

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**CONSTRAINT PROGRAMMING-BASED
SOLUTION APPROACH FOR THE FLEXIBLE
JOB SHOP SCHEDULING PROBLEM WITH LOT
STREAMING**

by

Pınar YUNUSOĞLU

July, 2021

İZMİR

**CONSTRAINT PROGRAMMING-BASED
SOLUTION APPROACH FOR THE FLEXIBLE
JOB SHOP SCHEDULING PROBLEM WITH LOT
STREAMING**

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science
in Industrial Engineering, Industrial Engineering Program**

**by
Pınar YUNUSOĞLU**

July, 2021

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “**CONSTRAINT PROGRAMMING-BASED SOLUTION APPROACH FOR THE FLEXIBLE JOB SHOP SCHEDULING PROBLEM WITH LOT STREAMING**” completed by **PINAR YUNUSOĞLU** under supervision of **PROF. DR. ŞEYDA AYŞE YILDIZ** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Şeyda Ayşe YILDIZ

Supervisor

Assoc. Prof. Dr. Şener AKPINAR

(Examining Committee Member)

Asst. Prof. Dr. Emrah Bünyamin EDİS

(Examining Committee Member)

Prof. Dr. Özgür ÖZÇELİK

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Prof. Dr. Şeyda Ayşe Yıldız for her professional guidance, patience, and understanding. I am grateful for her valuable advice in conducting this research. Her feedbacks have helped me to further improve my academic skills. I am thankful for her contribution towards the completion of this thesis.

I would like to express my special thanks to the professors and colleagues from Dokuz Eylül University and İzmir Bakırçay University for their support and tolerance. It is a pleasure to work with them.

I would like to express my special thanks to my parents Hülya Yunusoğlu and Kemal Yunusoğlu, and my sister Gonca Avcı for their support and understanding during the challenging stages of my thesis. They have always encouraged me to complete this work. The motivation they give me is the most important treasure to overcome the challenges in this thesis.

Pınar YUNUSOĞLU

CONSTRAINT PROGRAMMING-BASED SOLUTION APPROACH FOR THE FLEXIBLE JOB SHOP SCHEDULING PROBLEM WITH LOT STREAMING

ABSTRACT

This thesis studies the Flexible Job Shop Scheduling and Lot Streaming Problem (FJSSP-LS) with additional manufacturing resources other than machines. The related work in the literature reveals the influence of lot streaming for time-based objective functions. However, no study simultaneously considers the setup and transport resource restrictions when making scheduling decisions, which eventually limits the applicability of the resulting schedule in practice. Therefore, this thesis considers the FJSSP-LS with setup and transport activities and specific resources required by them. The aim is to achieve a schedule that minimizes the makespan. This problem is among the most constrained scheduling problems. Therefore, we propose a novel Constraint Programming (CP) model enriched by an efficient variable and value ordering strategy specifically designed for the problem. We also present a CP-based iterative improvement method, CP-based Large Neighborhood Search (CP-based LNS), that focuses on exploring the large neighborhoods through the CP model. We generate small-, medium-, and large-sized instances for the problem under study. The computational study shows that the proposed CP-based LNS improves the objective function value for the medium- and large-sized instances by 4.68 percent on average compared to the CP model.

Keywords: Lot streaming, flexible job shop scheduling, constraint programming, large neighborhood search

