$\mathcal{B}^2$: B2

$\mathcal{B}^3$: B3

$\mathcal{B}^4$: B4 and B5

$\mathcal{B}$ Set of all lacks of availability $\mathcal{B} = \bigcup_{k\in\mathcal{M}}\mathcal{B}^k$:

Lack of availabilities B1, B2, B3, B4 and B5.

$\mathcal{E}$ Set of all events $\mathcal{E} = \mathcal{A}\cup\mathcal{B}$:

Events: A0, A1, A2, A3, A4, A5, A6, B1, B2, B3, B4 and B5.

$\mathcal{M}$ Set of all programmers available:

Programmers: P1, P2, P3 and P4.

$\mathcal{M}^i$ Set of programmers capable of performing event i:

$\mathcal{M}^{A0}$: P1, P2, P3 and P4

$\mathcal{M}^{A1}$: P2 and P3

$\mathcal{M}^{A2}$: P1, P3 and P4

$\mathcal{M}^{A3}$: P1 and P3

$\mathcal{M}^{A4}$: P2 and P3

$\mathcal{M}^{A5}$: P1, P3 and P4

$\mathcal{M}^{A6}$: P1 and P3

$\mathcal{M}^{B1}$: P1

$\mathcal{M}^{B2}$: P2

$\mathcal{M}^{B3}$: P3

$\mathcal{M}^{B4}$: P4

$\mathcal{M}^{B5}$: P4

Parameters

H A large number

p_i Duration of event i

p_{A0}	p_{A1}	p_{A2}	p_{A3}	p_{A4}	p_{A5}	p_{A6}	p_{B1}	p_{B2}	p_{B3}	p_{B4}	p_{B5}
0	2	3	3	3	2	3	2	2	1	2	1

ω_l Weight of importance of project l

ω_A	1
ω_B	2

Π The set of ordered pairs of activities, (i, j) that are associated through "project-dependent" precedence constraints in such a way that if both are in the same project, then the completion time of activity i must precede the completion time of event j

(A1, A3)
(A2, A3)
(A5, A4)
(A6, A4)

$prec_{ij}$ The number of days the completion time of event i must precede the completion time of event j, $(i, j) \in \Pi$

$prec_{A1,A3}$	3
$prec_{A2,A3}$	1
$prec_{A5,A4}$	3
$prec_{A6,A4}$	1

cb_i The completion time of the lack of availability $i \in \mathcal{B}$

cb_{B1}	5
cb_{B2}	3
cb_{B3}	6
cb_{B4}	4
cb_{B5}	8

Variables

X_{ijk}	1 if event i precedes event j for programmer $k \in \mathcal{M}^i \cap \mathcal{M}^j$, 0 otherwise
C_i	Completion time of job i
CP_l	Completion time of project l

3.2.3 Results

The preliminary test with the suggested formulation was developed in the IBM ILOG CPLEX Optimization Studio 12.6.0.0 software. The code and the data input used are in Appendix A.

Table 14 shows the final values for the decision variable X_{ijk}. It gives the performing sequence for each one of the programmers.

Table 14 X_{ijk} values for the preliminary test

Event i	**Event j**	**Programmer k**	**Value**
A0	B1	P1	1
A2	A0	P1	1
B1	A2	P1	1
A0	B2	P2	1
A4	A0	P2	1
B2	A4	P2	1
A0	A5	P3	1
A1	A0	P3	1
A5	B3	P3	1
B3	A1	P3	1
A0	B4	P4	1
A3	A0	P4	1
A6	B5	P4	1
B4	A6	P4	1
B5	A3	P4	1
All other possible combinations of i, j, k			0

Table 15 shows the completion time for each one of the activities for the preliminary test.

Table 15 C_i values for the preliminary test

Event i	**Value**
A0	0
A1	8
A2	10
A3	11
A4	8
A5	5
A6	7
B1	5
B2	3
B3	6
B4	4
B5	8

Table 16 presents the completion times for project A and project B. The completion time of the project is given by the longest completion time among its activities. The value found by the objective function is 27, given by: $(w_A \cdot CP_A + w_B \cdot CP_B)$, $(1 \cdot 11 + 2 \cdot 8)$

Table 16 CP_l values for the preliminary test

Project	Value
A	11
B	8

The allocation of the programmers in all activities is showed in **Table 17**. The allocation was done correctly, all the restrictions were respected, the precedence relationship between activities of the same project was maintained, the lack of availabilities were allocated in the correct places and no event is done in parallel for the same programmer. Once the model was validated with a small test, in the next section, it is going to be tested with the real allocation problem.

Table 17 Result of Preliminary Test

		First 11 days of the current month										
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Programmers	**P1**				**B1**	**B1**			**A2**	**A2**	**A2**	
	P2		**B2**	**B2**			**A4**	**A4**	**A4**			
	P3				**A5**	**A5**	**B3**	**A1**	**A1**			
	P4			**B4**	**B4**	**A6**	**A6**	**A6**	**B5**	**A3**	**A3**	**A3**

4. RESOLUTION METHODS AND NUMERICAL EEXPERIMENTS

This chapter begins with the presentation of the data collected. The following sub section shows the results of the MILP model for the complete and partial sets of the real problem data. Subsequently, constructive heuristic theory is briefly presented and a possible heuristic for the problem resolution is proposed. The suggested heuristic is applied to the same small problem of the MILP example. The chapter ends with a comparison between the results of the MILP model and the heuristic model in order to validate it.

4.1 Data Collection

The general manager of the bank's software factory supplied the data set used in one cycle of projects, that was allocated in the first week of December, 2015. This data set includes:

- The **Gantt chart** of 32 projects and the description of 423 activities –**Table 18** shows the simplified Gantt Chart for two real projects. The skill name needed was changed for a reference identification (ID) number in order to keep the confidentiality of the data. The Gantt chart shows all the activities that are part of the project, their duration, the ID of the needed skill and the precedence relationship between them. For example, activity #3 needs 8 hours/day for 9 days with a programmer with skill 89, starting on the same day of activity #1 and it needs to finish two days early.

Table 18 Gantt Chart of 2 Real Projects

ID Project	ID Skill	Activity Duration	ID Activity	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11
#P01	133	11	#1	8	8	8	8	8	8	8	8	8	8	8
#P01	41	11	#2	8	8	8	8	8	8	8	8	8	8	8
#P01	89	9	#3	8	8	8	8	8	8	8	8	8	0	0
#P01	181	11	#6	8	8	8	8	8	8	8	8	8	8	8
#P01	40	9	#7	8	8	8	8	8	8	8	8	8	0	0
#P01	54	11	#8	8	8	8	8	8	8	8	8	8	8	8
#P01	31	9	#9	8	8	8	8	8	8	8	8	8	0	0
#P01	86	9	#10	8	8	8	8	8	8	8	8	8	0	0
#P01	35	9	#11	8	8	8	8	8	8	8	8	8	0	0
#P01	165	9	#12	8	8	8	8	8	8	8	8	8	0	0
#P01	129	9	#13	8	8	8	8	8	8	8	8	8	0	0
#P01	127	9	#14	8	8	8	8	8	8	8	8	8	0	0
#P01	93	9	#15	8	8	8	8	8	8	8	8	8	0	0
#P01	80	9	#16	8	8	8	8	8	8	8	8	8	0	0

ID Project	ID Skill	Activity Duration	ID Activity	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11
#P02	27	11	#17	8	8	8	8	8	8	8	8	8	8	8
#P02	174	11	#18	8	8	8	8	8	8	8	8	8	8	8
#P02	86	9	#19	8	8	8	8	8	8	8	8	8	0	0
#P02	161	11	#20	8	8	8	8	8	8	8	8	8	8	8
#P02	62	9	#21	8	8	8	8	8	8	8	8	8	0	0
#P02	65	11	#22	8	8	8	8	8	8	8	8	8	8	8
#P02	116	9	#23	8	8	8	8	8	8	8	8	8	0	0
#P02	44	9	#24	8	8	8	8	8	8	8	8	8	0	0
#P02	14	9	#25	8	8	8	8	8	8	8	8	8	0	0
#P02	172	9	#26	8	8	8	8	8	8	8	8	8	0	0

Source: Bank's Software Factory

- The **Map of Skills** of 1916 programmers – **Table 19** shows the map of skills of all the 1916 available programmers. The real identification of the programmers was changed for a reference ID number in order to keep their identity secret and the data confidential. The skills ID numbers are the same as that of the Gantt chart. There are 183 skills mapped. Zero (0) means that the programmer does not have the skill; one (1) means the programmer possesses the skill.

