Flexible Jobshop Scheduling Problem
with Resource Recovery Constraints

Tesis presentada por
Jobish Vallikavungal Devassia

como requisito parcial para obtener el grado de
Doctor en Ingeniería
con Especialidad en Ingeniería de Sistemas

julio 2017
Flexible Jobshop Scheduling Problem with Resource Recovery Constraints

Tesis presentada por
JOBISH VALLIKAVUNGAL DEVASSIA

Como requisito parcial para obtener el grado de
DOCTOR EN INGENIERÍA
con Especialidad en Ingeniería de Sistemas

julio 2017
Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica
Subdirección de Estudios de Posgrado

Los miembros del Comité de Tesis recomendamos que la Tesis “Flexible Job-shop Scheduling Problem with Resource Recovery Constraints”, realizada por el alumno Jobish Vallikavungal Devassia, con número de matrícula 1757843, sea aceptada para su defensa como opción al grado de Doctor en Ingeniería con Especialidad en Ingeniería de Sistemas.

El Comité de Tesis

Dra. María Angélica Salazar Aguilar
Directora

Dr. Vincent André Lionel Boyer
Co-Director

Jania Astrid Saucedo
Dra. Jania Astrid Saucedo Martínez
Revisor

Dr. Francisco Román Angel Bello Acosta
Revisor

Dra. Iris Abril Martínez Salazar
Revisor

Vo. Bo.

Dr. Simón Martínez Martínez
Subdirector de Estudios de Posgrado

San Nicolás de los Garza, Nuevo León, julio 2017
To my parents Devasiya and Annamma

to my affectionate brother Aneesh and lovely sister Shancy

to all my relatives

and to all those friends who have been with me during the storm and calm.
## CONTENTS

Acknowledgements ........................................... x

Summary .......................................................... xii

Resumen .......................................................... xv

1 Introduction ...................................................... 1

1.1 Motivation .................................................. 2

1.2 Problem statement ....................................... 5

1.2.1 Flexible Job-shop Scheduling Problem with Resource Constraints ........................................... 6

1.2.2 Scheduling problem in a brewing process ............ 7

1.3 Relevance .................................................. 8

1.4 Objectives .................................................. 10

1.4.1 Particular Objectives ................................ 11

1.5 Organization ............................................... 11

2 Literature review ........................................... 12
2.1 Resources in the Flexible Job-shop Scheduling Problem .......... 13
2.2 Resources in Other Scheduling Problems ......................... 16
2.3 Batch scheduling problems in parallel machines ............... 23

3 Scheduling problem with renewable resources ................... 28

3.1 The classical flexible job-shop scheduling problem .......... 29
  3.1.1 Example of the classical FJSP .......................... 29

3.2 The flexible job-shop scheduling problem with resource recovery con-
  straints (FRRC) .............................................. 31
  3.2.1 Example of the FRRC .................................... 32

3.3 Mathematical formulation ........................................ 36

4 Solution procedure for the FRRC .................................. 39

4.1 Variable Neighborhood Search ................................... 39
  4.1.1 Basic Variable Neighborhood Search ........................ 40
  4.1.2 General Variable Neighborhood Search ..................... 42

4.2 Solution procedure for FRRC based on GVNS .................... 43
  4.2.1 Solution representation ..................................... 43

4.3 Neighborhood structures of the GVNS ........................... 46
  4.3.1 Neighborhood structures in the shaking phase .............. 47
  4.3.2 Neighborhood structures for the local search ............... 50

4.4 Computational Results ........................................... 52
5 Case study - Scheduling problem in a brewery 57

5.1 Problem description - the brewing problem .................. 58

5.2 Solution procedure for the brewing problem .................. 60

5.2.1 Construction phase .................................. 61

5.2.2 Improvement phase .................................. 61

5.3 Experimental results ...................................... 62

5.4 Conclusions .............................................. 65

6 Conclusions ................................................. 66

6.1 Contributions and Future Work ................................ 66
## List of Figures

1.1 The breweries in Mexico ........................................ 3
1.2 Trade Balance of the Mexican Beverage Industry ............ 5
1.3 Production line configuration in a brewery .................... 6

3.1 A solution representation of a classical FJSP based on the instance described in Table 3.1 .................................. 31
3.2 A solution representation of a $2 \times 3$ FRRC based on Table 3.2 .......... 34

4.1 Solution representation of Figure 3.2 .......................... 44
4.2 Critical path of the solution representation shown in Figure 3.2 .... 45
4.3 $N_1$: Shifting a job in the critical path ........................ 47
4.4 $N_2$: Swapping two jobs in the critical path .................... 48
4.5 $N_3$: Swapping two jobs with machine reassignment ............... 49
4.6 $\overline{N}_1$: Reassigning the machines in the critical path .......... 50

5.1 General Brewing Process ........................................... 58
5.2 Makespan comparison with real data: company solution vs GRASP ........................................................................ 63
List of Tables

2.1 Literature on Scheduling Problems with Resource Constraints . . . . . . . 17
2.2 Batch production problems on parallel machines . . . . . . . . . . . . . 24
3.1 The $2 \times 3$ FJSP data . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
3.2 A $2 \times 3 \times 2$ FRRC data . . . . . . . . . . . . . . . . . . . . . . . . . . 33
4.1 Description of the tested instances . . . . . . . . . . . . . . . . . . . . . . . 53
4.2 CPLEX results using the proposed mathematical formulation . . . . 54
4.3 GVNS results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
5.1 Computational Experiments with Real Data . . . . . . . . . . . . . . . . . . 64
5.2 Average results of instances with varying demands . . . . . . . . . . . . 64
I am grateful to my dissertation supervisors, Prof. María Angélica Salazar Aguilar and Prof. Vincent André Lionel Boyer, for their insight, guidance, and encouragement to make this work possible, and for the friendship, we made during this time. Thanks to the members of my dissertation committee, for their contributions, comments, and advice, that significantly improved the quality of this research. I am very grateful to all those who have, in one way or another, contributed to this research. I thank each of the faculty members and fellow graduate students of the Graduate Program in Systems Engineering at Universidad Autónoma de Nuevo León, for all those unforgettable moments we have had.

I remember all of my friends who welcomed and helped me to have a wonderful start in Monterrey, both in academic and social life. Arthur, César Arturo, Pedro Inés, Olga Lineth, Mayela Estefania, Cristian Hernández, and Naohito Yamada are some of them. I thank them for giving me memorable moments in my life and my stay here in Monterrey such an exceptional experience. I am deeply grateful to my precious treasure, my family, with those who I shared the most special and exciting moments. Above all, I thank God for giving me the willingness, skills, and an opportunity to achieve my goals.

Last but not least, my research work would not have been possible without a doctoral fellowship from Consejo Nacional De Ciencia Y Tecnología (CONACyT) with grant number 385986, Universidad Autónoma de Nuevo León (UANL), and from the Facultad de Ingeniería Mecánica y Eléctrica (FIME) at UANL. Addition-
ally, my participation in some national and international conferences was supported by FIME-UANL and OPTISAD. Once again, I express my sincere gratitude to all of them to achieve my doctoral degree.
SUMMARY

Jobish Vallikavungal Devassia.

Ph.D. candidate in Engineering with a Specialization in Systems Engineering.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Title of the study: FLEXIBLE JOBSHOP SCHEDULING PROBLEM WITH RESOURCE RECOVERY CONSTRAINTS.

Number of pages: 92.

OBJECTIVES AND METHODS OF STUDY: The general objective of this research is to study a scheduling problem found in a local brewery. The main problem can be seen as a parallel machine batch scheduling problem with sequence-dependent setup times, resource constraints, precedence relationships, and capacity constraints.

In the first part of this research, the problem is characterized as a Flexible Job-shop Scheduling Problem with Resource Recovery Constraints. A mixed integer linear formulation is proposed and a large set of instances adapted from the literature of the Flexible Job-shop Scheduling Problem is used to validate the model. A solution procedure based on a General Variable Neighborhood Search metaheuristic is proposed, the performance of the procedure is evaluated by using a set of instances adapted from the literature.
In the second part, the real problem is addressed. All the assumptions and constraints faced by the decision maker in the brewery are taken into account. Due to the complexity of the problem, no mathematical formulation is presented, instead, a solution method based on a Greedy Randomize Adaptive Search Procedure is proposed. Several real instances are solved by this algorithm and a comparison is carried out between the solutions reported by our GRASP and the ones found through the procedure followed by the decision maker. The computational results reveal the efficiency of our method, considering both the processing time and the completion time of the scheduling. Our algorithm requires less time to generate the production scheduling (few seconds) while the decision maker takes a full day to do it. Moreover, the completion time of the production scheduling generated by our algorithm is shorter than the one generated through the process followed by the decision maker. This time saving leads to an increase of the production capacity of the company.

Contributions: The main contributions of this thesis can be summarized as follows: i) the introduction of a variant of the Flexible Job-shop Scheduling Problem, named as the Flexible Job-shop Scheduling Problem with Resource Recovery Constraints (FRRC); ii) a mixed integer linear formulation and a General Variable Neighborhood Search for the FRRC; and iii) a case study for which a Greedy Randomize Adaptive Search Procedure has been proposed and tested on real and artificial instances.

Signature of the faculty advisers:

Dra. María Angélica Salazar Aguilar

Dr. Vincent André Lionel Boyer
**RESUMEN**

Jobish Vallikavungal Devassia.

Candidato para obtener el grado de Doctor en Ingeniería con especialidad en Ingeniería de Sistemas.

Universidad Autónoma de Nuevo León.

Facultad de Ingeniería Mecánica y Eléctrica.

Título: **Flexible Jobshop Scheduling Problem with Resource Recovery Constraints**.

Número de páginas: 92.

**Objetivo y metodología de estudio:** El objetivo general de esta investigación es estudiar un problema de programación de producción que se identificó en una cervecería local. Este problema puede ser visto como un problema de programación por lotes en máquinas paralelas con tiempos de configuración dependientes de la secuencia, restricciones de recursos, relaciones de precedencia y restricciones de capacidad.

En la primera parte de esta investigación, el problema se caracteriza como un problema de programación flexible (tipo job-shop) con restricciones de recuperación de recursos. Se propone una formulación lineal entera mixta y se utiliza un gran conjunto de instancias adaptadas de la literatura del Flexible Job-shop Scheduling...
Resumen

Problem para validar el modelo. Se propone un procedimiento de solución basado en una metaheurística de tipo búsqueda de vecindarios variables (VNS), el desempeño del procedimiento se evalúa utilizando el conjunto de instancias adaptadas de la literatura. Los resultados computacionales muestran que el procedimiento propuesto es competitivo en cuanto a tiempo y calidad de las soluciones.

En la segunda parte, se aborda el problema real que se identificó en la empresa local. Se tienen en cuenta todos los supuestos y limitaciones a los que se enfrenta el tomador de decisiones en la cervecería. Debido a la complejidad del problema, no se presenta ninguna formulación matemática, sino que se propone un método de solución basado en un procedimiento voraz de búsqueda adaptativa aleatorizada (GRASP, por sus siglas en inglés). Varias instancias reales son resueltas por este algoritmo y se realiza una comparación entre las soluciones reportadas por el mismo y las encontradas a través del procedimiento seguido por el tomador de decisiones en la empresa. Los resultados computacionales revelan la eficiencia de nuestro método, considerando tanto el tiempo de procesamiento como el tiempo de finalización de la programación de la producción. Nótese que nuestro algoritmo requiere menos tiempo para generar la programación de producción (pocos segundos) mientras que el tomador de decisiones tarda un día entero en hacerlo. Por otra parte, el tiempo de finalización de la programación de producción generada por nuestro algoritmo es más corto que el generado a través del proceso seguido por el tomador de decisiones. Este ahorro de tiempo conduce a un aumento de la capacidad de producción de la empresa.

Contribuciones: Las principales aportaciones de esta tesis pueden resumirse de la siguiente manera: i) la introducción de una variante del Flexible Job-Shop Scheduling Problem, denominado como Flexible Job-shop scheduling problem with Resource Recovery Constraints (FRRC); ii) una formulación lineal entera mixta y una metaheurística de tipo búsqueda de vecindarios variables para el FRRC; y iii) el estudio del caso real para el cual se ha propuesto y probado un procedimiento voraz de
búsqueda adaptativa aleatorizada en instancias reales y artificiales.


Firma de directores: ____________________________
Dra. Maria Angélica Salazar Aguilar

______________________________
Dr. Vincent André Lionel Boyer
Decision-making in manufacturing and service industries includes planning and scheduling as important processes in production, transportation, and distribution, among others. The planning and scheduling role in a company is to allocate limited resources to the activities to be done. This allocation of resources has to be done in such a way that the company optimizes its objectives and achieves its goals. Resources may be machines in a workshop, runways at the airport, crews at a construction site, or processing units in a computing environment. Activities may be operations in a workshop, take-offs, and landings at an airport, stages in a construction project, or computer programs to be executed. Each activity may have a priority level, an earliest possible starting time and/or a due date. Objectives can take different forms, such as minimizing the time to complete all activities, minimizing the number of activities after the committed due dates, and so on.

In the last years, there has been an explosive growth in the development and implementation of computer-based scheduling systems in the industry. In the case of planning and scheduling in manufacturing, orders that are released in a manufacturing setting have to be translated into jobs with associated due dates. These jobs often have to be processed on the machines in a work center in a given order sequence. The processing of jobs may sometimes be delayed if certain machines are busy. Preemptions may occur when high priority jobs are released which have to
be processed at once. Unexpected events on the shop-floor, such as machine breakdowns or longer-than-expected processing times, also have to be taken into account, since they may have a major impact on the schedules. Developing, in such an environment, a detailed schedule of the tasks to be performed helps maintain efficiency and control operations.

In this research, two variants of scheduling problems are studied. Both can be seen like flexible job shop scheduling problems, however, different kind of constraints are taken into account. The first variant has been studied with theoretical purposes while the second one has been studied with practical reasons: solving a real case from a nationwide brewery. The main contributions of this work are the mathematical formulation for a flexible job-shop scheduling problem with resource recovery constraints and a metaheuristic to solve it. Additionally, a case study from a local brewing company has been carried out. The efficiency of the proposed solutions procedures has been shown on a large set of instances.