PARTİ BÖLMELİ ESNEK ATÖLYE ÇİZELGELEME PROBLEMİ İÇİN KISIT PROGRAMLAMA TABANLI ÇÖZÜM YAKLAŞIMI

ÖZ

Bu tez, esnek atölye çizelgeleme ve parti bölme problemini makineler dışındaki ek üretim kaynaklarıyla incelemektedir. Literatürdeki ilgili çalışma, zamana dayalı amaç fonksiyonları için parti bölmenin oldukça etkili olduğunu göstermektedir. Ancak literatürdeki hiçbir çalışma, çizelgeleme kararları verilirken ayar ve taşıma kaynağı kısıtlarını birlikte dikkate almamaktadır. Bu, oluşturulan çizelgenin pratikte uygulanabilirliğini sınırlamaktadır. Bu nedenle, bu çalışma esnek atölye çizelgeleme ve parti bölme problemini ayar ve taşıma faaliyetleri ve bunların gerektirdiği özelleşmiş kaynaklar ile dikkate almaktadır. Amaç, üretim süresini en aza indiren bir çizelge elde etmektir. Bu problem, en kısıtlı çizelgeleme problemleri arasındadır. Bu nedenle, problem için özel olarak tasarlanmış verimli bir değişken ve değer sıralama stratejisi ile zenginleştirilmiş yeni bir kısıt programlama modeli önerilmiştir. Ayrıca, kısıt programlama modeli aracılığıyla büyük komşulukları keşfetmeye odaklanan kısıt programlama tabanlı büyük komşuluk arama sezgiseli yinelemeli bir iyileştirme yöntemi olarak sunulmuştur. Ele alınan problem için küçük, orta ve büyük boyutlu problem örnekleri oluşturulmuştur. Hesaplamalı çalışma, önerilen kısıt programlama tabanlı büyük komşuluk arama sezgiselinin orta ve büyük ölçekli problem örnekleri için amaç fonksiyonu değerini kısıt programlama modeline kıyasla ortalama yüzde 4,68 oranında iyileştirdiğini göstermektedir.

Anahtar kelimeler: Parti bölme, esnek atölye çizelgeleme, kısıt programlama, büyük komşuluk arama sezgiseli

CONTENTS

	Page
M.Sc THESIS EXAMINATION RESULT FORM.....	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT.....	iv
ÖZ	v
LIST OF FIGURES	ix
LIST OF TABLES	x
CHAPTER 1 – INTRODUCTION	1
CHAPTER 2 – PROBLEM DEFINITION.....	4
2.1 Flexible Job Shop Scheduling Problem.....	4
2.2 Lot Streaming Problem	5
2.3 Flexible Job Shop Scheduling Problem with Lot Streaming	8
2.3.1 The Research Gap.....	8
2.3.2 Problem Statement.....	9
CHAPTER 3 – LITERATURE REVIEW	12
3.1 Flexible Job Shop Scheduling Problem.....	12
3.2 Lot Streaming Extension to Job Shop Scheduling Problems	15
CHAPTER 4 – CONSTRAINT PROGRAMMING SOLUTION APPROACH	25

4.1 The Preliminaries of CP	25
4.2 Proposed Constraint Programming Model	28
4.2.1 Notation	28
4.2.2 Objective Function and the Objective Function Related Constraint	29
4.2.3 Constraints Related to the Lot Streaming Decisions	30
4.2.4 Constraints Related to the Presence Status of the Operations	32
4.2.5 Constraints Related to the Allocation of Sublots to the Machines	32
4.2.6 Constraints Related to the Scheduling of Setup Activities	33
4.2.7 Constraints Related to the Scheduling of Transport Activities	35
4.2.8 Constraint Related to the Successive Operations of a Sublot	36
4.2.9 Constraints Related to the Usage of Renewable Setup and Transport Resources	37
4.3 Proposed Branching Strategies	37
4.3.1 Branching Strategy-1	39
4.3.2 Branching Strategy-2	40
4.3.3 Branching Strategy-3	40
4.3.4 Branching Strategy-4	41
4.3.5 Branching Strategy-5	42
4.3.6 Branching Strategy-6	43
 CHAPTER 5 – CONSTRAINT PROGRAMMING-BASED LARGE NEIGHBORHOOD SEARCH	 44
5.1 Hybridization of CP and Metaheuristics	44
5.2 The Proposed CP-Based LNS	46
5.2.1 Adaptive Parameters of the CP-based LNS	50

5.2.1.1 Adaptive Parameter for Defining the Subproblem Size	50
5.2.1.2 Adaptive Parameter for Time Limitation of the CP Model	52
CHAPTER 6 – COMPUTATIONAL STUDY	53
6.1 Generation of Test Instances	53
6.2 Experimental Analysis for Proposed Branching Strategies and Parameters ...	56
6.2.1 The Comparison of Branching Strategies for the CP Model.....	56
6.2.2 The Parameter Settings for the CP-based LNS.....	61
6.3 Computational Analysis	64
6.3.1 The Effect of Lot Streaming on the FJSSP.....	64
6.3.2 The Performance of the CP-based LNS.....	66
6.3.3 The Comparison of the Solution Approaches.....	68
CHAPTER 7 – CONCLUSION	75
REFERENCES.....	76