Table 19 Map of Skills for 8 programmers and 7 skills

		Skills						
Programmers	**ID**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	1	0	0	0	0	1	0	0
	2	0	0	0	0	0	0	1
	3	0	0	0	0	0	0	0
	4	0	1	1	0	0	0	0
	5	0	1	1	0	0	0	0
	6	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0

Source: Bank's Software Factory

- The **Schedule** for the next two months (December/2015 and January/2016) for the 1916 programmers. The timetable contains 40 working days. **Table 20** shows the schedules of 6 programmers for the first 8 working days of December. Zero (0) means the programmer is free to be allocated in a new project that day; one (1) means the programmer has a lack of availability that day, they may be busy with another project of the previous cycle of projects, or with internal training etc. According to the software factory general manager, it is reasonable to suppose that all the programmers are free from February/2015 onwards.

Table 20 Schedules of 6 programmers for the first working days of December

ID	**02/12/2015**	**03/12/2015**	**04/12/2015**	**07/12/2015**	**08/12/2015**	**09/12/2015**	**10/12/2015**	**11/12/2015**
1	1	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0

Source: Bank's Software Factory

- The **importance list** of the projects with an importance number is assigned to each project, as **Table 21** shows. The importance is assigned a number from 1 to 10, with 10 being the most important and 1 the least important. The importance of each project is decided in a monthly meeting for the prioritization of projects. In this meeting, each business area presents their reasons for demanding the project and why it should be prioritized. Projects that are not deemed as being of a high priority can be delayed to the next allocation cycle. On the other hand, projects with a really high priority can be immediately allocated, and do not need to wait until the allocation of the current cycle.

Table 21 Importance of the projects

Project	Importance
#P01	7
#P02	10
#P03	2
#P04	1
#P05	6
#P06	6
#P07	6
#P08	8
#P09	4
#P10	8
#P11	3
#P12	7
#P13	3
#P14	1
#P15	4
#P16	3
#P17	9
#P18	5
#P19	5
#P20	6
#P21	7
#P22	2
#P23	8
#P24	6
#P25	7
#P26	3
#P27	1
#P28	5
#P29	8
#P30	8
#P31	7
#P32	10

- The **allocation plan** carried out by the company to be compared with the results of this work. The allocation plan has the day each one of the activities started and finished and the programmer who performed it.

It is not possible to show the total data as this would be in violation of the confidentiality agreement.

4.2 MILP Model Resolution

Initially, an exact solution for the problem was tried to be found using the proposed mixed integer linear programming model as expressed in Chapter 3. The exact solution ensures that the global optimum was reached.

The IBM ILOG CPLEX Optimization Studio 12.6.0.0 software was used in order to find the exact solution for the problem. The choice for the CPLEX software was made because it is a free software for the academic environment, with a rather simple language and that the company could easily buy for a reasonable price and replicate the model later. Furthermore, it is a software commonly used in literature, see KRIVONOZHKO; FØRSUND & LYCHEV, (2014), and QUADT & KUHN (2009), as an example.

The code for the implemented model can be seen in **Appendix A.** One of the main advantages of this software is the fact that the model module is separated from the data module, making it easy to change the dataset that is being used. The data module was imported to the IBM ILOG CPLEX software via a connection with an EXCEL spreadsheet that contained it, as the real data was too numerous to input in the IBM ILOG CPLEX's own data module.

The model was set to run on a computer with 16 GB of RAM, running on a Windows 64-bit. After about 1 minute and 39 seconds of pre-processing, the software indicated that all the computer's RAM had been consumed, and thus the execution ended without results. The IBM ILGO CPLEX returned an error message "*not enough memory*".

The specifications of the company's computers are equal to or lower than the one used. To better understand the limitations of the problem, additional tests were performed by varying the size of the problem analysed. The results obtained are shown in **Table 22**.

Table 22 Comparison of the resolution times for the MILP when varying the inputs

	Problem	**1**	**2**	**3**	**4**	**5**	**Real**
Dimensions	**Programmers**	4	110	110	648	1155	1916
	Projects	2	3	3	32	10	32
	Activities	6	41	41	423	132	423
	Lacks of Availability	5	0	40	174	322	576
	Events	11	41	81	455	454	999
	x_{ijk}	484	184910	721710	1,34E+08	2,38E+08	1,91E+09
MILP	**CPU Time**	1,69 s	8,25 s	26,13 s	-	-	-
	Objective Function	27	213	213	-	-	-

- Exceeds the processing capacity

The software finds an exact solution for problem 1, the preliminary test, and for problems 2 and 3, reduced versions of the real problem. Problems 4 and 5 are also a reduced version of the real problem, however they both exceed the software’s memory, even though their size is significantly smaller than that of the real one.

This comparison makes it evident that it is not possible to find the exact solution for the real problem, hence a method for an approximated solution is required. As discussed in section **2.2.4**, the software factory problem is a NP-hard problem.

The NP-hardness of an optimization problem suggests that it is not always possible to find an optimal solution quickly. Therefore, instead of searching for an optimal solution with enormous computational effort, an heuristic algorithm to generate approximate solutions that are close to the optimum with considerably less investment in computational resources is a good choice. The path chosen by this work is to develop a constructive heuristic based on the list scheduling method in order to generate the approximate solution.

4.3 Constructive Heuristic

4.3.1 Literature Review

In contrast to exact methods, which guarantee an optimum solution for the problem, heuristic methods only attempt to yield a good, but not necessarily optimum solution. In addition to the need to find good solutions for difficult problems in a reasonable amount of time, there are other reasons for using heuristic methods, among which MARTÍ & REINELT (2011) highlights:

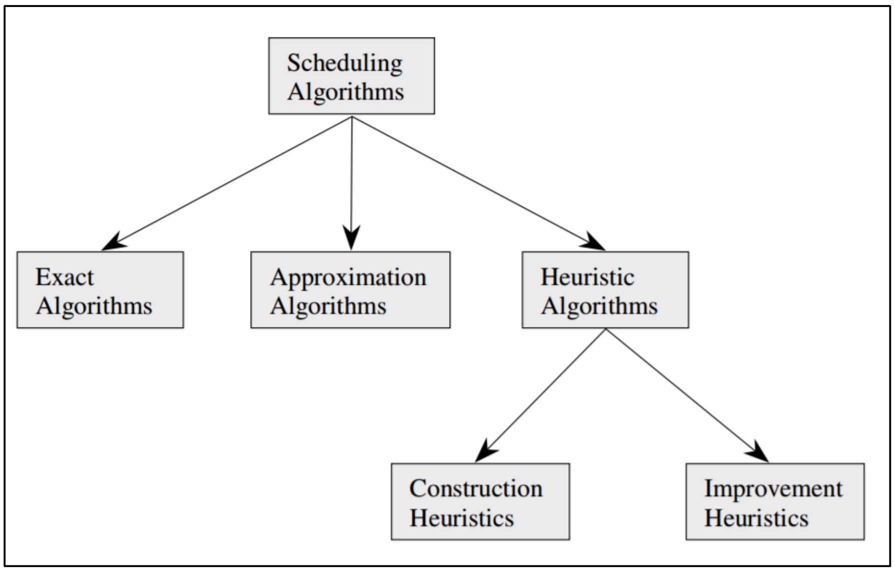
- No method for solving the problem to optimality is known.
- Although there is an exact method to solve the problem, it cannot be used with the available hardware.
- The heuristic method is more flexible than the exact method, allowing, for example, the incorporation of conditions that are difficult to model.
- The heuristic method is used as part of a global procedure that guarantees to find the optimum solution of a problem.

The software factory problem resolution through a heuristic method is mainly due to the second reason. The exact method to solve the problem was developed in the previous section of this work, but the available hardware to process it is not able to find the solution without exceeding its computational capacity. In order to solve the problem by using the exact method, hardware with a lot more processing power is required.