1.1 Motivation

In Mexico, beer production plays an important role given its significant contribution to the national economy. Mexican beer has high demand and constant growth, both in local and global markets. The annual report presented by National Brewers Association Craft Beer Market in Mexico 2015 illustrates the importance of beer in Mexico and its role in the Mexican economy. Moreover, brewing industry provides an outstanding number of direct and indirect jobs in Mexico. Figure 1.1 shows the major breweries in Mexico.

Official Mexican data shows that U.S. is the largest exporter of beer into Mexico, but it is mainly through large brands. According to the Mexican Brewers Association, craft beer market has expanded at 50% annual growth rate in the last ten years. For the fourth year running, Mexico was the world’s leading beer exporter
in 2014, $2.4 billion (17.9% of global beer exports). In 2013, Mexican beer exports reached a record of 2.2 billion dollars, a rise of 4.2% compared to 2012, and well ahead of both the Netherlands ($2.0 billion) and Belgium ($1.6 billion).

Mexico’s breweries provide about 80,000 jobs directly and a further 800,000 indirectly. Total beer sales each year are worth as much as 20 billion dollars, and due to the high demands, both domestic and foreign, these sales constantly rise each year.
Based on South American Business Information (SABI), Mexico was the 6th top beer producer in the world in 2014. In 2013, Mexico remained to be the top beer exporter for the fourth consecutive year. In that year, total beer exports reached 2.21 billion USD, representing 4.2% year-on-year rise and having 16.5% of the global beer market. For a better understanding, Chart 1.1 shows the income earned by beer exports, and Chart 1.2 illustrates a bar chart representation of beer exports of Mexico over some of the competitors in the global beer market.

<table>
<thead>
<tr>
<th>Country</th>
<th>Beer Exports (billion USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mexico</td>
<td>2.2</td>
</tr>
<tr>
<td>Netherlands</td>
<td>2.0</td>
</tr>
<tr>
<td>Belgium</td>
<td>1.6</td>
</tr>
<tr>
<td>Germany</td>
<td>1.4</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>1.0</td>
</tr>
<tr>
<td>United States</td>
<td>0.52</td>
</tr>
<tr>
<td>France</td>
<td>0.45</td>
</tr>
<tr>
<td>Denmark</td>
<td>0.32</td>
</tr>
<tr>
<td>Ireland</td>
<td>0.32</td>
</tr>
<tr>
<td>Portugal</td>
<td>0.27</td>
</tr>
</tbody>
</table>

Chart 1.2: Total Beer Exports of Countries (billion USD), 2013  
Source: Economista

Beer exports have been shown an increasing trend, from 1.790 million USD in 2009 to 2,211 million USD in 2013. Latest data show that by October 2014, beer exports totaled to 2,062.57 million USD, which was bigger than total beer exports in the same period in 2013 (1,872.72 million USD). An export and import balance of the Mexican beverage industry is shown in Figure 1.2. Due to this high exports of Mexican beer, and the influence of beer production in Mexican employment rate, the significant beer production can directly effect, the economy of Mexico.

One of the largest breweries in Mexico is located in Monterrey, Nuevo León. Hence, we established a relationship with the company and that allowed us to get insight into the brewing process. After multiple visits to the company, we observed that one of the hardest tasks in the company is the creation of the production schedule. That is the reason why we started this research work.
1.2 Problem statement

The planning and optimal production scheduling in the modern industry are very complex tasks. If we consider the brewing industry, the beer production needs exceptional attention because of a minor negligence leads to inferior products quality and significant productivity loss. It is practically impossible to make any small correction on the product since the errors may be realized only at the final stage of the production. Despite beer drawbacks, it has a significant role in social and economic growth in this twentieth century. Also, it is important to meet the customer orders by satisfying specific due dates. Thus, beer production needs an important consideration in production planning and handling. Figure 1.3 shows a typical production line and each stage of the production process in a brewery.

Inspired by this brewing process, we study two variants of scheduling problems. In the first one, we considered a flexible job-shop scheduling problem with a recovery time of resources. In the second one, we focused on the practical situation faced by the local company. Each variant is briefly described in the following subsections.
1.2.1 Flexible Job-shop Scheduling Problem with Resource Constraints

Due to the recovery nature of the yeast used in the brewing process, we identified the recovery nature of resources which have not been widely studied in the Flexible Job-shop Scheduling Problems (FJSP) literature. In the FJSP multiple operations must be assigned to machines in sequence (respecting the order of operations in each job) such that the total completion time is minimized. The most common characteristics of the FJSP are the following: the number of jobs and the number of machines are known in advance; each job is composed of a fixed sequence of operations; there is certain compatibility between operations and machines; the processing time of an operations depends on the machine assignment, and any machine can process only one operation at a time.

In addition to the classical FJSP characteristics, we take into account the renewability and availability of resources. The resources are available in batches and
the batch size and the recovery time of the resources depend on the resource type. Each operation may require a certain resource to be processed and during the production process the resources batch can be recovered once the entire batch of the resource is consumed. Then, the studied problem consists on assigning each operation to a compatible machine, by taking into account precedence relations between operations, resource requirements, compatibility and availability of resources and machines in order to minimize the makespan.

1.2.2 SCHEDULING PROBLEM IN A BREWING PROCESS

Scheduling of multiple products in parallel production lines, multiple stages of production, maintenance operations, and heterogeneous tanks makes the beer production scheduling a complex task. Moreover, the company produces various types of beers, and each beer type depends on the wort used.

There are three major stages in the brewing process: boiling, fermentation, and conditioning. During the boiling, all the ingredients are mixed and boiled to produce a particular wort which needs to be fermented. In the fermentation stage, multiple tanks can be used and each tank has a limited capacity, then we can start the boiling process of a wort, if and only if a fermentation tank is available. The conditioning stage is where the worts are conditioned and refined in order to obtain the final product. In our case, the conditioning phase is the bottling-and-packaging which has enough storage facility.

In order to maintain the quality of the products, we have to schedule cleaning operations regularly. These maintenance operations cannot be neglected and this condition leads to sequence-dependent setup times. The setup time depends on which wort is processing and which is going to be processed. Indeed, the frequency of these operations depends on the production sequence and the type of wort. The processing time and setup time in each machine are different since the machines are
unlike and depend on the wort type. The process features specify that not all jobs can be processed by any machine, hence, there is a compatibility between machines and worts.

In this scheduling problem, the jobs to be scheduled are composed of a set of operations (worts). Each job is composed of only one wort type. The size of the job (batch) depends on the size of the fermentation tank that has been assigned to the job. A job can be seen as a batch of worts, whose size is not known a priori. Notice that, the boiling of worts from the same batch (job) can be split into the available boiling lines, but the fermentation of the entire job occurs in a unique tank. Moreover, the scheduling of each operation depends on the previous operation, due to the precedence constraints.

In this work, we focus on the scheduling of wort production, and we consider only the boiling and fermentation stages- the most critical stages in the brewing process. The key features of this problem are the existence of parallel machines, multiple stages of production, multiple products, batch production, and sequence-dependent setup times. Furthermore, the batches have different sizes, and the total number of batches are unknown. Detailed information can be found in Chapter 5.

1.3 Relevance

Scheduling and planning in the breweries are two important ways of decision-making which have a significant impact in productivity level of the company. Moreover, market researchers have shown that Mexican beer has a constant growth in market volume and value. Concerning total volume, it is expected to maintain sustained growth with a Compound Annual Growth Rate (CAGR) of 3% in the forecast period 2015-2020. As a result, sales of beer are expected to reach 7.6 billion liters in 2020. In 2013, the Mexican beverages market grew by 4.0% to 67.8 billion USD from the previous year (65.1 billion USD). Between 2013 and 2018, the market is forecasted
to have a CAGR of 5.1% to reach the value of 87.1 billion USD in 2018. The market volume of Mexican beverage industry has been growing at a standard rate of 4.6%, since 2009. In 2013, the consumption volume of the country’s industry grew 4.1% from 62.1 billion liters in 2012 to 64.6 billion liters. By 2018, it is expected to grow at a standard rate of 4.5% to reach 80.6 billion liters. Due to this high demand and low facilities, production scheduling is inevitable to maintain the high-quality product in minimum time and cost.

Since the operations research point of view, the scheduling process in the brewing company is similar to a variant of the classical Job-shop Problem (JSP). In the general JSP, \( n \) jobs have to be processed in \( m \) unrelated machines, with respect to the precedence relations, job-machine compatibility, and availability of machines. The general JSP is among the hardest combinatorial optimization problems since it is strongly NP-hard (Garey et al. [68]).

The FJSP is an extended version of the JSP by assuming that a job has to be processed by any of the compatible machines (Brucker and Knust [21]). Moreover, the classical FJSP involves two important decisions -the operation-machine assignment and the order in which the operations must be processed in the machines, such that the makespan is minimized. Thus, the FJSP is a much more complex version of the JSP, so the FJSP is strongly NP-hard and combinatorial (Mati and Xie [118]). Since the problem under study is an extended version of the FJSP, the complexity of the problem is NP-hard.

In the FJSP literature, several heuristics and metaheuristics have been proposed as solution procedures. However, to the best of our knowledge, none of the published work has addressed the scheduling problems we address in this work, which asserts the relevance of this thesis.

Besides, in modern industries, the primary cause of resource constraints could be machines, human resources, and raw materials. Machines do have certain limitations to process the operations. For instance, each machine can handle only one
operation at a time, and each operation can be executed by at most one machine. Additionally, the execution of an operation can be restricted by the presence of scarce resources, which asserts the importance of the scheduling subject to the resource constraints. Moreover, the sequence-dependent setup time between operations and resource available in batches, as well as the renewability nature of resources are other complex features in modern industries, which we can see in this work.

In this thesis, we present a general FJSP problem and a case study from a local company. We propose a mathematical formulation for the FJSP with resource recovery constraints and a metaheuristic to solve it. For the case study, we propose a metaheuristic which has been tested on real data, the obtained solutions outperformed the scheduling generated by the expert in the company and the results have been published in Sáenz-Alanís, César A., V. D. Jobish, M. Angélica Salazar-Aguilar, and Vincent Boyer. “A parallel machine batch scheduling problem in a brewing company”. The International Journal of Advanced Manufacturing Technology 87, no. 1-4 (2016): 65-75.

1.4 OBJECTIVES

The objective of this research is to study a scheduling problem found in a brewing company and characterize it according to the existing literature. One goal is to represent the problem using a mathematical formulation and solve it by using the CPLEX optimizer, in order to obtain optimum solutions and determine the size of instances that can be handled by CPLEX. Another aim is to propose solution procedures based on metaheuristics like Greedy Randomized Adaptive Search Procedures (GRASP) and General Variable Neighborhood Search (GVNS). Finally, the performance of the proposed algorithms will be analyzed on a large set of artificial instances as well as on real instances.
1.4.1 Particular Objectives

i. Analyze the characteristics of the problem under study

ii. Classify and generalize the problem

iii. Define the problem and propose an efficient mathematical formulation

iv. Design and implement an instance generator for the problem under study

v. Validate the mathematical model using the CPLEX optimizer

vi. Characterize the problem under study and propose alternative solution procedures

vii. Evaluate the performance of the proposed algorithms on a large set of instances

viii. Compare the obtained results and analyze the problem nature

ix. Present preliminary results in scientific forums

x. Write and submit to revision the scientific articles derived from this work.

1.5 Organization

The content of this thesis is organized as follows: Chapter 2 presents a literature review related to our problems under study. Chapter 3 provides a detailed description and the proposed mathematical formulation of the flexible job-shop scheduling problem with resource recovery constraints. The proposed solution procedure and the analysis of results for the problem described in Chapter 3 are presented in Chapter 4. Chapter 5, contains a case study of the problem which is related to a real situation faced by a local brewing company (the solution procedure and the obtained results are also analyzed in this chapter). Finally, the main conclusions and future work are presented in Chapter 6.
Chapter 2

LITERATURE REVIEW

Resource management is the most important decision of any scheduling problem. The most known resources are the machines (Gargeya and Deane [69], Figielska [64]) that will process the operations, but one can also consider the availability of the workforce, tools, or raw materials. For instance, in the classical Resource Constrained Project Scheduling Problem (RCPSP) each operation requires certain units of resources to be processed; hence they might not be scheduled at their earliest starting time but later when the resources are available (Kolisch [93]). Due to the nature of the resources based on their availability, they are classified into four main categories: Słowiński [167] identified three categories of resources, i.e., renewable resources, nonrenewable resources, and doubly constrained resources; and later, Böttcher et al. [16] introduced partially renewable resources.

- A renewable resource has a limited availability, but it can be reused or recovered during the overall process. For instance, in most scheduling problems, machines are renewable resources since each machine can be assigned to another operation once the processing of the current one finishes.

- The nonrenewable resources can be used only once in the entire process. Hence, at the beginning, a certain amount of resources are available and it can be consumed throughout the planning horizon. For instance, in a car manufacturing
company, the paint used and the money spent for the car production are nonrenewable resources since their availability is limited and they cannot be reused.

- A *doubly-constrained resource* is both renewable and nonrenewable; that is, both usage and consumption are constrained. For instance, human resources are renewable since when an employee finishes a job, he can be assigned to another job, but the total working hours that can be assigned to an employee within a week is limited.

- *Partially renewable resources* are used to model resources whose availability changes over the planning horizon. For instance, the staff availability in a university varies at each hour depending on the shifts; hence the employees are considered as partially renewable resources.

### 2.1 Resources in the Flexible Job-shop Scheduling Problem

The classical Flexible Job-shop Scheduling Problem (FJSP) was introduced by Brucker and Schlie [24], in terms of Multi-Purpose Machines (MPM job-shop problem), and it is a generalized form of the classical Job-shop Scheduling Problem (JSP) (Graham et al. [71]). The general JSP is strongly NP-hard as shown by Garey et al. [68]. In the FJSP, each job consists of a sequence of operations, and each operation has to be performed in order to complete the job. The execution of each operation has to be processed in one of the subsets of compatible machines. Then the problem is to define a sequence of operations together with the assignment of the starting time and the machine for each operation. Mati and Xie [118] have shown that the classic FJSP is NP-hard since it can be reduced to a JSP, by assuming that an operation can be processed by only one machine.