LIST OF FIGURES

	Page
Figure 2.1 The FJSSP with four operations and eight machines	4
Figure 2.2 The FJSSP-LS without/with lot streaming	6
Figure 6.1 The main effects plot for S/N ratios	63
Figure 6.2 The Gantt chart for the resulting schedule of MFJS9:S125 without/with lot streaming	65
Figure 6.3 Convergence curves for the data set MFJS2 with four scenarios.....	67



LIST OF TABLES

	Page
Table 3.1 Related literature on the FJSSP with setup and transport activities.....	21
Table 3.2 Related literature on the classical JSSP-LS, assembly JSSP-LS, and FJSSP-LS.....	23
Table 4.1 The C# code for the branching strategy-1.....	39
Table 4.2 The C# code for the branching strategy-2.....	40
Table 4.3 The C# code for the branching strategy-3.....	41
Table 4.4 The C# code for the branching strategy-4.....	42
Table 4.5 The C# code for the branching strategy-5.....	42
Table 4.6 The C# code for the branching strategy-6.....	43
Table 5.1 The basic pseudocode of the LNS.....	47
Table 5.2 The notation used in the pseudocode of the CP-based LNS	48
Table 5.3 The pseudocode for the proposed CP-based LNS	49
Table 6.1 Characteristics of Fattahi et al.'s (2007) benchmark instances	54
Table 6.2 Generating conditions for the test instances	56
Table 6.3 Computational results for the test instances using the proposed branching strategies.....	58
Table 6.4 The results of paired t-tests for branching strategies-1, -2, and -3.....	59
Table 6.5 Comparison of the proposed branching strategies according to objective value	60
Table 6.6 The results of paired t-tests for the comparison of branching strategy -1, and branching strategies -4, -5, and -6	60
Table 6.7 The results of paired t-tests for the comparison of branching strategy -4, and branching strategies -4, -5, and -6	61
Table 6.8 Parameter levels for the Taguchi design	61
Table 6.9 The Taguchi design for the computational experiments	63
Table 6.10 Computational results for the small-sized instances	70
Table 6.11 Computational results for the medium-sized instances.....	72
Table 6.12 Computational results for the large-sized instances.....	74

CHAPTER 1

INTRODUCTION

In recent years, manufacturing industries struggle to improve their scheduling processes to meet customer demands on time. As demand for timely production increases, manufacturing industries must be agile to meet customer demands to stay competitive (Cheng, Mukherjee, & Sarin, 2013). For enhancing manufacturing systems' responsiveness, manufacturing processes evolve to process smaller batches that enhance time-based performance measures by making effective use of manufacturing resources. Flexibility is becoming popular in manufacturing industries for enhancing the effective use of manufacturing resources, which eventually leads to increased productivity. Therefore, to improve the benefits of small-batch production, manufacturing systems have to be supported with flexible resources, i.e., multi-purpose machines (Brucker & Schlie, 1990), which are assumed to be primary resources in flexible manufacturing systems. At the same time, the use of small batches complicates the scheduling problems with improved machine flexibility, and scheduling decisions and management of resources become difficult in such systems.

This thesis studies the lot streaming problem in a flexible job shop manufacturing system with multi-purpose machines. The routing of jobs between these machines becomes more difficult since there is more than one machine capable of handling a job. This flexibility in the routing of jobs increases the complexity of the problem. However, lot streaming enhanced by routing flexibility in a job shop manufacturing environment strengthens the responsiveness of the manufacturing system. With the broad applicability of lot streaming, the management of manufacturing resources has become critical due to the increasing number of sublots in the manufacturing environment. With the increased number of sublots, the number of setup and transport activities and the resources required for these activities also increase. Therefore, in practice, lot streaming may lead to congestion in the manufacturing environment due to the increasing demand for setup and transport resources. In order to benefit from lot streaming, the setup and transport activities must be counted, and scheduling additional resources for setup and transport activities other than machines must also

be taken into account to derive a viable schedule in implementation. The related literature on the Flexible Job Shop Scheduling Problem (FJSSP) does not consider the specific resources for setup and transport activities simultaneously. Some studies consider setup resources (Chan, Wong, & Chan, 2006) and some studies consider transport resources (Dai, Tang, Giret, & Salido, 2019; Du, Li, Luo, & Meng, 2021; Nouri, Driss, & Ghédira, 2016; Yan et al., 2021; Zhang, Manier, & Manier, 2012) separately. The related literature on the Flexible Job Shop Scheduling Problem with Lot Streaming (FJSSP-LS) also does not consider any specific resource for the setup or transport activities. Therefore, in this thesis, we investigate the FJSSP-LS under setup and transport resource restrictions simultaneously.