According to SHAKHLEVICH (2005), there are three major types of scheduling algorithms, as illustrated by **Figure 9**:

- **Exact algorithms** can find optimal solutions;
- **Approximation algorithms** produce solutions that are guaranteed to be within a fixed percentage of the actual optimum. Approximation algorithms are fast (have polynomial running time);
- **Heuristic algorithms** produce solutions, which are not guaranteed to be close to the optimum. The performance of heuristics is often evaluated empirically.

Figure 9 Scheduling Algorithms Classification



Source: SHAKHLEVICH (2005)

The scheduling heuristic algorithms are classified in two types: Construction Heuristics and Improvement Heuristics.

Improvement heuristics start with a feasible schedule and try to find a better similar schedule. Each step of the procedure carries out a movement from one solution to another one with a better value. The method terminates when, for a solution, there is no other accessible solution that improves it (MARTÍ & REINELT, 2011).

Constructive heuristics involve building a solution to the problem literally step by step from scratch. Usually they are deterministic methods and tend to be based on the best choice in each iteration. These methods have been widely used in classic combinatorial optimization. (MARTÍ & REINELT, 2011).

Dispatching (or Priority) Rules are the most common constructive heuristics for scheduling problems due to their easy implementation and low requirements in computational power. Although they perform very well in certain cases, it is not possible to point a rule that can be applied to all scheduling problems and perform satisfactorily (XHAFA & ABRAHAM, 2008).

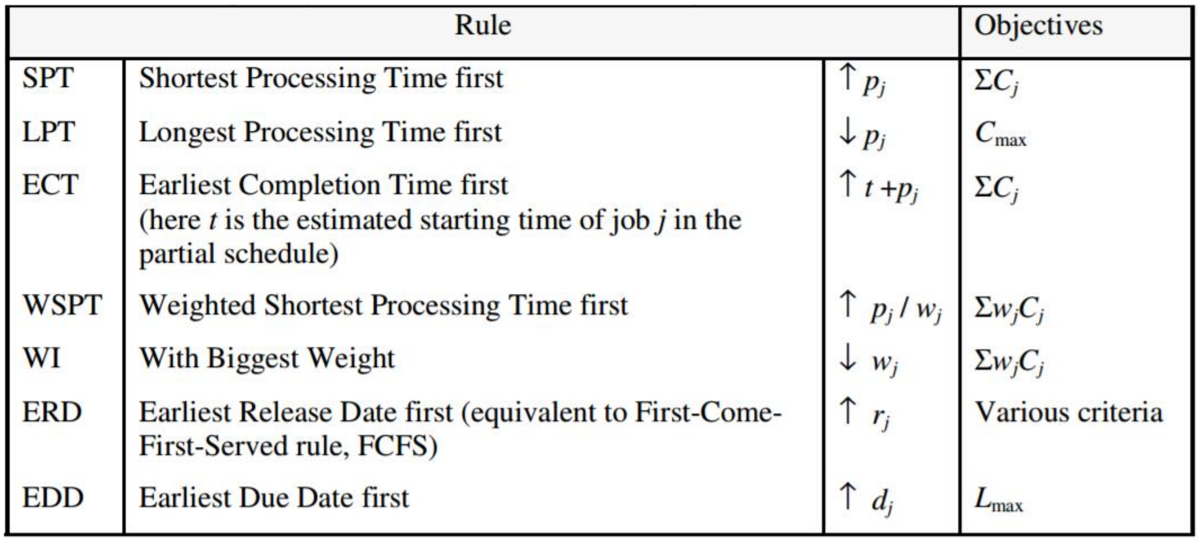
According to BROWNING & YASSINE (2010), priority rule (PR) heuristics are crucial for several reasons:

- More elaborated techniques, like meta-heuristics, improved performance comes at a greater computational expense, meaning that PRs are necessary for very large problems;

- PRs are a component of other (local search-based and sampling) heuristics and “are indispensable” for constructing initial solutions for meta-heuristics;
- PRs are used extensively by commercial project scheduling software due to their speed and simplicity;
- And perhaps the most important argument for PRs is that they are very important in practice.

Table 23 delineates the main priority rules, with the primarily pursued objective function. In the first column, there are the rule acronyms, in the second there are the rule names, in the third there are the methods used and, finally, in the fourth column the objective of the indicated rule. The first rule, for example, follows an increasing processing time (p_j), the objective is to minimize the total completion time.

Table 23 Main Priority Rules



Rule			Objectives
SPT	Shortest Processing Time first	$\uparrow p_j$	ΣC_j
LPT	Longest Processing Time first	$\downarrow p_j$	$C_{\max}$
ECT	Earliest Completion Time first (here *t* is the estimated starting time of job *j* in the partial schedule)	$\uparrow t + p_j$	ΣC_j
WSPT	Weighted Shortest Processing Time first	$\uparrow p_j / w_j$	$\Sigma w_j C_j$
WI	With Biggest Weight	$\downarrow w_j$	$\Sigma w_j C_j$
ERD	Earliest Release Date first (equivalent to First-Come-First-Served rule, FCFS)	$\uparrow r_j$	Various criteria
EDD	Earliest Due Date first	$\uparrow d_j$	$L_{\max}$

Source: PANWALKAR & ISKANDER (1977)

4.3.2 Heuristic Algorithm

The development of a heuristic algorithm in order to find an approximated solution for the software factory problem was necessary due to the computational difficulties faced when finding an optimal solution with the MILP model.

The heuristic resolution proposed in this section is based on priority rules heuristics. First the projects are ordered according to the priority rules. The algorithm can provide a number of different solutions by changing the order of allocation of the projects, using the list scheduling heuristic method. The order of the allocation can be given by multiple criterions:

- Importance order (with biggest weighed WI);
- Weighted shortest processing time first WSPT;
- Increasing number of activities order;
- Decreasing number of activities order;
- Increasing duration of the project order
- Decreasing duration of the project order
- A coefficient of rareness of the skills demanded by the project
- And a lot of combinations of the criterions above.

Two priority rules are going to be applied in this work: Weighted Shortest Processing Time first (WSPT) and With Biggest Weight (WI). This choice was based on the suggestion of PANWALKAR & ISKANDER (1977) for problems with objective function $\min \sum w_j C_j$, minimizing the total weighted completion time of the projects.

Once the projects are sorted, the activities that belong to this project also have to be sorted. The priority rule applied to order the activities within a project is the increasing order of programmers capable of performing this activity. The higher the difficulty in finding a resource to perform the activity, the higher the priority of this activity. This rule was developed after some empirical tests with the current data. The motivation for this rule was the frequency with which activities that demands rare skills were considered impossible or were delayed because all the programmers capable of performing them were already allocated to another activity.

The sorted programmers are a problem input. They are listed in an increasing order of their bank identification number. The longer the programmer has been working for the bank, the smaller their identification number. The priority rule is to allocate activities to more the experienced programmers first.

The algorithm tries to find a capable programmer available in the searched day for each one of the activities. If it is not possible, the algorithm searches in the next working day. The precedence rules and priority rules must be respected.

The proposed algorithm is given by the follow steps:

Step 1 Order the projects according to the selected priority rule. Select the first project of the list

Step 2 Select only the programmers that are capable of performing the activities of the selected project

Step 3 Order the activities of the project in order to prioritize the allocation of the activities with least amount of programmers needed to perform it. The main idea is that the smaller the number of programmers required that are capable of performing the activity, the sooner it should be allocated, so that the chance that an activity ends with no possible programmer is reduced. Select the activity with least amount of programmers needed.

Step 4 Start searching for the first feasible day for the first activity according to the Gantt chart. The algorithm looks for a programmer capable of performing the activity, who is also free on the first feasible day and free for the duration of the selected activity.

If the algorithm does not find a programmer for the selected activity in the first day, it starts to look for a programmer available on the following day and successively for the subsequent days until it finds an available programmer that meets the necessary criteria.

When the algorithm finds a programmer for the first analyzed activity, it starts to look for a programmer for the second activity on the day it should be allocated by following the precedence order established in the problem’s Gantt chart. The algorithm does the same for all others activities.

If the algorithm does not find a programmer for an activity, it returns to the allocation of the first activity and tries to allocate it for the next day.