Traditionally, two kinds of approaches are widely used for solving the FJSP:
hierarchical and integrated approaches. The assignment of operations to machines and the sequencing of operations on the machines are treated separately in the former one, and the assignment and sequencing are done simultaneously in the latter one. Hierarchical approaches are based on decomposing the original problem in order to reduce its complexity. Brandimarte [19] was the first to use the hierarchical approach on FJSP, by solving a routing subproblem first and then the scheduling subproblem. Integrated approaches were introduced by Dauzére-Pérés and Paulli [47]. They extended the disjunctive graph representation of the FJSP in order to perform the machine assignment and the sequencing of the operations at the same time. Fattahi et al. [60] show that the hierarchical algorithms have better performance over the integrated ones. However, Vaessens et al. [178], Dauzére-Pérés and Paulli [47], Hurink et al. [83], and Gambardella and Mastrolilli [65] reported better results with integrated approaches, even though the solution complexity is higher.

The pioneer work of FJSP with resource constraints is due to Chan et al. [30] and is termed as Resource-Constrained FJSP (RCFJSP). The problem under study includes multiple flexible machines for the operations, fixture and tool constraints, and operation dependent setup times. In particular, an operation of a job can be processed on a machine only if the associated fixture and tool are both available. Besides, it is assumed that all these resources, fixtures and tools, are renewable. Reference is also made to the work of Rajkumar et al. [148] and Karthikeyan et al. [88] who studied a similar problem. A GRASP algorithm is proposed in the former work, and a discrete firefly algorithm is adopted in the latter work as the solution procedure.

A flow shop scheduling problem with additional resources has been studied by Figielska [61]. The author studied two-stage flow shop scheduling problem with parallel machines, where preemption is permitted, and jobs use additional renewable resources apart from the machines in the first stage. All required resources are granted to the jobs for processing or resuming after preemption, and they are returned by the job on finishing the process. Other variants of this problem have
been studied afterward by Figielska [62, 63, 64]. Later, Cheng et al. [37] studied the resource-constrained flow shop scheduling with separate resource recycling operations. It is a two-stage flow shop problem where, initially, a common resource pool is available. A job can be processed only if the machine and a certain amount of resources are available. At the end of the process, a reduced amount of the resources is returned.

Mati et al. [116] have modeled and solved a practical FJSP with blocking constraints. The jobs have to wait on the current machine if the next machine where the job is going to be processed is occupied. This approach yields to a different management of resources through this blocking process called blocking constraints (Hall and Srisankarajah [74]), or the deadlock avoidance problem (Mati et al. [117]). Another variant of the FJSP is studied by Ortiz et al. [144], where multi-resource consumption is required for each operation. Besides, setup times and the times required to transfer the batches to the machines are also considered.

Mixed integer linear programming models for the FJSP have been proposed by Roshanaei et al. [152], Demir and İşleyen [55], Yulianty and Ma’ruf [194], and Shen and Yao [163]. Although, due to the complexity of the problem, mainly metaheuristics have been investigated. The most used metaheuristics include Simulated Annealing (SA) (Li et al. [106], Shao et al. [159]), Tabu Search (TS) (Zhang et al. [197], Li et al. [104], Bożejko et al. [18], Li et al. [105]), Genetic Algorithm (GA) (Li and Huang [101], Costa [42], Defersha and Chen [52], Zhang et al. [200, 196], Azzouz et al. [8]), Particle Swarm Optimization (PSO) (Huang et al. [82], Singh et al. [165], Rey et al. [150], Singh and Mahapatra [164]), Ant Colony Optimization (ACO) (Liouane et al. [109], Xing et al. [189]), Variable Neighborhood Search (VNS) (Huang et al. [82], Gao et al. [66], Li et al. [103], Tayebi Araghi et al. [174], Yazdani et al. [192], Lei and Guo [97], Bagheri and Zandieh [10], Gao et al. [66], Zhang et al. [195]), Greedy Randomized Adaptive Search Procedure (GRASP) (Rajkumar et al. [149, 148]), among others. A detailed review on the FJSP can be found in Chaudhry and Khan [32].
2.2 Resources in Other Scheduling Problems

In this section, we discuss the main contributions on scheduling problems with resource constraints. In Table 2.1 we present an overview of the related literature. The first column shows the author names and the second one the year of publication. The column ‘Rr’ refers to renewable resources, ‘Nr’ to nonrenewable resources, ‘Dc’ to doubly constrained resources, and ‘Pr’ to partially renewable resources.

Nonrenewable resources have limited availability in the entire project. Many of the works in the literature model the available money as the nonrenewable resource, (see Akkan et al. [1], Demeulemeester et al. [53], Nudtasomboon and Randhawa [142], Tareghian and Taheri [173]), with the objective of minimizing the production cost. Artigues et al. [7] generalize the notion of nonrenewable resources where a job can consume or produce such a resource. Each job needs a fixed number of units of nonrenewable resources at the starting time of its execution. If the resource requirement for a job is positive then the job consumes some units of the resource; and if the requirement is negative, then the job produces some units of the resource. Kolisch and Drexl [94] show that, in the case of RCPSP, nonrenewable resources may lead to an unfeasible problem, and with at least two nonrenewable resources, the problem is NP-hard. Extensive reviews on scheduling problems with nonrenewable resources can be found in Brucker et al. [20], Hartmann and Briskorn [76], and Orji and Wei [143].

A renewable resource can be used multiple times for the entire project duration. The most common renewable resources in the literature are the machines since a machine can be assigned to handle more than one operation, i.e. the machine availability is reset every time it finishes processing an operation. The renewable resources can also be found in scheduling problems in the form of tools, fixtures, labor force, raw materials, etc. As we can see in Table 2.1, many researchers worked on the RCPSP, FJSP, and their variants while dealing with renewable resources.
### Table 2.1: Literature on Scheduling Problems with Resource Constraints

The most relevant works of the FJSP with resource constraints have been discussed in the previous section, hence, in the section, we will focus on the RCPSP and its variants.
Few works have considered only renewable resources in the context of RCPSPs. For instance, Bianco and Caramia [12], and Naber et al. [135] proposed a mathematical formulation for the RCPSP with scarce resources. Chyu and Chen [39] proposed a payment model for the RCPSP. In this model, the contractor receives a profit for each job processed before a predetermined due date, and a penalty otherwise. The authors propose several VNS algorithms for the solution of this problem. Later, Chen and Chyu [33] proposed a model that aims to minimize the payment risk failure during a project execution. A memetic algorithm (MA) (Moscato et al. [132]) and a metaheuristic based on the VNS (Mladenović and Hansen [127]) algorithm, i.e. the Double Variable Neighborhood Search (DVNS) algorithm (Chyu and Chen [39]), are presented as solution procedures. The concept of resilience on RCPSP is introduced by Xiong et al. [190], where operations have an uncertain duration. The resilience refers to the capacity of the schedule to absorb perturbation.

Recently, Naber and Kolisch [134] proposed and evaluated, four different discrete time model formulations for the RCPSP with flexible resource profiles. In this work, one discrete time system is commonly used for all activities and resources. Resources required by an operation are classified into three general categories, namely principal, dependent, and independent resources. In this context, the resource quantity of a dependent resource depends on the one of a principal resource. The resources that do not depend on any other resource are termed as independent resources.

Dual resources have been introduced to model the machine-labor assignment. A model and design of machine and labor limited production systems have been presented by Nelson [136], and is considered to be a seminal work on Dual Resource Constrained (DRC) scheduling problems. The model includes a statistical arrival process (time interval between successive arrivals of jobs) and processing facilities consisting of some machine centers and a labor force. The author also considers the relative efficiency of each worker at each machine. Likewise, ElMaraghy et al. [59] studied a production scheduling problem in dual resource constrained manufacturing systems and proposed a GA as a solution procedure. Jaber and Neumann [84]
studied a DRC system to model the effects of human fatigue and recovery to avoid work overloading and injury to employees.

The dual resource constrained problem has also been studied in the context of JSP by taking into account resource constraints derived from both machines and workers. This problem is known as Dual Resource Constrained Job-shop Scheduling Problem (DRCJSP) (Treleven and Elvers [176]). The Extension Dual Resource Constrained Job-shop Scheduling Problem (EDRCJSP) is studied by Li and Huang [100]. In this problem, several batches of jobs need to be processed before a deadline on machines that can be operated by a set of workers with different efficiency. The authors proposed a hybrid method based on GA and ACO algorithm to solve the EDRCJSP, i.e. the Branch Population Genetic Algorithm (BPGA). This BPGA has also been used to solve the classical DRCJSP (see Li and Huang [101], Li et al. [102]).

As a combined version of FJSP and DRCJSP, Zheng and Wang [201], Yazdani et al. [193], Lei and Tan [98], and Paksi and Ma’ruf [145] studied the Dual Resource Constrained Flexible Job-shop Scheduling Problem (DRCFJSP). Lang and Li [96] considered the dual-resources as machines and workers. The authors present an algorithm that combines grey simulation technology and non-dominated sorting genetic algorithm-II (Deb et al. [51]) to solve the problem. Many solution procedures are proposed towards the solution of DRCFJSP: VNS (Lei and Guo [97]), SA and Vibration Damping Optimization (VDO) (Yazdani et al. [193]), GA (Paksi and Ma’ruf [145]), Knowledge Guided Fruit-fly Optimization Algorithm (KGFOA) (Zheng and Wang [201]), among others. Recently, Lei and Tan [98] studied DRCFJSP with multiple objectives and a novel local search with controlled deterioration (CDLS) was proposed to minimize the makespan and the total tardiness.

In certain scheduling problems, operations may need more than one resource to be performed. These problems are commonly known as Multi-Resource Constrained FJSP (Gao and Pan [67]) or Multi-Resource Constrained Scheduling Prob-
lems (Multi-RCPSP) (Huang et al. [81], Davis and Heidorn [49]). Demeulemeester and Herroelen [54], and Patterson and Roth [146] proposed, respectively, a branch-and-bound algorithm and an integer programming approach for the Multi-RCPSP. Dauzère-Pérès et al. [48] introduced the notion of resource flexibility, where for each operation a list of candidate resources is defined (see, for instance, Mati and Xie [119]). Almeida et al. [4] studied a Multi Skill Resource Constrained Project Scheduling Problem (MSRCPSP), where each resource, such as an employee, have a set of skills to perform certain operations.

Another variant of scheduling problems with resource constraints is the Multi-mode RCPSP (MRCPSP) (Talbot and Patterson [171]), in which each operation can be processed in one out of several execution ways, called modes. Each execution mode represents an alternative combination of resource requirements for an operation that can affect its processing time. For example, a job can finish with 10 employees in 10 days, while it can be done in only 6 days with 20 employees. Many authors presented mathematical models for the MRCPSP (see Schnell and Hartl [156, 155], Hermel et al. [80], Chakrabortty et al. [29], Beşikci et al. [11], Cheng et al. [35], and Naber and Kolisch [134]). A survey on multi-mode resources can be found in Hartmann and Briskorn [76] and Weglarz et al. [187].

Solution approaches for the MRCPSP with renewable resources include Branch and Bound (Sprecher and Drexl [168], Brucker et al. [22], Sprecher et al. [169], Kolisch et al. [95]), Branch and Cut (Zhu et al. [202]), GA (Ba et al. [9], Beşikci et al. [11], Van Peteghem and Vanhoucke [179], Lova et al. [111], Mati and Xie [119], Hartmann [75], Ulusoy et al. [177]), Local Search Heuristics (Wang and Fang [183], Kolisch and Drexl [94], De Reyck and Herroelen [50]), SA (Bouleimen and Lecoq [17]), PSO (Jarboui et al. [85], Zhang et al. [199]), and TS (Mika et al. [126]), among others. The most recent work on MRCPSP is due to Hermel et al. [80]. This research is motivated by a firm that develops container packing software for doors distribution at a cross-dock facility. The arrival times of the containers are programmed with the required resources such as human resources and machines. A methodological
framework and various mathematical models are presented as solution procedures.

We also make reference to the work of Wong et al. [188], and Ba et al. [9] who studied the multi-mode scheme in the context of JSP and FJSP, respectively. Wong et al. [188] considered the case where each operation requires a fixture to secure the process on the machine and each operation requires a tool to be processed. The authors proposed two heuristic approaches based on GA and PSO. Ba et al. [9] proposed a new mathematical model for a multi-resource flexible job-shop scheduling problem (MRFJSP). Machines, warehouses, vehicles, and detection equipment are the resources in this problem.

The MRCPSP has also been considered with both renewable and nonrenewable resources. For instance, Beşikci et al. [11] proposed a heuristic based on GA for this problem and Cheng et al. [35] explored the difference between preemption, activity splitting, and non-preemptive activity splitting. Van Peteghem and Vanhoucke [180] presented an overview of the existing metaheuristics to solve the MRCPSP with both renewable and nonrenewable resources. Moreover, a new benchmark dataset is proposed by the authors.

As we have seen before doubly constrained resources, introduced by Słowinski [167], are limited both for each period and for the whole project. For instance, Weglarz [186] studied a project scheduling problem where the doubly constrained resource is used to represent the number of staff-hours per day or week. In the literature, most of the authors did not consider doubly constrained resources alone, but they are combined with other types of resources.

The MRCPSP is studied by many researchers with renewable, nonrenewable, and doubly-constrained resources together. Bouleimen and Lecocq [17] proposed a heuristic based on SA as the solution procedure. Ulusoy et al. [177] considered the cash out- and in-flows associated with the operations, where at the beginning of each operation there is a cost (out-flow), and payments are received at payment points (in-flow) during the process. The operations are to be scheduled such that the overall
project does not exceed a given due date. De Reyck and Herroelen [50] proposed a local search based methodology, that divides the problem into two distinct phases: a mode assignment phase and a resource-constrained project scheduling phase with fixed mode assignments. Sprecher and Drexl [168] considered a branch-and-bound solution procedure for this problem.