The abovementioned problem is among the most complicated scheduling problems encountered in manufacturing industries. The problem is strongly NP-hard in nature, like the Job Shop Scheduling Problem (JSSP) shown to be NP-hard (Garey & Johnson, 1976). The motivation of this thesis is to develop CP-based solution methodologies for solving the highly constrained FJSSP-LS with setup and transport resources. The contributions of this thesis are to develop a novel Constraint Programming (CP) model with efficient branching strategies and propose a CP-based LNS as an iterative improvement method for solving this problem. The CP-based LNS is effective for solving highly constrained optimization problems that have large feasible solution spaces. The proposed CP-based LNS effectively addresses the complexity of the problem, and the adaptive parameters used in the algorithm tackle the computational difficulty of the solution approach, which originates from the lot streaming problem. The proposed CP-based LNS outperforms the CP model by 4.68 percent on average.

The rest of the thesis is structured as follows. In Chapter 2, the JSSP and FJSSP are first given, and the main concepts of the lot streaming problem are summarized. Next, the research gap is identified, and the problem statement is made afterward. Chapter 3 comprehensively reviews the literature for the FJSSP, Job Shop Scheduling Problem with Lot Streaming (JSSP-LS), assembly JSSP-LS, and FJSSP-LS. In Chapter 4, the CP preliminaries are first given, then the proposed CP model and branching strategies are explained in detail. In Chapter 5, first, information about the hybridization of CP

and metaheuristics is given, then the proposed CP-based LNS and its adaptive parameters are explained in detail. In Chapter 6, the computational results of the experimental studies concerning the parameter tuning and selection of effective branching strategies and a comparison of the solution methodologies are given. In Chapter 7, we give the concluding remarks and future directions.



CHAPTER 2

PROBLEM DEFINITION

2.1 Flexible Job Shop Scheduling Problem

The JSSP is among the most challenging scheduling problems encountered in manufacturing industries. The JSSP has proved to be NP-hard (Garey & Johnson, 1976). This problem arises in a shop manufacturing environment where the jobs have a fixed order of operations to be processed on machines. Given the strict order of operations and the set of machines, the aim is to find a feasible schedule that optimizes a predefined performance measure. The common performance measures for the JSSP are makespan, machine workload, total tardiness, and mean flow time minimization (Chaudhry & Khan, 2016).

The FJSSP is firstly stated in Brucker & Schlie (1990). They extend the JSSP by suggesting multi-purpose machines. These machines can process more than one type of operation, whereas, in the classical JSSP, each machine can process only one type of operation. The processing times of the operations may differ on each capable machine, and the machines can process only one operation at a time. Thus, the scheduling process is more complex compared to the classical JSSP. As Sun, Zhang, Lu, and Zhang (2021) indicated, the FJSSP is also NP-hard. An illustrative figure for the FJSSP is given in Figure 2.1, where the dashed lines show the capable machines for each operation.

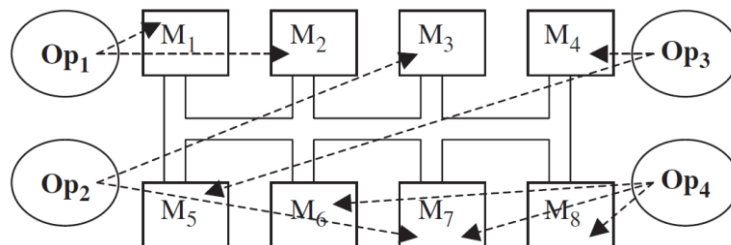
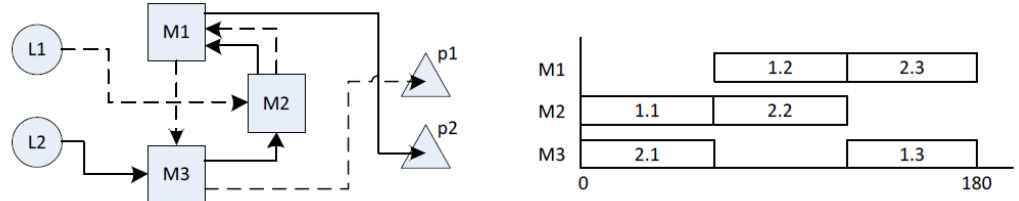