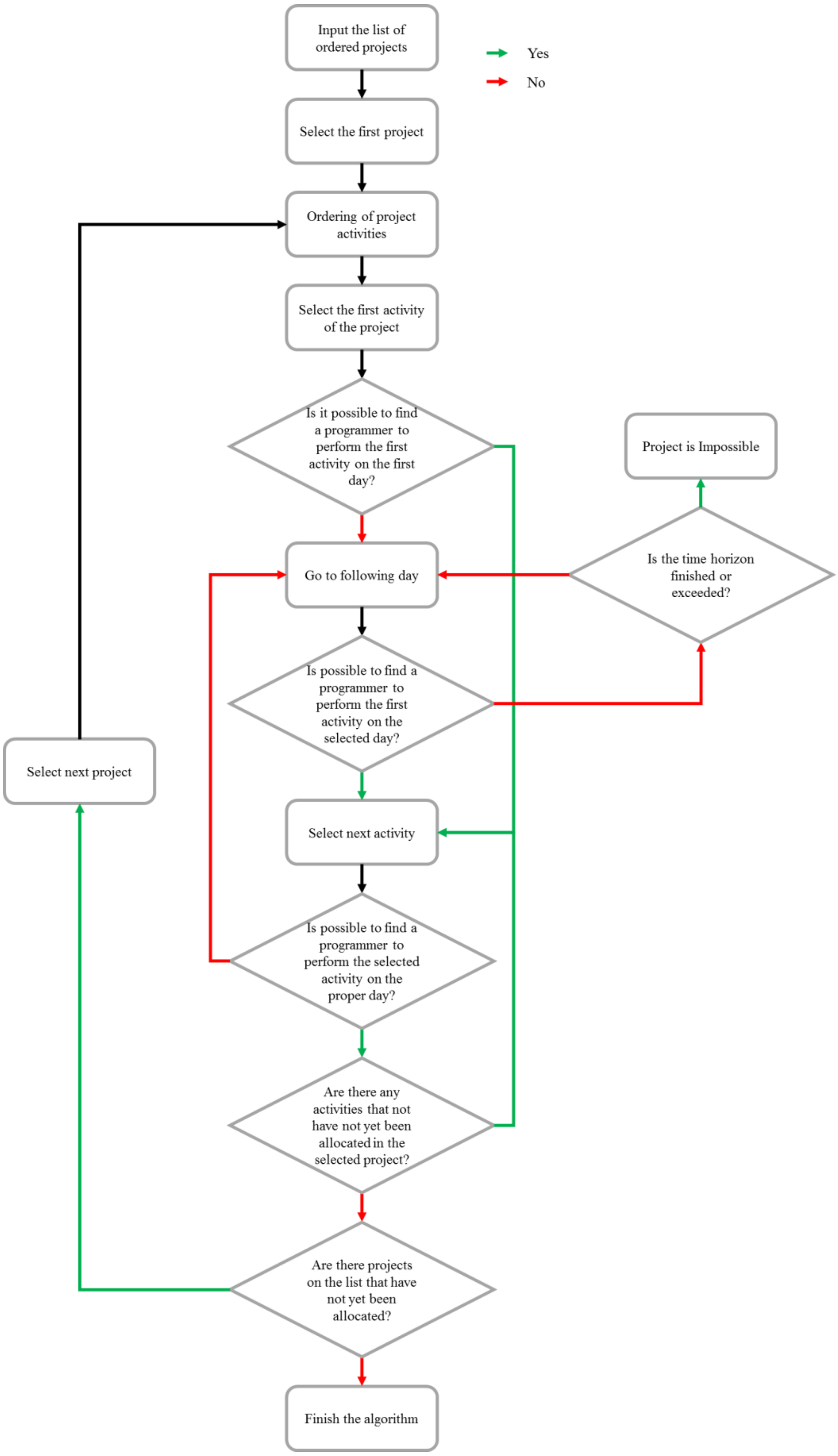
The allocation algorithm is repeated until all the activities of the selected project are allocated in the proper days, following the Gantt chart of their project

Step 5 Update the programmers' schedule once all activities of the project are allocated.

Step 6 Select the next project in the list and return to step 2. Repeats the algorithm until all the projects are allocated.

Figure 10 is a flow chart depicting a visual representation of the algorithm.

Figure 10 Flow Chart for the heuristic algorithm


Input the list of ordered projects
Yes
No
Select the first project
Ordering of project activities
Select the first activity of the project
Is it possible to find a programmer to perform the first activity on the first day?
Project is Impossible
Go to following day
Is the time horizon finished or exceeded?
Is possible to find a programmer to perform the first activity on the selected day?
Select next project
Select next activity
Is possible to find a programmer to perform the selected activity on the proper day?
Are there any activities that not have not yet been allocated in the selected project?
Are there projects on the list that have not yet been allocated?
Finish the algorithm

Source: Author

The proposed algorithm was developed in the free software GNU Octave, under the terms of the GNU General Public License. The entire code for the algorithm is presented in **Appendix B.** GNU Octave is a software featuring a high-level programming language, primarily intended for numerical computations. Octave is of great use for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with MATLAB.

4.3.3 Example

The heuristic method suggested is applied to Problem 1, previously described in section 3.2.1, in order to clarify the steps. The dispatched order used is the importance order.

Step 1 The first project to be allocated is Project B, the one with the higher importance between the two projects of the problem.

Step 2 All the programmers of the problem are capable of performing at least one of the activities of project B, so all the programmers are contained in the set of possible programmers.

Step 3 Project B has three activities, A4, A5 and A6. Activities A4 and A6 have two programmers capable of performing each one of them. Activity A5 has three programmers capable of performing it. The priority order for the allocation of the activities in Project B is A4, A6 and, lastly, A5.

Step 4 The first activity to be allocated is A4. A4 starts on the 3rd day of project B according to the Gantt Chart, so the first feasible day it can be allocated is day 3. The programmers capable of performing A4 are P2 and P3. On day 3, P2 is not available, but A4 can be allocated to P3. The second activity to be allocated is A6. A6 can be performed by programmers P1 and P4. Once A4 was allocated on day 3, A6 must necessarily start on day 2, according to the Gantt chart. Neither P1 nor P4 are available between day 2 and day 4, so it is not possible to allocate A6 on day 2. The allocation of activity A4 must, consequently, be revisited.

Table 24 Heuristic Method Step by Step (1)

	1	2	3	4	5	6	7	8	9	10
P1				**B1**	**B1**					
P2		**B2**	**B2**							
P3			**A4**	**A4**	**A4**	**B3**				
P4			**B4**	**B4**				**B5**		
		A6	**A6**	**A6**						

In the next two days, day 4 and day 5, A4 can be allocated, but A6 cannot. On the next day, day 6, A4 can be allocated to P2. A6 can then be allocated to P4 on day 5. Once A4 and A6 are allocated, it is time to allocate A5. A5 has to be allocated 2 days before A4, so it must be allocated on day 4. Programmers P1, P3 and P4 can perform activity A4. Programmer P1 is not available on day 4 and day 5, so P3 should be checked. Programmer P3 is available between days 4 and 5.

Step 5 As all the activities of Project B are properly allocated, the schedule of the programmers can be updated.

Table 25 Heuristic Method Step by Step (2)

	1	2	3	4	5	6	7	8	9	10
P1				**B1**	**B1**					
P2		**B2**	**B2**			**A4**	**A4**	**A4**		
P3				**A5**	**A5**	**B3**				
P4			**B4**	**B4**	**A6**	**A6**	**A6**	**B5**		

Step 6 The next project of the list is Project A and it has three activities: A1, A2 and A3. All the programmers possess the skill to perform at least one of the activities in project A, so all programmers are in the set of possible programmers. Activities A1 and A3 can be carried out by two possible programmers and activity A2 can be done by three possible programmers. The allocation priority order of Project A activities is A1, A3 and A2. Activity A1 starts on the first day of Project A, thus the first day to be analyzed is day 1. The possible programmers for activity A1 are P2 and P3. Programmer P2 is not available, thus the first attempt to allocate A1 is to P3 on day 1. Once A1 is allocated on day 1, A3 must be allocated on day 3, to remain in accordance with the Gantt Chart. The possible programmers for A3, P1 and P4, are not available between day 3 and day 5.