When the availability of a resource changed at each period of the planning horizon, like timetabling and employees’ shifts in universities, it is referred as the partially renewable resource as introduced by Böttcher et al. [16]. Alvarez-Valdes et al. [5] described some preprocessing techniques and develop a heuristic algorithm, based on Scatter Search, for project scheduling problems under partially renewable resources. Alvarez-Valdés et al. [6] presented heuristics based on GRASP and path relinking. Drexl et al. [58] introduced a generalized project scheduling model where precedence constraints, multiple modes of resources, and multiple resources requirement are considered. The authors presented different formulations based on the RCPSP model and an instance generator.

Finally, some authors studied the RCPSP considering the three different kinds of resource constraints: renewable, nonrenewable, and doubly resource constrained resources. Nudtasomboon and Randhawa [141] considered multiple objectives and developed an implicit enumeration technique as a solution procedure. Kolisch and Drexl [94] showed that the problem may lead to an unfeasible solution with at least two nonrenewable resources and hence the complexity of the problem is NP-complete. De Reyck and Herroelen [50] solved the multi-mode version of the problem by dividing the problem into two distinct phases: a mode assignment phase and a resource-constrained project scheduling phase with fixed mode assignments. Zhu et al. [202] presented a branch and cut algorithm with local branching, variable reduction techniques, cuts generation, and bound tightening in order to accelerate the convergence of the algorithm. Finally, Van Peteghem and Vanhoucke [179] presented a GA where operation splitting is allowed. The reader is also referred to the book of Brucker and Knust [21] that discusses various branch-and-bound algorithms and
heuristic approaches for this problem.

Apart from these major classification of resources, there are some other less common variants that can be found in the literature. If the available amount of a resource is an arbitrary number from an interval, as the petroleum in a moving car, such resources are called continuous (or continuously divisible) resources (Weglarz [185], Schwindt and Zimmermann [157]). Some resources such as containers, tanks, or other storage facilities have to store final or intermediate products during the production process. Such facilities could be depleted and replenished over time. To represent these resources, Selle [158] proposed the notion of cumulative resources (see Neumann and Schwindt [137], Neumann et al. [138, 139], and Brucker and Knust [21]). Gargeya and Deane [69] termed the resources that improve the productivity of labor and machine as auxiliary resources, such as machine attachments and accessories, or equipment for transportation.

2.3 Batch scheduling problems in parallel machines

In many of the modern industries, the production process must be handled in batches. Due to the extensive literature on Batch Scheduling Problems (BSP) and the characteristics of our problem under study, we concentrate only on BSP on parallel machines. The other major characteristics of our problem under study are: sequence-dependent setup time ($ST_{sd}$) due to cleaning process between operations or other shutdowns during the process; family ($f$) of products due to the similar characteristics of the operations or products; resource ($res$) availability; and capacity ($C$) constraints of the machines. In Table 2.2, we present most of the related works, to the best of our knowledge, with their main characteristics. Apart from the works presented in the table, some other works can be found in the surveys of Potts and Kovalyov [147], Allahverdi et al. [3], Méndez et al. [124], Drexl and Kimms
From Table 2.2, one can see that only Lozano and Medaglia [113] meets all the major characteristics of our problem. The authors studied a problem found in an automotive safety glass manufacturing facility and proposed a two-phase heuristic with two evaluation criteria: total tardiness and machine utilization. Although they consider batches of different capacity, batch splitting is not allowed, and the number of batches is fixed, contrary to our problem. The products with same ballistic level belong to the same family, and the processing time depends on the family.

Crauwels et al. [43] studied a batch scheduling problem with parallel machines where the jobs are partitioned into families. The jobs that require the same processing facility belong to the same family. The authors proposed a heuristic, where a job sequence is constructed by repeatedly solving a knapsack problem for each machine.
Méndez et al. [122] proposed a two-step systematic methodology, batching and scheduling, for the scheduling of a single stage multi-product batch plant. The batches are of different sizes according to the customer order of each product. The authors (Mendez and Cerdá [123]) also studied a reactive scheduling problem with resource-constrained multistage batch facilities. They proposed a MILP framework to solve this problem where the goal is to update the current schedule when unforeseen events occur.

Mixed integer formulations have been proposed in the literature to represent some variants of batch production problems (see for instance Lozano and Medaglia [113], Cheng et al. [34], Wang et al. [184]). Many authors proposed heuristics for solving the BSP such as GA (Costa [42], Noroozi et al. [140], Kashan et al. [89], Malve and Uzsoy [114]), ACO (Cheng et al. [34]), TS (Shen et al. [160], Buscher and Shen [27]), SA (Costa [42], Wang and Chou [182], Kashan et al. [89]), VNS (Shen et al. [161, 162]), and GRASP (Lozano and Medaglia [113], Damodaran et al. [44]), among others.

Many authors address batch scheduling problems where job families play an important role. The diffusion and oxidation operations in the wafer fabrication process are studied by Klemmt et al. [92], and Malve and Uzsoy [114]. Due to the chemical nature of the process, it is impossible to process jobs with different recipes together in the same batch. Hence the jobs are classified into family accordingly. Scheduling of heat-treatment operations of steel casting is studied by Mathirajan et al. [115]. In this scheduling problem, the jobs (castings) are classified into a number of job families based on the alloy type. These families are further classified into various sub-families based on the type of heat-treatment operations required. In general, the notion of family is used to group jobs with similar requirements (tools operation sequence, processing times, etc). The reader is referred to the work of Shen et al. [162, 161], Li and Yuan [107], Buscher and Shen [27], and Brucker et al. [23] where families of the product are considered in the context of batch production systems.
In many of the parallel batch scheduling problems, capacity constraints are considered due to the limitations of machines. In this context, the batch size must be less than the capacity of the machine (Wang and Chou [182], Buscher and Shen [28], Chung et al. [38], Chang et al. [31]). For instance, Wang et al. [184] study a problem where the capacity constraint due to the fuel storage capacity of the machines is considered. In Kashan et al. [89], the limited capacity of the burn-in oven in a semiconductor manufacturing problem leads also to a capacity constraint. Machines with fixed capacity (Jia and Leung [86], Mehta and Uzsoy [121]), identical capacity (Cheng et al. [34]), or different capacities (Wang and Chou [182], Klemmt et al. [92]) have also been considered in the literature.

Due to the complexity of the parallel batch scheduling problems with capacity constraints, most of the researchers proposed heuristics to solve the problem. For instance, Jia and Leung [86], and Cheng et al. [34] proposed an ACO based meta-heuristic, Wang and Chou [182] used a GA and a SA, Klemmt et al. [92] used a VNS heuristic, Kashan et al. [89] proposed a Hybrid Genetic Heuristic (HGH) based on a batch-based representation, and Mehta and Uzsoy [121] presented a dynamic programming algorithm. Moreover, Mixed Integer Non-Linear Program (MINLP) (Torabi et al. [175]), and Mixed Integer Linear Programming (MILP) (Cheng et al. [34], Wang and Chou [182], Buscher and Shen [28]) representations can also be found in the literature.

Due to the pace of development and breadth of research, a truly comprehensive review is probably impossible, and certainly beyond the scope of this thesis. This literature review presents the major work related to our problems. The scheduling problems address in this thesis are based on a real situation found in a local brewery. We study this situation as a parallel batch scheduling problem with sequence-dependent setup time, family considerations, resource constraints, and capacity constraints. From this problem, we derived a flexible job-shop scheduling problem with renewable resources where the resources are renewed by batch. To the best of our
knowledge, both of these problems have characteristics that have not been addressed in the literature.
Chapter 3

SCHEDULING PROBLEM WITH

RENEWABLE RESOURCES

In this chapter, we study a variant of the FJSP where resource constraints and recovery time of resource batches are taken into account. This variant has been called as the Flexible job-shop scheduling problem with Resource Recovery Constraints (FRRC). In the FRRC, any operation may require some resources to be processed and these resources are available in a limited quantity. Once a batch of resources is depleted, it can be recovered; which induces a recovery time (dead time in the production line). Moreover, the processing time of each operation depends on the selected machine, and each machine can process a subset of operations. Thus the machines are known as ‘dedicated machines’.

In order to provide a better idea about the studied problem, this chapter provides a brief description of the FJSP and the FRRC through two examples and presents the proposed mathematical formulation for the FRRC.
3.1 THE CLASSICAL FLEXIBLE JOB-SHOP SCHEDULING PROBLEM

The classical Flexible Job-shop Scheduling Problem (FJSP) consists of scheduling a set $J = \{j_i : i = 1, 2, 3, \ldots, n\}$ of $n$ jobs on a set $M = \{m_i : i = 1, 2, 3, \ldots, m\}$ of $m$ machines. A job $j \in J$ is composed of a set $O_j = \{o_{ji} : i = 1, 2, 3, \ldots, n_j\}$ of $n_j$ operations. An operation $o_{ji}$ can be processed by any of the machines $m$, such that $m \in M_{ji}$; $M_{ji} \subseteq M$ is the set of compatible machines for operation $o_{ji}$. The processing time $p_{jim}$ of an operation $o_{ji}$ depends on the machine $m$. Each machine $m$ can process at most one operation at a time. The operations must be processed without preemption, i.e., the operations cannot be interrupted until the execution of the operation is completed. The sets of jobs and machines as well as the compatibility operation-machine are known in advance. Then, the problem is to assign each operation to a compatible machine and determine the sequence followed by the operations on those machines with the objective of minimizing the makespan.

3.1.1 EXAMPLE OF THE CLASSICAL FJSP

Consider a $2 \times 3$ FJSP, i.e., a FJSP with two jobs and three machines. Let $J$ be the set of jobs, then $J = \{j_i : i = 1, 2\}$ and $M$ be the machine set, $M = \{m_i : i = 1, 2, 3\}$. Job $j_1 \in J$ is composed by a set $O_1 = \{o_{1i} : i = 1, 2, 3, 4, 5\}$ of five operations, and job $j_2 \in J$ is composed by a set $O_2 = \{o_{2i} : i = 1, 2, 3, 4\}$ of four operations. Each operation $o_{ji}$ has a candidate machine set $M_{ji}$, where the operation can be performed, with a certain processing time $p_{jim}$. The complete data of the candidate machine set, operation-machine compatibility, and processing time are shown in Table 3.1. If an operation cannot be processed in a machine (incompatible), we represent this by a ‘-’ symbol corresponding to the machine.

For operation $o_{21}$, for instance, the candidate machine set $M_{21}$ is $\{m_1, m_3\}$. 
By referring to Table 3.1, it could be precisely understandable that operation $o_{21}$ can be processed in $m_1$ with a processing time of 4 units, and in $m_3$ with 1 unit of processing time, and these times are represented by $p_{211} = 4$ and $p_{213} = 1$, respectively. Operation $o_{21}$ cannot be processed in machine $m_2$, due to machine-operation incompatibility, and that is represented by a ‘-’ symbol in the table.

<table>
<thead>
<tr>
<th>Job</th>
<th>Operation</th>
<th>$Machine_{-id}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$m_1$  $m_2$  $m_3$</td>
</tr>
<tr>
<td>$o_{11}$</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$j_1$</td>
<td>$o_{12}$</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>$o_{13}$</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>$o_{14}$</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>$o_{15}$</td>
<td>1</td>
</tr>
<tr>
<td>$o_{21}$</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>$j_2$</td>
<td>$o_{22}$</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>$o_{23}$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>$o_{24}$</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3.1: The $2 \times 3$ FJSP data

A feasible solution of the $2 \times 3$ FJSP, with the data given in Table 3.1, is shown in Figure 3.1 through a Gantt chart. The constraints that should be taken into account in the FJSP can be enlisted as:

i. Each job $j$ has to be processed by following an operations order (precedence constraints). i.e., an operation $o_{ji}$ cannot start if the previous operation $o_{ji-1}$ has not finished;

ii. Each machine can perform at most one operation at a time;

iii. Assignment of operations has to be done over compatible and available machines;
iv. Each operation cannot be interrupted during its execution (non-preemption condition).

The objective is to find the schedule with the shortest makespan, i.e., the minimum time required to process all operations in the job-shop.

Figure 3.1: A solution representation of a classical FJSP based on the instance described in Table 3.1

As we have mentioned before, when we add the resource constraints to the classical FJSP, we obtain the FRRC.

### 3.2 The Flexible Job-Shop Scheduling Problem with Resource Recovery Constraints (FRRC)

Besides the classical constraints of the FJSP, in some of the existing industries, the operations may consume renewable resources. Suppose $O_{jr}$ be the set of operations of job $j$ that require the resource $r$ to be processed. To start processing an operation, the required resources and a compatible machine must be available simultaneously. Notice that, some operations may not require renewable resources to be processed. Let $R$, be the set of available resources. A resource $r \in R$ is available in batches that contain $b_r$ units of the same resource $r$. 
Each batch of resources can be recovered with a recovery time $\tau_r$, soon after the entire batch is completely consumed. Therefore, it can induce some dead time in the production line during the recovery process, i.e., between each batch of the same resource $r$.

Then, the problem is to assign each operation to a compatible machine and to determine its processing sequence on the machines by taking into account the availability of resources and machines in order to minimize the makespan. The resulting problem is called as the Flexible job-shop scheduling problem with Resource Recovery Constraints (FRRC). An example of the $2 \times 3 \times 2$ FRRC—an FRRC with 2 jobs, 3 machines, and 2 resources, is presented here, which is developed by adding FRRC constraints along with FJSP constraints that we have seen in section 3.1.1.