Figure 2.1 The FJSSP with four operations and eight machines (Chan et al., 2006)

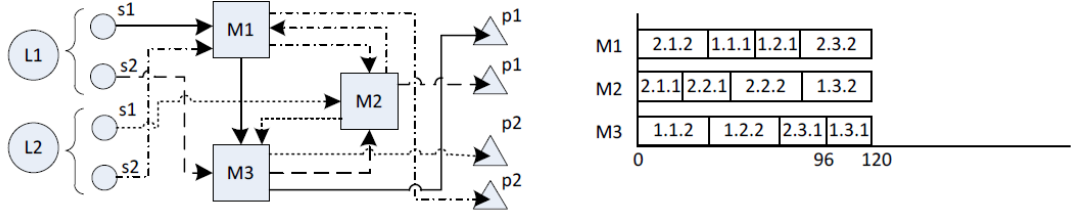
The FJSSP problem consists of two subproblems: *Routing Problem* and *Sequencing Problem*. The routing problem is the allocation of operations on one of the alternative machines, and the sequencing problem is to determine the order of operations allocated to each machine. With improved machine flexibility, the routing problem complicates the FJSSP. In this context, Kacem, Hammadi, and Borne (2002) classify the FJSSP as total or partial flexible, based on the number of operations that can be processed on each machine. In the total flexible JSSP, all machines can handle all operations, while in the partial flexible JSSP, some machines can handle some operations. In this thesis, we study the partial flexible FJSSP and extend it with lot streaming.

2.2 Lot Streaming Problem

Lot streaming is a sophisticated technique of splitting lots into smaller sublots to increase the continuity of production flow by permitting overlapping of successive operations of sublots from the same lot. More precisely, this technique permits the simultaneous processing of sublots in different stages and is used in scheduling problems with different machine configurations such as flow shops, hybrid flow shops, job shops, open shops, parallel machines, and assembly systems (Cheng et al., 2013). Employing the lot streaming, a significant reduction can be made in time-based performance measures such as cycle time, mean flow time, and makespan, prevalent in scheduling problems. In addition, a significant reduction in some cost-based measures, such as work-in-process inventory and total cost, may also be observed due to the accelerated production process (Cheng et al., 2013). An illustrative figure is given in Figure 2.2 to show the reduction achieved in makespan when the FJSSP is extended with lot streaming in Novas (2019).



(a) A FJS processing two lots without lot splitting



(b) The same FJS processing two lots with lot streaming

Figure 2.2 The FJSSP-LS without/with lot streaming (Novas, 2019)

Henceforth, a brief review of the lot streaming problem, including basic concepts, terms, and features, will be presented. In the lot streaming problem, each lot is considered a batch of identical products that can be divided into a predetermined number of sublots, namely transfer batches. In this context, Božek and Werner (2018) define the sublot as the “smallest indivisible and uninterruptible processing unit.” Accordingly, sublots may consist of a real or integer number of products concerning the product characteristics. Those whose sizes consist of real numbers are referred to as continuous sublots, while the others whose sizes consist of integer numbers are referred to as discrete sublots.

Moreover, there are three types of sublots: *equal* sublots, *consistent* sublots, and *variable* sublots. The equal sublots are the consistent sublots that are identical in size. These sublots cannot differ in size, and they remain consistent over the processing stages. The consistent sublots are the sublots that remain the same throughout the manufacturing processes. Therefore, once the sublot sizes are determined, their sizes cannot vary between successive manufacturing processes. On the other hand, the size of variable sublots may differ between successive manufacturing processes, and there are no restrictions on the size of these sublots.

There are some lot streaming policies that can be investigated in the scheduling problems. The no-idling/intermittent-idling feature of sublots can be considered in the lot streaming problem. Intermittent idling permits idle time between sublots scheduled on the same machine. However, as Chang and Chiu (2005) stated, better makespan values can be achieved using sublots with the no-idling feature than those with the intermittent-idling feature. The wait policy permits the delay of a subplot between its operations on different stages or machines, while the no-wait policy forces an immediate transport of a subplot to the next operation to be processed on a different stage or machine without any delay. The intermingling/non-intermingling feature of sublots is encountered in a multi-product manufacturing environment. When intermingling is allowed in the lot streaming problem, the sequence of the sublots on a machine can be interrupted by the sublots belonging to a different lot and scheduled on the same machine.