Table 26 Heuristic Method Step by Step (3)

	1	2	3	4	5	6	7	8	9	10
P1				B1	B1					
P2		B2	B2			A4	A4	A4		
P3	A1	A1		A5	A5	B3				
P4			B4	B4	A6	A6	A6	B5		
			A3	A3	A3					

Hence, the allocation of activity A1 must be shifted. On the next day, day 2, it is possible to allocate A1, but it is not yet possible allocate A3. On day 3, neither P2 nor P3 are available to perform activity A1. On day 4, P2 can perform A1 and P1 can perform A3. Activity A2 has to be performed during days 5, 6 and 7, and programmers P1, P3 and P4 are not available on those days.

Table 27 Heuristic Method Step by Step (4)

	1	2	3	4	5	6	7	8	9	10	11
P1				B1	B1	A3	A3	A3			
P2		B2	B2	A1	A1	A4	A4	A4			
P3				A5	A5	B3					
P4			B4	B4	A6	A6	A6	B5			
					A2	A2	A2				

In the next couple of days, day 5 and day 6, P2 and P3 cannot perform A1. The next feasible day for Al is day 7 for programmer P3. Activity A3 can be performed by P1. Activity A2 has to be performed between days 8 and 10, but P1, P3 and P4 are not available.

Table 28 Heuristic Method Step by Step (5)

	1	2	3	4	5	6	7	8	9	10	11
P1				B1	B1				A3	A3	A3
P2		B2	B2			A4	A4	A4			
P3				A5	A5	B3	A1	A1			
P4			B4	B4	A6	A6	A6	B5			
								A2	A2	A2	

Activity A1 can be allocated on the next day, day 8, to programmer P3. Activity A3 can be performed by programmer P1 on day 10. Activity A2 can be performed by programmer P4 on day 9.

Table 29 Heuristic Method Step by Step (6)

	1	2	3	4	5	6	7	8	9	10	11	12
P1				**B1**						**A3**		
P2		**B2**				**A4**						
P3				**A5**		**B3**		**A1**				
P4			**B4**		**A6**			**B5**	**A2**			

All of the activities of project A are allocated, the schedule of the programmers can then be updated. All the projects of the problem are already allocated, thus the heuristic is finished.

Table 30 Heuristic Method Step by Step (7)

	1	2	3	4	5	6	7	8	9	10	11	12
P1				**B1**						**A3**		
P2		**B2**				**A4**						
P3				**A5**		**B3**		**A1**				
P4			**B4**		**A6**			**B5**	**A2**			

Problem 1 solved via the heuristic returned an objective function value of 28, a value only 3.7% higher than the optimal result of the linear programming. The heuristic method cannot guarantee the optimal result of the final solution, but it can find a good feasible solution with a reasonable computational cost. The final solution of Problem 1 helps to validate the heuristic. The comparison between the linear programming and the heuristic methods for problems 2 and 3 is showed in **Table 31**.

Table 31 Comparison between MILP and Heuristic Method

	Problem	1	2	3	4	5	Real
Dimensions	**Programmers**	4	110	110	648	1155	1916
	Projects	2	3	3	32	10	32
	Activities	6	41	41	423	132	423
	Lacks of Availability	5	0	40	174	322	576
	Events	11	41	81	455	454	999
	x_{ijk}	484	184910	721710	1,34E+08	2,38E+08	1,91E+09
MILP	**CPU Time**	1,69 s	8,25 s	26,13 s	-	-	-
	Objective Function	27	213	213	-	-	-
Heuristics	**CPU Time**	3,78 s	19,37 s	21,48 s	1m32s	24,28 s	28,59 s
	Objective Function	28	213	213	8424	826	1925

- Exceeds the processing capacity

The difference between Problem 2 and Problem 3 is the inclusion of the lack of availabilities of the 110 selected programmers. This addition resulted in a 217% increase in the CPU Time for the MILP resolution and a 10.7 % increase in the CPU Time for the Heuristic. The heuristic was capable of finding the optimal solution in both problems.

For problems 4, 5 and the real one, the dimension of the quantity of variables x_{ijk} is significantly larger than in the previous problem. The CPLEX software is not capable of creating and processing all of these variables.

The particularly high CPU time for the heuristic in Problem 4 can be explained by the coefficient between the quantity of programmers and activities. The coefficient for Problem 4 is 1.53 programmer/activity, while the coefficient for Problem 5 and the Real Problem is, respectively, 9.09 and 4.54 programmer/activity. This means that the heuristic algorithm has a smaller quantity of options for the allocation of activities and has to search more days in order to find a proper allocation for all activities.

A good heuristic algorithm, according to MARTÍ & REINELT (2011), should find a solution with reasonable computational effort. The heuristic presented a good result in terms of CPU time. In the next chapter, the comparison between the heuristic result for the real problem and the result of the bank model is going to be explored.

5. DISCUSSION OF RESULTS

In this chapter a comparison between the solution found by the proposed heuristic and the current solution used by the company is presented. Some improvement suggestions for the model are presented and other success factors for a good allocation solution in addition to the weighted completion time.

Some additional data about the projects and the results of the allocation is shown in **Table 32**. In the first three columns, the data inputs for the project are presented: project ID, importance and duration. In the next three, the data obtained from the heuristic is presented: the completion and starting times for each project, and the weight factor assigned to the project that will be used in the objective function. The heuristic was developed with priority rules WI and WSPT, with both of them presenting the same result. In the final three columns, the real allocation data is presented: the completion and starting times for each project, and the weight factor assigned to the project that will be used in the objective function.

Table 32 Comparison between the results of the heuristic and the real problem

Project Data			Heuristic			Real		
Project	**Importance (ω_l)**	**Duration**	**Completion time (CP_l)**	**Start time**	$\omega_l \cdot CP_l$	**Completion time (CP_l)**	**Start time**	$\omega_l \cdot CP_l$
#P01	7	11	11	0	77	11	0	77
#P02	10	11	11	0	110	11	0	110
#P03	2	13	13	0	26	20	7	40
#P04	1	21	21	0	21	22	1	22
#P05	6	13	25	12	150	16	3	96
#P06	6	15	15	0	90	23	8	138
#P07	6	13	13	0	78	13	0	78
#P08	8	9	9	0	72	9	0	72
#P09	4	7	7	0	28	-	-	-
#P10	8	9	9	0	72	9	0	72
#P11	3	19	19	0	57	20	1	60
#P12	7	13	13	0	91	13	0	91
#P13	3	11	21	10	63	22	11	66
#P14	1	5	5	0	5	5	0	5
#P15	4	5	5	0	20	5	0	20
#P16	3	7	7	0	21	7	0	21
#P17	9	7	7	0	63	7	0	63
#P18	5	5	5	0	25	5	0	25
#P19	5	5	5	0	25	5	0	25
#P20	6	5	5	0	30	5	0	30
#P21	7	11	11	0	77	24	13	168
#P22	2	9	9	0	18	9	0	18
#P23	8	19	29	10	232	31	12	248
#P24	6	9	9	0	54	9	0	54
#P25	7	7	7	0	49	7	0	49
#P26	3	13	13	0	39	13	0	39
#P27	1	2	9	7	9*	-	-	-
#P28	5	1	1	0	5	1	0	5
#P29	8	7	7	0	56	7	0	56
#P30	8	9	9	0	72	9	0	72
#P31	7	20	20	0	140	21	1	147
#P32	10	5	5	0	50	5	0	50

* Adapted project allocation

The result obtained by the heuristic method is 4.6% better than the current schedule and allocation plan of the software factory, as shown in **Table 33**.

Table 33 Comparison between the heuristic and the real problem objective functions

Weighted Completion Time	**Heuristic**	**Real**
$\sum \omega_l \cdot CP_l$	1925	2017

The solution used by the bank was generated by a VBA Excel tool. The method used by the tool is a greedy algorithm that searches for the local optimum for each one of the projects. The tool was not able to find a feasible allocation for projects #P09 and #P27. Its CPU Time was 3m34s. Six times the CPU time of the heuristics. The heuristic method identified that project #P27 has one activity with no feasible allocation. Two activities that should be done in parallel need a specific skill that only one programmer has. The heuristic identified the activity as impossible, removed it and allocated all the others activities of the project normally. The idea of the allocation for the adapted project (without the impossible activity) is to give a better idea of the general allocation. A suggestion to solve the impossibility of the activity is to review the Gantt Chart and check if it is really mandatory for the two conflicting activities to be done in parallel. If that is not really the case, then the activity is not impossible anymore. If that is, in fact, the case, the software factory should consider the possibility of hiring outside services in order to perform the impossible activity.