### 3.2.1 Example of the FRRC

Consider a $2 \times 3 \times 2$ FRRC with a set of two jobs, $J = \{j_i : i = 1, 2\}$, a set of machines, $M = \{m_i : i = 1, 2, 3\}$, and a set of resources $R = \{r_i : i = 1, 2\}$. Suppose that job $j_1$ has five operations and $j_2$ has four operations. The operation-machine compatibility, corresponding processing time, and resource requirements of the operations are shown in Table 3.2(a). One can see for instance that operation $o_{12}$, the second operation of job $j_1$, has the candidate set of machines $M_{12} = \{m_1, m_2\}$ with a processing time equal to 3 and 1, respectively, and which cannot be processed in machine $m_3$. Furthermore, operation $o_{12}$ requires resource $r_2$, while operation $o_{14}$ does not require any resource to be processed. Besides, the resource recovery time $\tau_r$ and batch size $b_r$ are shown in Table 3.2(b).
A feasible solution of the $2 \times 3 \times 2$ FRRC, based on Table 3.2 is presented in Figure 3.2 by means of a Gantt chart. Notice that the set of constraints involved in the FRCC are:

i. Each job $j$ has to be processed by following an operations order (precedence constraints). i.e., an operation $o_{ji}$ cannot start if the previous operation $o_{ji-1}$ has not finished;

ii. Each machine can perform at most one operation at a time.

iii. Assignment of operations has to be done over compatible machines.

iv. Each operation cannot be interrupted during its execution (non-preemption condition).
v. An operation can be assigned to a machine, if and only if the machine and the required resource unit are available.

vi. The resource-operation allocation must not exceed the batch size, during any period.

vii. Before assigning the first unit of a resource batch, to any operation, there must be a recovery time of the previous batch, if any.

d. Recovery time is affected from the starting time of the last unit of resource in a resource batch.

As in the FJSP, the objective of the FRRC is to find the schedule with the shortest makespan, i.e., the minimum time required to process all operations in the job-shop.

![Gantt chart](image)

Figure 3.2: A solution representation of a $2 \times 3$ FRRC based on Table 3.2

In this Gantt chart, the operation-machine assignment and the operations sequence are presented along with a resource allocation and it is shown with a blue dot bar for $r_1$ and a red dot bar for $r_2$. For instance, consider an assignment of resource $r_1$, with batch size equal to 2, and a recovery time equal to 4 units, see
Table 3.2(b). The last unit of the first batch of resource \( r_1 \) is assigned to operation \( o_{22} \) which takes place at time 1. Thus, the next batch of resource \( r_1 \) is available for any other operation after the starting time of the operation \( o_{22} \), at time 1, plus the recovery time (4 units of time), i.e., \( r_1 \) will be available at time 5. So, during the period (1,5), where no units of resource \( r_1 \) is available, operations, that require \( r_1 \), cannot be processed. This unavailability of the resource is shown in a blue dots bar on the top of the chart, with name \textit{recovery of \( r_1 \)}. Hence, for this case, the next operation \( o_{13} \) can be assigned to any compatible machine, using resource \( r_1 \), at time 5. Even though the machine \( m_2 \) is available and the precedence operation \( o_{12} \) was completed at time 4, operation \( o_{13} \) cannot be assigned to any machine until resource \( r_1 \) is available. However, multiple operations can use the same type of resource while the size of the batch is not reached, for instance, operations \( o_{11} \) and \( o_{22} \) are using the same resource \( r_1 \), in the period (1,2), because the batch size of resource \( r_1 \) is equal to 2.

Moreover, the resources cannot be recovered by units but can be recovered by batches, and the recovery time of a batch starts soon after the assignment of the last unit of the resource. For instance, the recovery of the first batch of resource \( r_2 \) starts at time 3, and it is available for processing at time 6. The second batch of the same resource starts at time 8, and it is available for processing at time 11, since the batch size of resource \( r_2 \) is two, and the recovery time is equal to 3 units of time.

We have seen the complexity of the FJSP is \( NP-hard \). The FJSP is a particular case of the FRRC, on the assumption that none of the operations require a resource to be processed. Thus, we can conclude that the FRRC is also \( NP-hard \). Moreover, we represent this problem with a mixed integer linear programming model (MILP) which is described in the following section.
3.3 MATHEMATICAL FORMULATION

The FRRC formulation is based on the one proposed by Roshanaei [151] for the FJSP.

The decision variables used in the mathematical model are the following:

\[ x_{jilk} = \begin{cases} 
1, & \text{if operation } o_{ji} \text{ is processed after operation } o_{lk} \\
0, & \text{otherwise} 
\end{cases} \]

\[ y_{jim} = \begin{cases} 
1, & \text{if operation } o_{ji} \text{ is assigned to machine } m \\
0, & \text{otherwise} 
\end{cases} \]

\[ z_{ijrq} = \begin{cases} 
1, & \text{if operation } o_{ji} \text{ is assigned to the } q^{th} \text{ unit of resource } r \\
0, & \text{otherwise} 
\end{cases} \]

\( c_{ji} \): completion time of operation \( o_{ji} \)

\( s_{ji} \): starting time of operation \( o_{ji} \)

\( t_{rq} \): starting time of the \( q^{th} \) unit of resource \( r \)

Using these decision variables, we then proposed the following formulation for the FRRC:

\[
\begin{align*}
\text{min } & \quad C_{\text{max}} \\
\text{s.t. : } & \quad \sum_{m \in M_{ji}} y_{jim} = 1, \quad \forall j \in J, 1 \leq i \leq n_j, \quad (3.1) \\
& \quad \sum_{j \in J} \sum_{i \in O_{jr}} z_{ijrq} \leq 1, \quad \forall r \in R, 1 \leq q \leq n_r, \quad (3.2) \\
& \quad \sum_{q=1}^{n_r} z_{ijrq} = 1, \quad \forall r \in R, j \in J, i \in O_{jr}, \quad (3.3) \\
& \quad c_{ji} \geq c_{j(i-1)} + \sum_{m \in M_{ji}} p_{jim} y_{jim}, \quad \forall j \in J, i \in O_j, \quad (3.4)
\end{align*}
\]
Chapter 3. Scheduling problem with renewable resources

\( c_{ji} = s_{ji} + \sum_{m \in M_{ji}} p_{jim} y_{jim} \), \quad \forall j \in J, i \in O_j, \quad (3.6) 

\( c_{ji} \geq c_{lk} + p_{jim} - M(3 - x_{jilk} - y_{jim} - y_{lkm}) \), \quad \forall j < n, i \in O_j, l > j, k \in O_l, \quad m \in M_{ji} \cap M_{lk}, \quad (3.7) 

\( c_{lk} \geq c_{ji} + p_{lkm} - M(x_{jilk} + 2 - y_{jim} - y_{lkm}) \), \quad \forall j < n, i \in O_j, l > j, k \in O_l, \quad m \in M_{ji} \cap M_{lk}, \quad (3.8) 

\( t_{r(q-1)} \leq t_{rq} \), \quad \forall r \in R, 1 < q \leq n_r, \quad (3.9) 

\( t_{r(q-1)} + \tau_r \leq t_{rq} \), \quad \forall r \in R, 1 < q \leq n_r, q \equiv 0 \mod b_r, \quad (3.10) 

\( s_{ji} \geq t_{rq} + M(z_{jirq} - 1) \), \quad \forall j \in J, i \in O_j, r \in R, 1 \leq q \leq n_r, \quad (3.11) 

\( s_{ji} \leq t_{rq} + M(1 - z_{jirq}) \), \quad \forall j \in J, i \in O_j, r \in R, 1 \leq q \leq n_r, \quad (3.12) 

\( c_{jn} \leq C_{\text{max}} \), \quad \forall j \in J, \quad (3.13) 

\( s_{ji} \geq 0 \), \quad \forall j \in J, i \in O_j, \quad (3.14) 

\( t_{rq} \geq 0 \), \quad \forall r \in R, 1 \leq q \leq n_r, \quad (3.15) 

\( c_{ji} \geq 0 \), \quad \forall j \in J, i \in O_j, \quad (3.16) 

\( x_{jilk} \in \{0, 1\} \), \quad \forall j \in J, i \in O_j, l \in J, k \in O_l, \quad (3.17) 

\( y_{jim} \in \{0, 1\} \), \quad \forall j \in J, i \in O_j, m \in M_{ij}, \quad (3.18) 

\( z_{jirq} \in \{0, 1\} \), \quad \forall j \in J, i \in O_j, r \in R, 1 \leq q \leq n_r. \quad (3.19) 

The constant \( M \) represents a big number. The objective (3.1) corresponds to the minimization of the makespan. Constraints (3.2) and (3.3) assure that each operation is assigned to one and only one machine and that each unit of resource is assigned to at most one operation, respectively. Since for each resource, we can compute the exact number of units necessary to process all the operations, each unit of resource will be assigned to at most one operation. The fact that each operation should consume exactly one unit of the required resource is modeled by constraints (3.4), and constraints (3.5) modeled the precedence relationship within the operations of a job. Linking the starting time of an operation with its completion time is
modeled by constraints (3.6). Constraints (3.7) and (3.8) prevent the overlapping of operations processed on the same machine. Constraints (3.9) give priorities to the use of resource units in order to remove symmetries. Constraints (3.10) guarantee the recovery time between batches of the same resource. Constraints (3.11) and (3.12) synchronize the starting time of the operation with the one of its assigned unit of resource. Constraints (3.13) are related to the makespan. Constraints (3.14)-(3.19) state the nature of the variables.

This mathematical model has been implemented in C++ and solved by using the branch-and-bound method included in the IBM CPLEX Optimizer. The obtained results are discussed in Chapter 4.
Chapter 4

Solution procedure for the FRRC

In this chapter, we describe the proposed solution procedure for solving the FRRC as well as the validation of the model described in Chapter 3. The proposed solution procedure is based on a well-known metaheuristic called General Variable Neighborhood Search (GVNS). In the GVNS, the first component (shaking) is used to avoid local optima by randomly selecting a solution in a defined neighborhood of the best-known solution, then the second component (local search) improves the solution by exploring different neighborhoods. We designed five shaking procedures and five local searches based on different neighborhood structures. The neighborhood structures are explained with the help of diagrams in this chapter. Extensive computational tests have been carried out, on a large set of instances, in order to evaluate the performance of the proposed procedure. The analysis of the results obtained by the proposed GVNS is presented at the end of this chapter.

4.1 Variable Neighborhood Search

The idea behind Variable Neighborhood Search (VNS) is that distinct local optimal solutions are obtained by applying unlike neighborhood operators on the same solu-
tion. VNS exploits this aspect of the solutions by employing multiple neighborhood operators and it diversifies the search by allowing different neighborhood operators to explore different regions in the search space. Before applying VNS to a solution, a set of neighborhood operators \( N = \{N_i : i = 1, 2, 3, \ldots, n_{max}\} \) must be defined, typically this set is ordered by increasing size of the neighborhood space. The set of solutions in the \( k^{th} \) neighborhood of \( x \) is denoted by \( N_k(x) \). In the literature, several variants of the VNS have been proposed. In this research, we propose a General Variable Neighborhood Search (GVNS) which is an advanced form of the Basic Variable Neighborhood Search (Basic VNS).

4.1.1 Basic Variable Neighborhood Search

The steps involved in the Basic VNS are presented in Algorithm 1. Before starting the Basic VNS, it is needed to define a set of neighborhood structures and the stopping criterion of the procedure. There is a big number of stopping criteria used in the literature, such as the maximum CPU time allowed, maximum number of iterations, maximum number of iterations between two improvements, etc.

In the Basic VNS presented in Algorithm 1, \( k_{max} \) neighborhood structures are available. These neighborhood structures define neighborhoods around any solution \( x \in X \), where \( X \) is the solution space. Then the local search is carried out over a randomly selected solution \( x' \) (shaking), which belongs to the neighborhood \( N_k(x) \), \( (k = 1 \) initially), of \( x \). After applying the local search routine over \( x' \), we obtain a local optimum solution \( x'' \) with cost \( f(x'') \). Then three possibilities can occur:

i.) \( x = x'' \) : then the procedure continues with the next neighborhood, i.e., \( k \leftarrow k + 1 \)

ii.) \( x \neq x'' \) & \( f(x'') \geq f(x) \) : then the procedure is iterated with \( k \leftarrow k + 1 \)

iii.) \( x \neq x'' \) & \( f(x'') < f(x) \) : then the procedure is restarted, \( k = 1 \), with the updated \( x \).
Algorithm 1 Basic VNS

Input:

\{N_k; k = 1, \ldots, k_{\text{max}}\} : Neighborhood structures

x : Initial solution

f(x): Local optimum of x

A stopping criterion

Output: \ x \ initial \ solution

1: \ Repeat
2: \ \ k \leftarrow 1
3: \ \ \textbf{while} \ k \leq k_{\text{max}} \ \textbf{do}
4: \ \ (a) \ \textit{Shaking}: \ Select \ randomly \ a \ solution \ x' \in N_k(x)
5: \ \ (b) \ \textit{Local search}: \ Find \ the \ best \ solution \ x'' \ from \ x
6: \ \ (c) \ \textit{Update solution}: \ If \ f(x'') < f(x) \ then \ x \leftarrow x'' \ and \ k \leftarrow 1; \ else \ k \leftarrow k + 1
7: \ \ \textbf{end while}
8: \ \ \textbf{until} \ Meet \ stopping \ criterion
9: \ \ \textbf{return} \ x

In case i), we iterate to the next neighborhood with \( k \leftarrow k + 1 \), because the
best-found solution is the same as the incumbent solution. Therefore, we are no
more interested in repeating the procedure with the same solution again and we
continue with the next neighborhood. In case ii.), we obtain a different best solution
in the neighborhood, but the local minimum value is worse than the one we have
so far (the incumbent). Then, we would not continue with this solution, and we
iterate to continue the exploration of the next neighborhood. Finally, in case iii.),
we obtain a different solution and a local optimum that is better than that of the
incumbent. In this case, the incumbent solution \( x \) is replaced by \( x'' \), and the search
is restarted with the first neighborhood. The last neighborhood should be reached
without a solution better than the incumbent one. The search is repeated at the
first neighborhood \( N_1(x) \) until a stopping criterion is met.
4.1.2 General Variable Neighborhood Search

The General Variable Neighborhood Search (GVNS) is a generalized version of the Basic VNS in which the simple local search is replaced by a set of local search structures, see Algorithm 2. The GVNS stops when none of the shaking procedures combined with the local search structures can improve the best-found solution.

**Algorithm 2** General VNS

**Input:**

\{N_k; k = 1, \ldots, k_{\text{max}}\} : Neighborhood structures for the shaking component

\{N_l; l = 1, \ldots, l_{\text{max}}\} : Neighborhood structures to be used in the local search

x : Initial solution by constructive heuristic

f(x) : Local optimum of x

A stopping criterion

**Output:** x initial solution

1: repeat
2: \hspace{0.5cm} k \leftarrow 1
3: \hspace{0.5cm} while \hspace{0.5cm} k \leq k_{\text{max}} \hspace{0.5cm} do
4: \hspace{1cm} (a) Shaking: Generate a solution x' at random from the k^{th} neighborhood N_k(x) of x
5: \hspace{1cm} (b) Local search:
6: \hspace{1.5cm} l \leftarrow 1
7: \hspace{1cm} repeat
8: \hspace{2cm} Find the best solution x'' \in N_l(x')
9: \hspace{1.5cm} if \hspace{0.5cm} f(x'') < f(x') \hspace{0.5cm} then \hspace{0.5cm} x' \leftarrow x'' \hspace{0.5cm} and \hspace{0.5cm} l \leftarrow 1; \hspace{0.5cm} else \hspace{0.5cm} l \leftarrow l + 1
10: \hspace{1cm} until \hspace{0.5cm} l > l_{\text{max}}
11: \hspace{1cm} (c) Update the best found solution: if \hspace{0.5cm} f(x'') < f(x) \hspace{0.5cm} then \hspace{0.5cm} x \leftarrow x'' \hspace{0.5cm} and \hspace{0.5cm} k \leftarrow 1; \hspace{0.5cm} else \hspace{0.5cm} k \leftarrow k + 1
12: \hspace{1cm} end while
13: \hspace{0.5cm} until \hspace{0.5cm} Meet the stopping criterion
14: return \hspace{0.5cm} x
In order to solve the FRRC we designed a solution procedure based on the GVNS scheme. The detailed description of the proposed algorithm is provided in the following section.

4.2 Solution procedure for FRRC based on GVNS

The proposed GVNS starts with an initial solution $x$ which is obtained through a constructive algorithm. The constructive algorithm can be seen in Algorithm 3. Notice that, the constructive algorithm is a greedy procedure where at each iteration the operation with the smallest release time is selected, and then it is assigned to the machine that leads to the earliest starting time. In our implementation, the release time of an operation is defined as its earliest starting time taking only into account the precedence constraints, and ignoring the machines and resources availability. If the assignment of an operation is not compatible with the precedence constraints, its release time is considered as infinity. The earliest starting time of an operation on a machine is computed according to the machine availability, and the resource recovery time of the required resource, if needed.

Once we have the initial solution $x$, reported by Algorithm 3, this solution is used to feed the GVNS (Algorithm 2). The stopping criterion is reached when the GVNS finds 10 consecutive non-improvement solutions. For a better understanding of the proposed algorithm, in the following subsections full details are provided.

4.2.1 Solution representation

In order to illustrate a solution and the neighborhood structures used in Algorithm 2, we represent a solution $x$ as an array whose size corresponds to the total number
Algorithm 3 Constructive Algorithm

Input:

\( x \): An empty solution
\( \mathcal{O} \): Set of operations to be assigned

Output: \( x \leftarrow \text{solution} \)

1. while \( \mathcal{O} \neq \emptyset \) do
2. \( o_{ji} \leftarrow \text{operation with the smallest release time, where } o_{ji} \in \mathcal{O}, j \in J, i \in O_j \)
3. \( \mathcal{O} \leftarrow \mathcal{O}\setminus\{o_{ji}\} \)
4. \( m \leftarrow \text{machine where } o_{ji} \text{ achieves the earliest starting time } t_{ij}; m \in M_{ji} \)
5. Assign \( o_{ji} \) to machine \( m \) at time \( t_{ij} \) in \( x \)
6. end while
7. return \( x \)

of operations to be scheduled. Each cell of the array contains a triplet \((j, o, m)\) where \( j \) represents the job index, \( o \) the operation index, and \( m \) the assigned machine index. Here we use only the indices of the job, operations, and machines, to ward off the intricacies of the solution array. For instance, \((2,1,3)\) refers to the assignment of operation one of job \( j_2 \) \((o_{21})\) to machine \( m_3 \). Consider the example that we saw in section 3.2.1. Here the total number of operations is 9, so the array size. A solution representation of the example is provided in Figure 3.2, and the array solution representation is shown in Figure 4.1.

| (2,1,3) | (1,1,1) | (2,2,2) | (1,2,2) | (1,3,1) | (2,3,2) | (1,4,3) | (2,4,1) | (1,5,1) |

Figure 4.1: Solution representation of Figure 3.2

From this solution array, it is possible to compute the starting time and the completion time of each operation through the greedy algorithm, where operations are assigned one by one, from left to right, in the solution array of the selected machine \( m \). Indeed, the resource availability should be taken into account during this process.
Moreover, because of the precedence constraints within the operations of a job, the order of appearance of the operation is known. Thus, from the array representation shown in Figure 4.1, the operation \( o \) in a triplet \( (j, o, m) \) can be deduced from \( j \) and its position in the table. Then we know that the first time a job \( j \) appears in the array, it is associated with its first operation, the second time it appears, it is related to the second operation, and so on. Otherwise, we would have an unfeasible solution, due to the precedence constraints. This can be used to maintain feasibility when the positions in the array are changed with the neighborhood exploration, assuming that there is no conflict in the machine assignment.

4.2.1.1 THE CRITICAL PATH AND THE CRITICAL BLOCK

In the neighborhood structures, we use the concept of the critical path and critical block. Consider the example that we have seen in section 3.2.1.

![Figure 4.2: Critical path of the solution representation shown in Figure 3.2](image)

Given a solution, its critical path is the sequence of operations that yields to the makespan value on that solution. For example, from Figure 4.2, the last completed
Chapter 4. Solution procedure for the FRRC

operation is $o_{15}$. The reason why operation $o_{15}$ has a delay is the unavailability of $r_2$, which has been used by $o_{24}$. Operation $o_{24}$ is delayed due to the precedence constraint of operation $o_{23}$, which is delayed due to the unavailability of $r_2$ that has been used by $o_{12}$. Operation $o_{12}$ is delayed due to the unavailability of machine $m_2$ and its precedence operation $o_{22}$, which is delayed due to the precedence relation with $o_{21}$. Thus, these operations together form a critical path. The formation of the critical path is illustrated using the red backward arrows and circles in Figure 4.2. Thus the operations $o_{15}, o_{24}, o_{23}, o_{12}, o_{22}$ and $o_{21}$ together form the critical path of the current solution.

A block is a continuous sequence of operations in the critical path. Although we can use different criteria to define a block, here in our case, the operations that integrate the block are the consecutive operations that belong to the critical path and are performed on the same machine. For example, from Figure 4.2, one can see that operations, $o_{22}, o_{12},$ and $o_{23}$ are processing on the same machine $m_2$. So these operations form a block. Operations $o_{24}, o_{15}$ are also processing on the same machine, $m_1$, hence they also form a block.

4.3 Neighborhood structures of the GVNS

As we can see in Algorithm 2, we need to define the neighborhood structures for both shaking and local search phases. We represent the $k^{th}$ neighborhood of $x$ in the shaking phase with $N_k(x)$, and the $l^{th}$ neighborhood of a solution $x$ in the local search phase by $\overline{N}_l(x)$. Indeed, shaking a solution $x$ consists of selecting a random solution from one of its neighborhoods $N_k(x)$, $k = 1, ..., k_{max}$, and a local search on a solution $x$ refers to finding the best solution in one of its neighborhoods $\overline{N}_l(x)$, $l = 1, ..., l_{max}$. Among all the neighborhoods we tested, we present here five neighborhoods for the shaking phase and five for the local search, which are the ones that appear to be the more efficient to solve the FRRC.
4.3.1 Neighborhood structures in the shaking phase

The shaking procedure allows to diversify the search and to escape from local optima, so the neighborhoods in the shaking phase must be of diverse nature as much as possible. The neighborhood structures used in the shaking are presented below, and throughout each of the neighborhoods, we refer as $x$ to the current (incumbent) solution and $x'$ as (resulting) one of its neighbor solutions.

4.3.1.1 $N_1$: Shifting a job in the critical path

Select two random triplets $(j, o, m)$ and $(j', o', m')$ from the critical path of the current solution $x$ and shift job $j$ to the next position of $(j', o', m')$. Then, update the operations according to their order of appearance in the array and keep the machine assignment in $x'$ as it is in $x$.

The move related to this neighborhood structure is shown in Figure 4.3. As one can see, for the given example, two random triplets, $(1, 2, 2)$ and $(2, 4, 1)$, have been selected from the critical path. Then, the job from the first triplet $(1, 2, 2)$ is shifted to the next position of the second triplet $(2, 4, 1)$. After that, the operation indices between the triplets that have been chosen are updated by preserving the machine assignment as in the original solution $x$. Finally, we get a resulting solution $x'$ which is a neighbor of $x$. 

![Figure 4.3: $N_1$: Shifting a job in the critical path](image-url)
4.3.1.2 $N_2$: Swapping two jobs in the critical path

Choose two random triplets from the critical path of a solution $x$, namely $(j, o, m)$ and $(j', o', m')$, respectively. Then, the job indices $j$ and $j'$ are swapped and the operations order in each position is updated between them, and the machine assignment is kept as in $x$. A schematic representation of this move is shown in Figure 4.4.

Figure 4.4: $N_2$: Swapping two jobs in the critical path

4.3.1.3 $N_3$: Swapping two jobs with machine reassignment

Two random triplets $(j, o, m)$ and $(j', o', m')$ are chosen from a solution $x$. Then, jobs $j$ and $j'$ are swapped and the operations order in each position are updated, while the machine assignment is kept as in $x$. Finally, the operations $o_{jo}$ and $o_{j'o'}$ are reassigned to a different compatible machine selected at random. A schematic representation of this move is shown in Figure 4.5.
4.3.1.4 $N_4$: Shifting a job in the critical path with machine reassignment

Two triplets $(j, o, m)$ and $(j', o', m')$ are randomly selected from the critical path. Then the shifting movement of $N_1$ is applied, and operations $o_{jo}$ and $o_{j'o'}$ are reassigned to a different compatible machine which is randomly chosen. This neighborhood is very similar to $N_1$.

4.3.1.5 $N_5$: Shifting a job and reassigning to a different machine

Two random triplets $(j, o, m)$ and $(j', o', m')$ are chosen from a solution $x$, and job $j$ is shifted after the position of $(j', o', m')$ as in neighborhood $N_1$. Notice that in this case, the triplets $(j, o, m)$ and $(j', o', m')$ are not necessarily part of the critical path. Then, the operation $o_{jo}$ is assigned to another compatible machine which is randomly chosen.
4.3.2 NEIGHBORHOOD STRUCTURES FOR THE LOCAL SEARCH

A local search consists of exploring all the neighbor solutions of a given solution $x'$, and returning the best neighbor $x''$ according to an objective function. An objective function can maximize profits, minimize costs, minimize makespan, etc. In our case the objective function is the makespan, i.e., the minimization of the completion time of the last job. The defined neighborhood should not be too large, to avoid time overhead, and should contain enough good quality solutions to improve the incumbent solution found so far.

4.3.2.1 $N_1$: Reassigning the machines in the critical path

The triplets are taken one by one, from the critical path of the incumbent solution $x'$, going from left to right in the solution array. For each of the triplets $(j, o, m)$, the corresponding operation $o_{jo}$ is reassigned to the machine $m' \in M_{jo}$ that leads to the smallest makespan. If there is not a machine assignment that gives a better solution than the current one, the machine assignment remains the same. This process is applied to the next triplet on the critical path, and so on. A schematic representation of this procedure is illustrated in Figure 4.6.

![Figure 4.6: Reassigning the machines in the critical path](image-url)
4.3.2.2 \( \mathcal{N}_2 \): Reassigning the machines

This neighborhood is very similar to \( \mathcal{N}_1 \), but instead of performing the machine reassignment in the critical path, it is carried out on a subsequence of the incumbent solution \( x' \). In order to reduce the size of the neighborhood, the number of operations of this subsequence is limited to 15% of the total number of operations. But in order to maintain a diversity of the neighborhoods, the first operation is randomly selected. The resultant feasible solution is named as \( x'' \).

4.3.2.3 \( \mathcal{N}_3 \): Swapping two jobs and reassigning the machines in a block

Two random triplets \((j, o, m)\) and \((j', o', m')\) are selected in a block of the solution \( x' \) and swapped as in \( N_2 \). Then, operations \( o_{j_0} \) and \( o_{j'_0} \) are reassigned to compatible machines by figuring out the machine that produces the smallest makespan. In this neighborhood, all possible swaps within the chosen block are explored.

4.3.2.4 \( \mathcal{N}_4 \): Shifting a job

A subsequence of operations from solution \( x' \) is randomly selected such that its length is approximately 15% of the total number of operations. Then the first operation of this subsequence is shifted to another position within this subsequence. All possible positions for the shifting are explored, while the machine assignment is preserved.

4.3.2.5 \( \mathcal{N}_5 \): Swapping two jobs

A subsequence of operations from solution \( x' \) is randomly selected such that its length is approximately 15% of the total number of operations. Then two triplets,
(j, o, m) and (j', o', m'), are selected, after that these triplets swap their positions to find out the best sequence that produces the minimum makespan, whilst conserving the machine assignment. All possible swaps are explored within this subsequence.