Other factors used to measure the success of the heuristic are the quantity of projects not allocated, the mean starting time and the completion time of the project portfolio. **Table 34** shows the value of these success factors, and it can be observed that for every case the heuristic method provides the better solution.

Table 34 Other heuristic success factors

Success Factors	**Heuristic**	**Real**
Quantity of Impossible Projects	1	2
Mean Time to Start a Project	1,2	1,9
Completion Time of the Project Cycle	29	31

Other priority rules can be tested in the model in order to check if a better solution is possible to be found with the proposed heuristic.

5.1 Sensibility Analysis of the Resource Constraints

Some modifications in the model can also be tested, such as an improved order of the programmers. Two different methods for ordering the programmers can be proposed:

- Rarity skill index – the programmers with rare skills are the last to be allocated, in order to keep them available for any uncommon activities. An effort to identify what the top rare skills are could also help pointing out the skills that can be covered in training.

- Availability rate – the programmers with more idle time are allocated first. **Table 35** shows the programmers occupation rate after the heuristic allocation. The occupation rate is given by dividing the number of days the programmer was occupied in the cycle by the completion time of the project cycle, in this case 29 days. 1130 programmers were not allocated to any activity in the cycle, while only 1% of the programmers had more than 69% of their days occupied. Sorting the programmers by their availability rate could improve resource levelling, and also improve the diversification of programmers.

Table 35 Programmers Occupation Rate after Heuristic Result

Occupied days	Occupation rate	Number of programmers	Percentage of programmers
0	0%	1130	58,98%
1	3%	272	14,20%
2	7%	160	8,35%
3	10%	13	0,68%
4	14%	19	0,99%
5	17%	13	0,68%
6	21%	4	0,21%
7	24%	40	2,09%
8	28%	9	0,47%
9	31%	40	2,09%
10	34%	28	1,46%
11	38%	48	2,51%
12	41%	20	1,04%
13	45%	38	1,98%
14	48%	1	0,05%
15	52%	7	0,37%
16	55%	8	0,42%
17	59%	14	0,73%
18	62%	13	0,68%
19	66%	17	0,89%
20	69%	7	0,37%
21	72%	7	0,37%
22	76%	2	0,10%
24	83%	2	0,10%
26	90%	2	0,10%
27	93%	1	0,05%
28	97%	1	0,05%

After the result of the first heuristic allocation, one of the outputs is the new schedule of the programmers. One possible way for sorting the programmers in order to improve the availability rate is to sort them by occupation rate after a first heuristic allocation. The idle programmers in the first allocation result are going to be searched first in the algorithm and the programmers with high occupation rate will be the last. After the programmers are sorted, the heuristic method is applied again.

The second allocation by heuristic method presented an occupation rate that was more balanced among the pogrammers. The completion time of the project cycle is also 29 days. As **Table 36** shows, after the second allocation only 907 programmers were completely idle, 19.7% less than after the first allocation. Since the processing time of the heuristic is considered short (30 seconds, approximately), carrying out the heuristic twice does not incur a great cost if a more balanced occupation is considered as an interesting secondary objective. In terms of the objective function, the result was only slightly better, from 1925 to 1918, an improvement of less than 1%.

Table 36 Programmers Occupation Rate – Second Allocation

Occupied days	Occupation rate	Number of programmers	Percentage of programmers
0	0%	907	47,34%
1	3%	348	18,16%
2	7%	349	18,22%
3	10%	16	0,84%
4	14%	3	0,16%
5	17%	37	1,93%
7	24%	32	1,67%
9	31%	53	2,77%
10	34%	1	0,05%
11	38%	45	2,35%
12	41%	5	0,26%
13	45%	30	1,57%
14	48%	10	0,52%
15	52%	6	0,31%
16	55%	5	0,26%
17	59%	18	0,94%
18	62%	14	0,73%
19	66%	17	0,89%
20	69%	6	0,31%
21	72%	4	0,21%
22	76%	5	0,26%
24	83%	2	0,10%
26	90%	3	0,16%

The possibility of reducing the number of programmers available for the project cycle should be carefully analysed. Problem 4 presented some interesting results for the sensibility analyses of resource constraints. This problem is a small version of the real problem, where only programmers with a unique range of skills were kept; in other words, there were not two programmers with the same group of skills. The choice for removing only programmers with a duplicate group of skills was a way of removing some programmers, but also keeping all the skills present in the programmers' skill range.

The results of the success factors for Problem 4 are shown in **Table 37**. The projects take longer to start and the completion time of the portfolio is significantly higher than in the original problem. The removal of 1268 duplicate programmers considerably worsened the allocation scenario. This result is particularly interesting once it is observed in **Table 35** that 1130 programmers are completely idle after the heuristic allocation of the complete problem. It is suggested that the bank does a comprehensive study of the skill profiles of its programmers in order to find what are the more valuable profiles for projects and what are those that have a higher probability of being idle. This study could help the human resources department to provide training for those programmers that have a high probability of being idle and also to help direct possible layoffs as a cost cutting measure.

Table 37 Success Factors – Reduced Number of Resources

Success Factors	**Heuristic Problem 4**
Weighted Completion Time	8424
Quantity of Impossible Projects	1
Mean Time to Start a Project	39,1
Completion Time of the Project Cycle	139

6. CONCLUSIONS

This work developed a MILP model and a heuristic method in order to address a real allocation problem of the software factory of an important Brazilian bank. In view of the results presented in the previous chapters, this work was able to successfully create a valid model to represent the problem of the company and, through this model, find a solution to the problem that satisfies the boundary conditions proposed. This model, which differs in its specific characteristics to those found in other models from literature, can be considered a valuable solution for practical problems.

The method developed was capable of finding a better solution than the one currently found in the software factor of an important Brazilian bank. The reduction in 4,6% in the weighted completion time of the projects, driven mainly by an improvement in resource allocation, will help the software factory improve its internal service level agreement (SLA) with the departments in the bank. At the same time, this improved allocation can reduce the cycle time of projects and decrease the line of non-prioritized projects.

In addition to the contributions to the company, this work contributes to the literature on the development of a mixed integer linear programming scheduling and a constructive heuristic for resource-constrained multi-project scheduling with precedence relations, resource availability constraints and resource eligibility restrictions.

In order delve deeper into the topic of the study conducted in this project, it would be interesting to collect other examples of problems characterized by resource-constrained multi-project scheduling with precedence relations, resource availability constraints and resource eligibility restrictions problems.

Another variation of this study could be to relax the precedence constraints and add in the model the possibility of resumable cases. This implies that if the activity cannot be finished before the lack of availability of the programmers, it can continue to be performed after the programmer is available again. It is reasonable to assume that a programmer can have a free day or a training day in the middle of the activity and can continue to perform it after their free day or once the training is complete.

Regarding the heuristic improvements, a very interesting continuation of the model would be to test different list scheduling possibilities not only for the projects, but also for the programmers and activities. Combining the different methods for sorting programmers, activities and projects could conceivably lead to a better solution than the one developed in this work.

REFERENCES

ARENALES, M.; ARMENTANO, V.; MORABITO, R.; YANESSE,H. **Pesquisa Operacional**: Para Cursos de Engenharia. 2ª ed. Elsevier, 2015.

BROWNING, T. R.; YASSINE, A. **Resource-constrained multi-project scheduling:** Priority rule performance revisited. International Journal of Production Economics, 2010. v. 2.

FEBRABAN **Bank Technology Survey**, 23º ed. São Paulo: 2014.

FEBRABAN **Bank Technology Survey**, 24º ed. São Paulo: 2015.

FERNANDES, A. A.; TEIXEIRA D. S. **Fábrica de Software**: Implantação e Gestão de Operações. São Paulo: Atlas, 2004.

GOMES, H. C.; NEVES, F. A.; SOUZA, M. J. F. **Multi-objective metaheuristic algorithms for the resource-constrained project scheduling problem with precedence relations**. Computers & Operations Research, 2014. v. 44.

KERZNER, H. **Project management**: A Systems Approach to Planning, Scheduling, and Controlling. 10º ed. New Jersey: John Wiley & Sons, 2009.