4.4 Computational Results

The proposed GVNS described in Algorithm 2 has been coded in C++, and the mathematical model has been used to solve the FRRC with the branch and bound available in CPLEX 12.6.3. All tests were carried out on a computer with a Processor 3.7 GHz Quad-Core Intel Xeon E5 with 12GB of RAM memory. For the tests, we used a set of benchmark instances adapted from the ones proposed by Dauzère-Pérès and Paulli [46] and Dauzère-Pérès and Paulli [47]. The resource requirement of operations was randomly chosen as well as the batch size of each resource. We made sure that at least two batches are available for the process. The recovery time was randomly selected between the minimum and the maximum processing time of the operations, to get consistent values according to the instance. As of stopping criterion for CPLEX, we considered one hour of computation time.

The characteristics of each class of the tested instances are presented in Table 4.1. Each row in this table is related to the instance class, in which ten different configurations of resources have been tested, the makespan and gap (%) are the averages of the ten instances. For instance, ‘frrc5_08’ is a class of ten instances with the same number of operations and machines, but with different resource requirements configuration. The number ‘5’ in the name indicates that five resources are used in the instance class. Moreover, the number of jobs, operations, machines, and the number of resources used in each class of the instance is shown explicitly in Table 4.1.
Chapter 4. Solution procedure for the FRRC

Table 4.1: Description of the tested instances

<table>
<thead>
<tr>
<th>Instances</th>
<th>Number of -</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>jobs operations machines resources</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>frrc1_01</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_02</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_03</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_04</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_05</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_06</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_07</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_08</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_09</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_10</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_11</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_12</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_13</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_14</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_15</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_16</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_17</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>frrc1_18</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>frrc5_01</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_02</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_03</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_04</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_05</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_06</td>
<td>10</td>
<td>196</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_07</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_08</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_09</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_10</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_11</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_12</td>
<td>15</td>
<td>293</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_13</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_14</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_15</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_16</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_17</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>frrc5_18</td>
<td>20</td>
<td>387</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

The mathematical model presented in section 3.3 has been implemented in C++, and solved with the branch and bound included in CPLEX. The results reported by CPLEX are presented in Table 4.2. Notice that in the cases where CPLEX did not report a feasible solution, within the one-hour time limit, the symbol ‘-’ is used. For each class of the instances, we report the average makespan in the *makespan* column, the average optimality gap reported by CPLEX in column *gap* (%), and the number of instances where a feasible solution has been found out of the ten instances of the class in shown in the last column *#sol*. The averages have been computed taking into account only the instances where a feasible solution has been found. Note that, even for the smallest instances, CPLEX was not able to
solve to optimality any of them and the optimality gap stays relatively high. For instances with more than 15 jobs, CPLEX struggles to find any feasible solution. The behavior with one or five resources is quite similar.

Table 4.2: CPLEX results using the proposed mathematical formulation

<table>
<thead>
<tr>
<th>Instances</th>
<th>makespan</th>
<th>gap (%)</th>
<th># sol</th>
</tr>
</thead>
<tbody>
<tr>
<td>frrc1_01</td>
<td>10273.00</td>
<td>86.05</td>
<td>1/10</td>
</tr>
<tr>
<td>frrc1_02</td>
<td>10138.33</td>
<td>85.87</td>
<td>9/10</td>
</tr>
<tr>
<td>frrc1_03</td>
<td>10030.50</td>
<td>86.11</td>
<td>6/10</td>
</tr>
<tr>
<td>frrc1_04</td>
<td>10276.33</td>
<td>85.99</td>
<td>3/10</td>
</tr>
<tr>
<td>frrc1_05</td>
<td>9953.67</td>
<td>86.26</td>
<td>9/10</td>
</tr>
<tr>
<td>frrc1_06</td>
<td>10168.13</td>
<td>86.82</td>
<td>8/10</td>
</tr>
<tr>
<td>frrc1_07</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc1_08</td>
<td>15133.20</td>
<td>90.75</td>
<td>5/10</td>
</tr>
<tr>
<td>frrc1_09</td>
<td>15085.44</td>
<td>90.72</td>
<td>9/10</td>
</tr>
<tr>
<td>frrc1_10</td>
<td>15124.00</td>
<td>90.66</td>
<td>2/10</td>
</tr>
<tr>
<td>frrc1_11</td>
<td>14983.40</td>
<td>91.08</td>
<td>5/10</td>
</tr>
<tr>
<td>frrc1_12</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc1_13</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc1_14</td>
<td>19967.60</td>
<td>93.22</td>
<td>10/10</td>
</tr>
<tr>
<td>frrc1_15</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc1_16</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc1_17</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc1_18</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>Averages</td>
<td>12830.33</td>
<td>88.5</td>
<td>3.72/10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instances</th>
<th>makespan</th>
<th>gap (%)</th>
<th># sol</th>
</tr>
</thead>
<tbody>
<tr>
<td>frrc5_01</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc5_02</td>
<td>8270.43</td>
<td>83.04</td>
<td>7/10</td>
</tr>
<tr>
<td>frrc5_03</td>
<td>8858.67</td>
<td>84.16</td>
<td>9/10</td>
</tr>
<tr>
<td>frrc5_04</td>
<td>8589.50</td>
<td>83.15</td>
<td>8/10</td>
</tr>
<tr>
<td>frrc5_05</td>
<td>8889.89</td>
<td>84.57</td>
<td>9/10</td>
</tr>
<tr>
<td>frrc5_06</td>
<td>9218.80</td>
<td>85.45</td>
<td>5/10</td>
</tr>
<tr>
<td>frrc5_07</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc5_08</td>
<td>13995.60</td>
<td>89.99</td>
<td>5/10</td>
</tr>
<tr>
<td>frrc5_09</td>
<td>14881.67</td>
<td>90.57</td>
<td>3/10</td>
</tr>
<tr>
<td>frrc5_10</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc5_11</td>
<td>13666.33</td>
<td>90.21</td>
<td>3/10</td>
</tr>
<tr>
<td>frrc5_12</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc5_13</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc5_14</td>
<td>18725.63</td>
<td>92.77</td>
<td>8/10</td>
</tr>
<tr>
<td>frrc5_15</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc5_16</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc5_17</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>frrc5_18</td>
<td>-</td>
<td></td>
<td>0/10</td>
</tr>
<tr>
<td>Averages</td>
<td>11677.39</td>
<td>87.1</td>
<td>3.17/10</td>
</tr>
</tbody>
</table>

Table 4.3 shows the results obtained with the proposed GVNS, presented in Algorithm 2. Column Instances contains the name of the class, column Ini sln refers to the initial solution obtained with the constructive algorithm (Algorithm
3), and column imp shows the percentage of improvement between the best-found solution in column makespan and the initial solution Ini sln. The column time shows the average processing time in seconds. As one can see, the GVNS is able to improve the initial solution by around 9% and the processing time stays below 1500 s, in both the cases. Interestingly, the processing time is smaller with five resources; this is mainly due to an early convergence and then an early termination of the search.

<table>
<thead>
<tr>
<th>Instances</th>
<th>Ini sln</th>
<th>makespan</th>
<th>imp</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>frrc1_01</td>
<td>3557.20</td>
<td>3175.40</td>
<td>10.67</td>
<td>79.22</td>
</tr>
<tr>
<td>frrc1_02</td>
<td>2834.40</td>
<td>2533.70</td>
<td>10.46</td>
<td>102.17</td>
</tr>
<tr>
<td>frrc1_03</td>
<td>2521.20</td>
<td>2354.70</td>
<td>6.56</td>
<td>166.46</td>
</tr>
<tr>
<td>frrc1_04</td>
<td>3328.10</td>
<td>2878.50</td>
<td>12.75</td>
<td>95.98</td>
</tr>
<tr>
<td>frrc1_05</td>
<td>2599.00</td>
<td>2374.40</td>
<td>8.27</td>
<td>127.13</td>
</tr>
<tr>
<td>frrc1_06</td>
<td>2770.20</td>
<td>2574.30</td>
<td>7.68</td>
<td>128.73</td>
</tr>
<tr>
<td>frrc1_07</td>
<td>3007.80</td>
<td>2662.50</td>
<td>11.38</td>
<td>412.31</td>
</tr>
<tr>
<td>frrc1_08</td>
<td>2521.90</td>
<td>2253.50</td>
<td>10.47</td>
<td>513.60</td>
</tr>
<tr>
<td>frrc1_09</td>
<td>2251.40</td>
<td>2180.80</td>
<td>3.12</td>
<td>392.71</td>
</tr>
<tr>
<td>frrc1_10</td>
<td>3057.40</td>
<td>2719.60</td>
<td>10.93</td>
<td>469.05</td>
</tr>
<tr>
<td>frrc1_11</td>
<td>2963.00</td>
<td>2670.70</td>
<td>9.23</td>
<td>300.29</td>
</tr>
<tr>
<td>frrc1_12</td>
<td>2374.80</td>
<td>2254.50</td>
<td>4.68</td>
<td>446.55</td>
</tr>
<tr>
<td>frrc1_13</td>
<td>3434.40</td>
<td>2885.40</td>
<td>15.34</td>
<td>1031.93</td>
</tr>
<tr>
<td>frrc1_14</td>
<td>2507.60</td>
<td>2370.20</td>
<td>5.43</td>
<td>949.06</td>
</tr>
<tr>
<td>frrc1_15</td>
<td>2388.50</td>
<td>2299.20</td>
<td>3.66</td>
<td>1484.57</td>
</tr>
<tr>
<td>frrc1_16</td>
<td>4382.00</td>
<td>3972.70</td>
<td>12.16</td>
<td>817.31</td>
</tr>
<tr>
<td>frrc1_17</td>
<td>3910.10</td>
<td>3722.20</td>
<td>6.24</td>
<td>893.03</td>
</tr>
<tr>
<td>frrc1_18</td>
<td>2978.60</td>
<td>2878.70</td>
<td>3.38</td>
<td>1337.27</td>
</tr>
<tr>
<td>Averages</td>
<td>2965.98</td>
<td>2707.44</td>
<td>8.47</td>
<td>541.52</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instances</th>
<th>Ini sln</th>
<th>makespan</th>
<th>imp</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>frrc5_01</td>
<td>3420.20</td>
<td>2974.40</td>
<td>12.89</td>
<td>76.63</td>
</tr>
<tr>
<td>frrc5_02</td>
<td>2960.20</td>
<td>2547.40</td>
<td>13.44</td>
<td>80.27</td>
</tr>
<tr>
<td>frrc5_03</td>
<td>2722.40</td>
<td>2520.50</td>
<td>7.05</td>
<td>60.99</td>
</tr>
<tr>
<td>frrc5_04</td>
<td>3388.20</td>
<td>2860.90</td>
<td>15.33</td>
<td>69.22</td>
</tr>
<tr>
<td>frrc5_05</td>
<td>2702.20</td>
<td>2467.20</td>
<td>8.41</td>
<td>55.85</td>
</tr>
<tr>
<td>frrc5_06</td>
<td>2519.40</td>
<td>2397.70</td>
<td>4.80</td>
<td>80.94</td>
</tr>
<tr>
<td>frrc5_07</td>
<td>3908.50</td>
<td>3183.70</td>
<td>17.25</td>
<td>234.98</td>
</tr>
<tr>
<td>frrc5_08</td>
<td>2755.20</td>
<td>2487.50</td>
<td>8.47</td>
<td>164.74</td>
</tr>
<tr>
<td>frrc5_09</td>
<td>2893.10</td>
<td>2702.30</td>
<td>5.12</td>
<td>155.20</td>
</tr>
<tr>
<td>frrc5_10</td>
<td>3353.20</td>
<td>2869.50</td>
<td>13.88</td>
<td>326.79</td>
</tr>
<tr>
<td>frrc5_11</td>
<td>3096.20</td>
<td>2754.10</td>
<td>10.09</td>
<td>174.13</td>
</tr>
<tr>
<td>frrc5_12</td>
<td>2396.10</td>
<td>2278.30</td>
<td>4.31</td>
<td>184.44</td>
</tr>
<tr>
<td>frrc5_13</td>
<td>3218.80</td>
<td>2850.00</td>
<td>11.32</td>
<td>678.35</td>
</tr>
<tr>
<td>frrc5_14</td>
<td>2960.30</td>
<td>2817.40</td>
<td>4.56</td>
<td>369.73</td>
</tr>
<tr>
<td>frrc5_15</td>
<td>3489.20</td>
<td>3408.10</td>
<td>2.03</td>
<td>327.83</td>
</tr>
<tr>
<td>frrc5_16</td>
<td>3777.80</td>
<td>3219.20</td>
<td>14.16</td>
<td>706.10</td>
</tr>
<tr>
<td>frrc5_17</td>
<td>2800.30</td>
<td>2610.40</td>
<td>5.63</td>
<td>417.00</td>
</tr>
<tr>
<td>frrc5_18</td>
<td>2773.90</td>
<td>2664.50</td>
<td>3.16</td>
<td>525.69</td>
</tr>
<tr>
<td>Averages</td>
<td>3063.07</td>
<td>2756.28</td>
<td>9.00</td>
<td>260.49</td>
</tr>
</tbody>
</table>

Table 4.3: GVNS results

When many resources are introduced, we have more flexibility in the arrangement of the operations and less total waiting time for the availability of the resources. With only one resource, it is more likely that we generate large dead time in the
process during the resource recovery period which leads to a more complex scheduling. Alternating operations using different resources is not an option in this case. However, the improvement of the initial solution stays relatively consistent with one and five resources. If we compare the results reported by CPLEX and the GVNS, we can remark that the average makespan of the GVNS is much smaller than the one reported by CPLEX. The difference is around 75% and the processing time of the GVNS stays relatively low.
In this chapter, we present a case study from a local brewing company. The objective is to propose a complete beer production schedule in order to minimize the makespan. It can be seen as a multi-stage batch scheduling problem where each stage involves multiple parallel machines and sequence-dependent setup times. The jobs are grouped in families of products according to the kind of beer to be produced and the capacity of the different tanks used during the process should not be exceeded. Besides the tanks, another renewable resource to be considered is the yeast, required for the fermentation process of beer.