KRIVONOZHKO, V..; FØRSUND, F.; LYCHEV, A. **Measurement of returns to scale using non-radial DEA models**. European Journal of Operational Research, Elsevier, 2014. V. 232

LEE, C.-Y. **Machine scheduling with an availability constraint**, Journal of Global Optimization, 1996. V. 9

LI, C. L. **Scheduling unit-length jobs with machine eligibility restrictions**. European Journal of Operational Research. 2006. v. 174.

MA, Y.; CHU, C.; ZUO, C**. A survey of scheduling with deterministic machine availability constraints.** Computers & Industrial Engineering. Elsevier, 2010. v. 58.

MARTÍ, R.; REINELT, G. **The Linear Ordering Problem, Exact and Heuristic Methods in Combinatorial Optimization.** Berlin: Heidelberg , 2011

MEDEIROS, V. N.; ANDRADE C.A.R.; ALMEIDA, E. S.; ALCUQUERQUE, J.; MEIRA S. **Construindo uma fábrica de Software:** da Concepção às Lições Aprendidas. Anais da XXX

Conferencia Latinoamericana de Informatica (CLEI2004), Arequipa Peru.. Available on: http://clei.org/clei2004/HTML/PDFS/123.PDF

MULLER, F. M.; DIAS, O. B. ARAÚJO, O. C.B. **Algoritmo para o problema de sequenciamento em máquinas paralelas não-relacionadas**. Revista Produção, 2002. v.12.

PAI, S. K.; VERGUESE, P; RAI, S. **Application of Line of Balance Scheduling Technique (LOBST) for a Real estate sector**; International Journal of Science, Engineering and Technology Research (IJSETR), 2013. v. 2.

PANWALKAR S.S.; ISKANDER W. **A Survey of Scheduling Rules**; Operations Research, 1977

PINEDO, M. L. **Scheduling**: Theory, Algorithms, and Systems. 3º ed.; Springer, 2008.

POTTS C.; CHEN, B.; WOEGINGER G. **A Review of Machine Scheduling**: Complexity, Algorithms and Approximability - Handbook of Combinatorial Optimization. Kluwer, 1998.

PMBOK GUIDE. **A Guide to the Project Management Body of Knowledge**. Project Management Institute, 2000.

QUADT, D.; H. KUHN. **Capacitated lot-sizing with extensions**: a review. 4OR - Quarterly Journal of Operations Research, 2009. v. 6

RUSS J. M., DRAGAN Z. M. **Project Management ToolBox**: Tools and Techniques for the Practicing Project Manager. 2º ed. New Jersey: John Wiley & Sons, 2016.

SHAKHLEVICH, N. **Scheduling**: Models and Algorithms, School of Computing, University of Leeds. 2005

WEI, C. C.; LIU. P. H.; TSAI, Y. C**. Resource-constrained project management using enhanced theory of constraint**. International Journal of Project Management. Elsevier, 2002. v. 20.

WINSTON, L. W. **Operations Research:** Applications and Algorithms. 4º ed. Thomson Learning, 2004.

XHAFA, F.; ABRAHAM, A. **Metaheuristics for Scheduling in Industrial and Manufacturing Applications.** Springer. 2008

YAMASHITA, D. S. **Scatter search para programação de projetos com custo de disponibilidade de recursos sob incerteza**. Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação. 2003.

APPENDIX A

The code in OPL 12.6.0.0 Model for the mixed integer linear programming

```
/*********************************************
 * OPL 12.6.0.0 Model
 * Author: Christianne_Sepulveda
 * Creation Date: 29/08/2016 at 08:04:36
 *********************************************/


 int n_job = ...; // number of events
 int n_recur = ...; // number of programmers
 int H = ...;// a large number
 int n_proj = ...;// number of projects

  //************************** sets****************************************//

 range recur = 1..n_recur; // set of programmers
 range job = 0..n_job; // set of events
 range proj = 1..n_proj;//set of projects

 //************************** Parameters*******************************//

 float t_process [job] = ...;      // duration time of the event
 float job_recur [recur][job] = ...; //set of programmers capable of performing the event
 float import [proj] = ...; // importance of the project
 float preced [job][job] = ...; // Precedence relationship between activities of a same Project
 float e_block [job] = ...;        // set of lack of availability
 float c_block [job] = ...;        // completion time of the lack of availability
 float job_proj[proj][job]=...;// set of activities belonging to Project

//**************************   Variables    ********************//

 dvar int x[job][job][recur] in 0..1;// 1 if event I (indicated by the first "job" in this line of
code) precedes event j (represented by the second "job" in this line of code) for programmer k
(recur), 0 otherwise
 dvar int+ c[job];// completion time of activity job
 dvar int c_proj[proj];// completion time of Project proj

 //***************     Objective function**********************************//

// Minimize the weighted sum of the completion time for each project
minimize  sum(l in proj)import[l]*c_proj[l];


 //**************************      Restrictions **************************//
 subject to {

 //Restriction (1) Each event j has only one preceding activity, except activity 0//

 restriction1:
```

```
 forall (j in job: j!=0){
   sum (i in job, k in recur: job_recur[k][j]!=0 && job_recur[k][i]!=0) x[i][j][k]==1;}

  //Restriction (2) Each programmer k, if allocated in any event, has only one processing
sequence//

  restriction2:
  forall (k in recur){
  sum (j in job: j!=0 && job_recur[k][j]!=0) x[0][j][k]<=1; }

  //Restriction (3) Each activity j has only one immediate successor activity, except activity 0,
that sets the beginning and the end of all processing sequences for programmer k

  restriction3:
  forall (h in job, k in recur:h!=0 && job_recur[k][h]!=0){
  sum(i in job: i!=h && job_recur[k][i]!=0)x[i][h][k] - sum(j in job: j!=h &&
job_recur[k][j]!=0)x[h][j][k] == 0;

  }
  //Restriction (4) The completion time of event j is higher or equal to the completion time of
event i, plus the duration of activity j, when event i precedes event j in the same programmer
sequence k. (xijk=1)

  restriction4:

  forall (i in job, j in job: j!=0){

    c[j]>=c[i] + sum(k in recur: job_recur[k][j]!=0)x[i][j][k]*t_process[j] + H*(sum(k in recur:
job_recur[k][j]!=0)x[i][j][k]-1);

 }

 //Restriction (5) The completion time of a event i precedes the completion time of event j in a
given number of days if i and j belong to the same project

 restriction5:

 forall (i in job, j in job: preced[i][j]!=0){

 c[i]==c[j]+preced[i][j];

 }
 //Restriction (6) The completion time of a event is already given if it is a lack of availability

 restriction6:

 forall (i in job: e_block[i]!=0){

 c[i]==c_block[i];
 }
```

```
//restriction (7) The completion time of the Project is equal to the completion time of the last
activity to finish that belongs to the project

restriction7:

forall (l in proj, i in job: job_proj[l][i]!=0){

c_proj[l]>=c[i];
}
}
```

The input data for the preliminary test OPL 12.6.0.0 Model.

```
/*********************************************
 * OPL 12.6.0.0 Data
 * Author: Christianne_Sepulveda
 * Creation Date: 29/08/2016 at 08:04:36
 *********************************************/

n_job =11; // Number of activities
n_recur = 4; // Number of programmers
n_proj = 2;// Number of projects

               //A0,A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5// Events
t_process =   [0, 2, 3, 3, 3, 2, 3, 2, 2, 1, 2, 1]; // Duration of the events


               //A0,A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5// Events
job_recur =    [[1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0],  // P1
                 [1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0],  // P2
                 [1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0],  // P3
                 [1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1]]; // P4

     //A,B// Projects
import = [1,2]; // Importance of the projects


H = 10000; // Large Number

                //A0,A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5// Events
preced =        [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  // A0
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  // A1
                  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  // A2
                  [0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],  // A3
                  [0, 0, 0, 0, 0, 3, 1, 0, 0, 0, 0, 0],  // A4
```

```
                   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  // A5
                   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  // A6
                   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  // B1
                   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  // B2
                   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  // B3
                   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  // B4
                   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]; // B5

                  //A0,A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5// Events
e_block =       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1];

                  //A0,A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5// Events
c_block =      [0, 0, 0, 0, 0, 0, 0, 5, 3, 6, 4, 8];

                  //A0,A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5// Events
job_proj =        [[0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0], // Proj A
                   [0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0]]; //Proj B
```

APPENDIX B

The present section presents the code developed in the MATLAB language, using the OCTAVE software for the constructive heuristic proposed. All the lines starting with “%” are comments explaining and clarifying parts of the code.

```
%% Clean the work area
close all
clear all
clc

t0 = time();

%% #0 Step: Control Parameters

% These parameters indicate whether one of the main steps is executed or
% not

flag_read_data = 1;
flag_process = 1;
flag_pos_proc = 1;

%% #1 Step: Reading Data
if flag_read_data

   file1 = 'programmers_agenda.xls';
   file2 = 'ativity_duration.xlsx';
   file3 = 'programmers_activities.xlsx';
   file4 = 'importance.xlsx';
   file5 = 'project_ativity.xlsx';

   [ag_func, txt1, raw1]=xlsread(file1);
   [d_atv, txt2, raw2]=xlsread(file2);
   [func_atv, txt3, raw3]=xlsread(file3);
```

```
[imp, txt4, raw4]=xlsread(file4);
[proj_atv, txt5, raw5]=xlsread(file5);

% Sort data according to the list scheduling
[Simp, impID] = sort(imp(:,2));
imp = imp(impID,:);
impAux = imp(:,3);
proj_atv = proj_atv(impID,:);
imp(impAux==0,:) = [ ];
proj_atv(impAux==0,:) = [ ];
impID(impAux==0) = [ ];

% Clean data not useful
ag_func(:,1) = [ ];
func_atv(:,1) = [ ];

% Gives 1 to free days and 0 to lack of availability
ag_func(ag_func==1)=2;
ag_func(ag_func==0)=1;
ag_func(ag_func==2)=0;

% Create the schedule for the following days (from the third month onwards)
ag_func = [ag_func ones(length(ag_func(:,1)),100)];

% Back up the original schedule
org_ag_func = ag_func;

% Starts the matrix for impossible activities
impossible{length(proj_atv(:,1)),3} = [];
for i=1:length(impossible(:,1))
   impossible{i,1} = 0;
end
```

```
   % Starts the matrix for the activities schedule
   for i=1:length(d_atv(:,1))
       ag_atv(i,:) = [ones(1,d_atv(i)) zeros(1,length(ag_func(1,:))-d_atv(i))];
   end

 % Save the data that has been processed up to this point
save('dados.mat','proj_atv','ag_atv','impossible','ag_func','org_ag_func','func_atv','d_atv','imp','
impID');

end
% Shows the current time after the first step

t1 = time;

%% #2 Step: Data processing
if flag_process

   load('dados.mat');
% for all projects i
   for i=1:length(proj_atv(:,1))
       ret_atv = 1; %indicates if all activities were correctly read
       while ret_atv
           ret_atv = 0;
           aux_ag_func = ag_func;
           atv_per_proj = find(proj_atv(i,:)==1); % Selects only the activities of Project i
           r_func_atv = func_atv(:,atv_per_proj); %Selects only the programmers capable of
performing at least one of the activities of Project i
           sum_r_func_atv = sum(r_func_atv); %Sum of how many programmers are capable of
performing each activity
           [S_sum_r_func_atv, ID_sum_r_func_atv] = sort(sum_r_func_atv); %Sort the activities
by the quantity of programmers capable of performing them
           S_r_func_atv = r_func_atv(:,ID_sum_r_func_atv);
           S_atv_per_proj = atv_per_proj(:,ID_sum_r_func_atv);
```

```
      r_ag_atv = ag_atv(S_atv_per_proj,:); %Selects only the schedule of the selected
programmers
      max_days_per_proj = max(d_atv(S_atv_per_proj)); %Determines the duration of the
Project (the duration of the last activity to end minus the first day of the Project)
      shift_days = 0; %Starts the parameters that delays the Project to the next day
      aux_shift_proj = shift_days; % flag to determine if the initial day of the Project must be
delayed
      proj_in = 1; %flag to determine if all the activities of Project i are allocated
      while proj_in
        aux_ag_func = ag_func; %back up the schedule before the allocation of the activities
of Project i
        term_proj = 1; %flag that determines if a Project is finished or not, it is used to modify
the value of the flag proj_in
        atv_in = find(r_ag_atv(:,1)==1);
         atv_in(impossible{i,2}) = [ ]; % Do not try to allocate activities already declared
impossible
        r_d_atv = d_atv(S_atv_per_proj); %determines the duration of the activities of the
project
%          r_d_atv(impossible{i,2}) = [ ]; %Removes  the impossible activities
        for k=1:length(atv_in) % for all k in the activities of the project
          if find(impossible{i,2}==k) % checks if the activity is possible
            continue
          end
          func_atv_k = find(S_r_func_atv(:,atv_in(k))==1); % Selects only the
programmers capable of performing activity k
          for w=1:length(func_atv_k) % for all programmer w capable of performing
activity k
            if
sum(aux_ag_func(func_atv_k(w),1+shift_days:r_d_atv(k)+shift_days))==r_d_atv(k)
%Verifies if programmer w has all the days of the duration of activity k free

aux_ag_func(func_atv_k(w),1+shift_days:r_d_atv(k)+shift_days)=zeros(1,length(r_d_atv(k)))
; %If programmer w is available for activity k, update the programmer’s schedule
              break
            end
            if w==length(func_atv_k) %Verifies if no programmer was allocated to activity
k
              shift_days = shift_days+1; %If the activity was not correctly allocated in the
searched day, the algorithm tries to find an allocation for the following day
              if shift_days+max_days_per_proj>length(aux_ag_func(1,:)) %If all the days
in the schedule were already tried and it is still not possible to find a allocation for activity k, it
is declared impossible
                impossible{i,1} = 1; % Determines that the Project i is impossible with the
current Gantt chart and available resources
                impossible{i,2} = [impossible{i,2}; k]; % Identifies the ID of the
impossible activity
                impossible{i,3} = [impossible{i,3}; S_atv_per_proj(k)]; %Identifies the
global ID of the impossible activity
                ret_atv = 1; % Assumes the project does not have a solution with the
original data
              end
              break
            end
          end
          if shift_days ~=aux_shift_proj %Verifies if there was a change in the searched day
            aux_shift_proj = shift_days; %
            term_proj = 0; %Indicates that it was not possible to allocate the Project i or that
there was a change in the searched day
            break
          end
        end
        if term_proj %Verifies if Project i is finished
          ag_func = aux_ag_func; %If positive, update the original schedule of the
programmers
          proj_in = 0; %Indicates that the current Project is already allocated and it is
possible to start the algorithm for the Project i + 1
```

```
                proj_days(i,:) = [shift_days+max_days_per_proj;shift_days;max_days_per_proj];
%Stores the completion time of the Project i, the start day of the Project and the duration of the
project
            elseif impossible{i,1}==1 %Verifies if the project has an impossible activity
                proj_in = 0; %In case there is an impossible activity, this indicates that Project i
has to be allocated without the impossible activity
                impossible{i,1} = 0; % after the removal of the impossible activity, the Project is
deemed feasible again
            end
        end
    end
end

% Indicates if there was any impossible activity in Project i
for i=1:length(impossible(:,1))
    if isempty(impossible{i,2})==0
        impossible{i,1} = 1;
    end
end

%save the results after data processing
 save('results.mat','org_ag_func','ag_func','impossible','proj_atv','imp','proj_days','impID');

end

% Gives the current time after data processing
t2 = time();

%% #3 Step: Final adjustments after processing
if flag_pos_proc

    load('results.mat')

    mod_ag_func = -(ag_func - org_ag_func);
```

```
%    arr_mod_ag_func = [ ];
%
%    for i=1:length(mod_ag_func(:,1))
%        for j=1:40
%            arr_mod_ag_func = [arr_mod_ag_func; i j mod_ag_func(i,j)];
%        end
%    end

    count_ag_func = sum(mod_ag_func')';

% Calculates the objective function

    obj_func = sum(imp(:,1).*proj_days(:,1));


save('pos.mat','org_ag_func','ag_func','impossible','proj_atv','mod_ag_func','count_ag_func','i
mp','proj_days','obj_func','impID')

end

% Gives the current time after the final adjustments
t3 = time();
```