To the best of our knowledge, this problem has not been previously addressed in the literature. We propose a Greedy Randomize Adaptive Search Procedure (GRASP) for its solution. The contents of this chapter have been published in: Sáenz-Alanís, César A., V. D. Jobish, M. Angélica Salazar-Aguilar, and Vincent Boyer. “A parallel machine batch scheduling problem in a brewing company”. The International Journal of Advanced Manufacturing Technology 87, no. 1-4 (2016): 65-75.
5.1 PROBLEM DESCRIPTION - THE BREWING PROBLEM

In the beer brewing process, beers are obtained from the production of worts, which contain the sugars that will be fermented to produce alcohol. The peculiarities of a beer depend on the malt used to produce the wort and on the yeast. Since different kinds of beers can be obtained by a brewery, the production of different kind of worts has to be scheduled. Furthermore, the process is constrained by the cleaning operation of the tanks, the availability of the tanks, the time required to empty a tank, and the availability of the yeasts. The process consists of three main phases: the coction, the fermentation, and the conditioning. In this work we consider only the two first phases since, as observed in our case study, there are always enough tanks in the conditioning phase to receive the worts leaving the fermentation tanks.

Let $W$ be the set of wort types and $d_w$ the demand of each wort type $w \in W$, then the objective is to schedule the coction and the fermentation of the worts in order to satisfy the demand as soon as possible. Coctions can be carried out on a set $H$ of coction houses, which is composed of tanks with capacity $q_h$, $h \in H$. The coction houses can operate in parallel and are unrelated. For each wort type $w \in W$, there is a subset of compatible coction houses $H_w \in H$. In the fermentation phase, there is a set $T$ of heterogeneous tanks available. Each tank $t \in T$ has a capacity $k_t$.
which represents the number of worts needed to fill the tank $t$.

In this context, we model a job as a batch of alike worts. The number of worts on this batch is equal to the capacity $q_h$ of the fermentation tank $h$ used for the second phase. Hence, the batch size for each job is unknown a priori and, consequently, the total number of jobs to be scheduled either. Although, the fermentation of a batch occurs in a unique tank, the coction of worts within a batch can be done in parallel on the different coction houses. Nevertheless, one has to assure that the assigned fermentation tank is available when the first wort of the batch leaves the coction house, and only the wort from the associated batch can fill this fermentation tank.

The availability of the yeast should also be considered. Let $Y$ be the set of yeasts for the fermentation phase and $n_y$ the maximum number of consecutive jobs that can use yeast $y \in Y$ before it is depleted. Each wort $w \in W$ can be fermented by using a specific yeast $y_w \in Y$. Hence, a yeast $y \in Y$ is used for a subset $F_y \subset W$ of wort type. $F_y$ will be called the family of worts type that are compatible with yeast $y$. In order to assure the availability of yeast, the production of worts using a different kind of yeast is alternated. Indeed, this policy is used by the company in order to ensure yeast recovery. Besides, as we will see in the following, this strategy is particularly relevant when considering the maintenance activities.

Maintenance activities occur at each stage of the process. The coction tanks should be cleaned every time there is a changed in the wort type, and also after $n$ consecutive coctions of the same wort type. Likewise, the fermentation tanks should be cleaned after each use. The duration of these cleaning operations depends on the wort yielding to Sequence Dependent Setup Time. Moreover, a general maintenance of the production lines should be scheduled every 7 to 10 days.
5.2 Solution Procedure for the Brewing Problem

For the solution of this NP-hard scheduling problem, we propose a Greedy Randomized Adaptive Search Procedure (GRASP). GRASP is a multi-start process, where each iteration consists of a construction phase and an improvement phase. The construction phase constructs a solution for the problem, then this solution is improved through the improvement phase, which is a local search procedure. The general steps of the GRASP algorithm is presented in Algorithm 4, where $S^b$ is the best-found solution.

**Algorithm 4 GRASP algorithm**

**Input:**

- Instance of the problem
- $it_{max}$: Maximum number of iterations for the GRASP

**Output:** $S^b$ Best found solution.

```
1: $j \leftarrow 0$
2: $S^b \leftarrow \emptyset$
3: while $j < it_{max}$ do
4:   $S \leftarrow$ construct_solution()
5:   $S' \leftarrow$ improve_solution($S$)
6:   Update $S$
7:   Update $S^b$
8:   $j \leftarrow j + 1$
9: end while
10: return $S^b$
```
5.2.1 CONSTRUCTION PHASE

This phase consists of constructing an initial solution which will be fed to the improvement phase. At each step of the algorithm, an empty batch $B$ is created and the wort type to be produced in $B$ is randomly selected from a Restricted Candidate List (RCL). The RCL contains the two types of worts, one that has the largest uncovered demand and the one whose required yeast is available at the moment. The batch $B$ is assigned to the first available fermentation tank $t^* \in T$, which gives the number $n_B$ of coctions that should be programmed. The required coctions are then assigned iteratively to the earliest available coction house. The procedure is repeated until the demand of all the wort types is satisfied. This constructive procedure is given in Algorithm 5.

**Algorithm 5** Constructive algorithm

<table>
<thead>
<tr>
<th>Input:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data of the problem</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S^b$ Best found solution.</td>
</tr>
</tbody>
</table>

1. $S \leftarrow \emptyset$
2. while Meet stopping criteria do
3. Build RCL of wort batches $B$
4. Select a random batch $B$ from RCL
5. Update $(S, S^b)$
6. Evaluate stopping criteria
7. end while
8. return $S^b$

5.2.2 IMPROVEMENT PHASE

The improvement phase tries to improve the constructed solution $S$. This procedure works in an iterative fashion by successively replacing the current solution $S^*$ by
the best solution $S'$ in its neighborhood. It finishes when no better solution has been found. The neighborhood of the current solution is obtained by randomly swapping two batches of different wort type and recomputing the makespan. If a better solution is found, it becomes the new current solution, and its neighborhood is afterward explored, and so on. An outline of this procedure is presented in Algorithm 6.

**Algorithm 6 Solution Improvement**

**Input:**
- Data of the problem
- $S$ : Initial solution

**Output:** $S^*$ Best found solution.

1: $S^* \leftarrow S$
2: while Meet stopping criteria do
3: Find $S' \in N(S^*)$, such that makespan of $S'$ is better than $S^*$
4: Update $S^*$
5: end while
6: return $S^*$

### 5.3 Experimental results

The computational experiments are carried out with real instances from a brewery and we also generated some random instances to evaluate the efficiency and robustness of the proposed GRASP. The GRASP Algorithm has been implemented in Java and tested on a Workstation HP Z620 with an Intel Xeon(R) CPU E5-2620 v2 (2.10 GHz) and 64GB of RAM memory. The main characteristics of these instances are,

- Two coction houses with different setup times
- Two sizes of fermentation tanks that can contain 4 or 8 units of worts
- Two kinds of yeasts
* At most three consecutive uses of each kind of yeast

* Six different wort types

For the real data, the brewery provided the demand per wort type on 20 cases from the year 2014. The comparison between the makespan (number of days required to carry out the production) obtained by GRASP and the one from the solutions implemented by the company are shown in Figure 5.2. It is clear that the GRASP outperforms the solution used by the decision maker over all tested instances.

![Figure 5.2: Makespan comparison with real data: company solution vs GRASP](image)

Table 5.1 presents a comparative study on the values reported by our GRASP and the one of the decision maker at the brewery. The number of days required for production is significantly reduced when GRASP is used. An average of 44.80% of the total number of production days needed by the company is saved. This result has a direct impact on the production capacity of the plant and the tolerance to faults that can occur throughout the process. Furthermore, the computation time required by the proposed GRASP is less than 11 seconds in average, while the decision maker has to work during a whole working day to obtain a feasible sequence of production per instance.
Chapter 5. Case study - Scheduling problem in a brewery

<table>
<thead>
<tr>
<th>Company (days)</th>
<th>GRASP (days)</th>
<th>Time (s)</th>
<th>Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>20.44</td>
<td>7.08</td>
<td>31.02</td>
</tr>
<tr>
<td>Avg</td>
<td>23.51</td>
<td>10.70</td>
<td>44.80</td>
</tr>
<tr>
<td>Max</td>
<td>29.78</td>
<td>15.49</td>
<td>57.90</td>
</tr>
</tbody>
</table>

Table 5.1: Computational Experiments with Real Data

The production demand in the brewery varies throughout the year: a low level of production during winter, a high level of production during summer, and a normal level of production in the rest of the year. Thus, to test the robustness of our heuristic, from the real data, we have generated three classes of instances with low, medium, and high demand.

For each class, we generated 20 instances, and we report the total number of days (makespan) required to fulfill the demand and the processing time used by the proposed GRASP. The results obtained are displayed in Table 5.2.

<table>
<thead>
<tr>
<th>Demand - Average deviation</th>
<th>GRASP(days)</th>
<th>time(seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>14.74</td>
<td>9.83</td>
</tr>
<tr>
<td>Medium</td>
<td>15.72</td>
<td>13.42</td>
</tr>
<tr>
<td>High</td>
<td>16.98</td>
<td>16.39</td>
</tr>
</tbody>
</table>

Table 5.2: Average results of instances with varying demands

From Table 5.2, one can notice that the proposed GRASP takes less than 19 seconds in average to provide a solution for those artificial instances. Furthermore, the processing time is consistent over all tested instances and in the worst case the number of days (makespan) required to fulfill the demand stays below 18 days while the company usually takes more than 20 days in the best case. These results show that a brewing company can highly benefit from such an approach when it needs to
schedule its production.

5.4 Conclusions

In this chapter, we have presented a scheduling problem found in a brewing company. Due to the complex nature of the problem, we propose a computational procedure based on the metaheuristic known as GRASP. The main characteristics of the problem are the existence of parallel machines, two stages of production, multiple products, batch production, and sequence-dependent setup times. Moreover, the batches have different sizes and their total number depends on the production schedule. The experimental tests over the real instances from the brewing company and over artificial instances show the high efficiency and the robustness of our approach. Indeed, our GRASP can generate a feasible sequence of production in a significantly negligible time in comparison to the current time required by the decision maker at the company. Furthermore, with the instances provided by the brewery, the solution obtained by GRASP saves a significant amount of time in production, which gives the opportunity to the company to increase its capacity of production without inverting in new equipments. The proposed approach can also be used to evaluate the need of a future expansion of capacity in the plant.
In this study, we introduce a new problem called the Flexible job-shop scheduling problem with Resource Recovery Constraints, i.e., FRRC. The primary characteristics of the FRRC are the renewable resources (which are available by batches), resource recovery time, and operation-resource compatibility. Some other features are the existence of multiple jobs composed of operations, multiple machines available in parallel, precedence constraints, operation-machine compatibility, and processing time dependent of the machine and operation. Several researchers have studied problems with some of these characteristics and proposed a variety of solution procedures and mathematical model representations towards these problems. But, to the best of our knowledge, none of the researchers have worked on a scheduling problem that tackles these features simultaneously.

A mixed integer linear formulation to represent the FRRC and a solution procedure based on a General Variable Neighborhood Search metaheuristic have been proposed. Computational tests have been carried out in a large set of instances adapted from the literature of the FJSP. The results clearly showed the efficiency of our proposed algorithm (see Chapter 4). Moreover, the manuscript derived from this
part of the research has been submitted for revision to The International Journal of Production Research.

Additionally, a case study from a local brewery has been carried out (see Chapter 5). A solution procedure based on GRASP has been proposed. The efficiency of the algorithm has been shown by solving real and random instances. The description of the algorithm as well as the obtained results are already published in: Sáenz-Alanís, César A., V. D. Jobish, M. Angélica Salazar-Aguilar, and Vincent Boyer. “A parallel machine batch scheduling problem in a brewing company”. The International Journal of Advanced Manufacturing Technology 87, no. 1-4 (2016): 65-75.

Notice that, the complex nature of the modern industries often involved many resources in their process, with limited production units and pieces of machinery. But the production must have to finish in a short time without loss in quality. In this context, a well-planned production is essential. On behalf of this research, we note that manual scheduling is very time consuming, and the outputs are not very efficient in practice. Besides, even linear mathematical models have limitations to provide optimum or near optimum solutions in a reasonably short period. Solution procedures based on GVNS or GRASP are light and easy to implement heuristics that could save time, money and improve significantly the production capacity of the companies.

In a future work, other solution approaches for the FRRC should be investigated in order to state the quality of the solutions obtained with the proposed GVNS. Besides, the mathematical model can be studied and improved in order to obtain feasible solutions for at least some instances. It could be also interesting to extend the problem where each operation uses more than one resource, with different availability profiles in order to tackle more complex industrial problems. Regarding the case study, the implementation in the company of the proposed GRASP would be an interesting task. Furthermore, one can also consider adding the conditioning
phase and the bottling phase in order to optimize the overall production process.

6.1 Autobiographical Reflection

Undertaking this research is an invaluable learning experience in my academic life and career. Before starting this research, I did not have much idea on how the products in a supermarket or available to order online are processed. Now I realize that behind each product there is an immense study on the gathering of raw materials, the configuration of the production lines, the conditioning of the product, and even their distribution. At each stage of production and distribution behind each product, a high attention of the decision maker is required in order to fulfill the demand.

More than anything else, I have gained an in-depth understanding of the nature of the research, that it can be cyclical, sometimes messy or even failed. I have learned how to gather information, to define and model a problem, to review the literature, to develop heuristic solution procedures and to evaluate them. All together gave me a life long experience in both my academic and personal life.


[156] Schnell, A. and Hartl, R. F. (2017). On the generalization of constraint programming and boolean satisfiability solving techniques to schedule a resource-


Jobish was born in Mananthavady, a village in India state of Kerala, on April 16, 1988. He is the second son of Devasiya Vallikavunkal Mathai and Annamma. He graduated with a bachelor degree of science in mathematics from Kannur University in 2009. He joined afterward the National Institute of Technology, Calicut and graduated with a master degree in science and technology in mathematics and scientific computing in 2012. In his master studies, his work focused on topological structures in the context of soft sets. He published research articles in international journals and had a growing interest for research. After his master studies, he joined the Vimal Jyothi Engineering College as assistant professor. In 2014, he joined the Universidad Autónoma de Nuevo León, Mexico for his doctoral studies.