Furthermore, the attached/detached nature of the setups can be considered in the lot streaming problems. When the setup depends on the availability of a subplot, subplot-attached setups are required; otherwise, subplot-detached setups are considered. In the subplot-attached setups, the setup activity cannot begin unless the subplot has arrived at the machine. The attached/detached setups are correspondingly referred to as non-anticipatory and anticipatory setups in the literature.

Existing studies in the literature have emphasized the applicability of lot streaming in single- or multi-product manufacturing environments under time-based or cost-based objective functions. Accordingly, this thesis considers applying lot streaming for a multi-product job shop environment in which the configuration of machines is flexible. The intermingling policy is allowed between successive operations of a subplot assigned to the same machine. More detailed information regarding lot streaming can be found in (Chang & Chiu, 2005; Cheng et al., 2013).

2.3 Flexible Job Shop Scheduling Problem with Lot Streaming

Extending the lot streaming policy to the FJSSP can improve scheduling performance measures such as makespan, machine workload, and machine utilization (Chang & Chiu, 2005; Cheng et al., 2013). Furthermore, while flexible machines are capable of performing different operations, lot streaming permits the simultaneous processing of operations on these machines. Hence, machine flexibility utilizes the benefits of lot streaming. In addition, operations are scheduled to overlap, further improving the utilization of machines, which are considered a vital manufacturing resource in job shops. Based on these advantages, it is decided that the lot streaming problem in a flexible job shop environment must be tackled in this thesis.

2.3.1 The Research Gap

The existing studies on the FJSSP-LS show that the lot streaming problem handled with machine flexibility is effective in terms of the makespan objective. However, in real-world settings, lot streaming increases the number of setups and transport activities in the manufacturing environment. The FJSSP-LS addressed without considering setup and transport activities limits the applicability and implementation of the resulting schedule and may cause problems in the production process as the setup and transport resources are limited in practice. Therefore, lot streaming decisions must be made wisely, considering the resource restrictions encountered in real-world manufacturing environments. As a result, scheduling setup and transport activities must also be considered when lot streaming is applied.

The existing research on the JSSP-LS mainly considers the machines as a unary resource. However, additional resources are required to perform setup and transport activities in the job shop manufacturing system. Better performance measures can be achieved in a scheduling problem extended with lot streaming. However, lot streaming causes an increase in the total number of setup and transport activities required for the sublots and the demand for the setup and transport resources. There is scarce literature

on the JSSP-LS with sequence-dependent setup and transport activities; please refer to (Bożek & Werner, 2018; Bożek & Wysocki, 2016). However, these studies do not consider the restricted amount of setup and transport resources. Therefore, this thesis aims to fill this research gap and investigate the effect of setup and transport resources in lot streaming problems in flexible job shops.

2.3.2 Problem Statement

The abovementioned problem is entitled as the FJSSP-LS under limited setup and transport resources. The problem is stated as follows. There is a set of machines $M = \{1, 2, \dots, |M|\}$ where $|M|$ denotes the number of machines and a set of operations $O_j = \{1, 2, \dots, |O_j|\}$ to be executed for each job j where $|O_j|$ represents the number of required operations for job j . The manufacturing route for each job is given as $(o_1, o_2, \dots, o_{|O_j|})$ in advance, and the operations can be executed by any of the machines from the eligible machine set $M_{j,o}$ given for each operation of a job. The jobs are defined as a batch of products (lots) that can be divided into smaller sublots. The maximum possible number of sublots that can be formed for a job is determined by the S_j parameter where $|S_j|$ is an upper bound on the possible number of sublots for job j . The subplot sizes are *discrete*; in other words, each subplot consists of an integer number of products. The sublots are *consistent* in size for each operation performed on the given manufacturing route. Furthermore, intermingling is allowed between the successive operations of sublots belonging to the same lot and scheduled on the same machine.

There are setup operations between each pair of subplot coming from a different job and scheduled on the same machine. The setup times depend on the sequence, which may differ concerning the previous subplot and the new arrival subplot on the same machine. The subplot-attached setups are considered in the problem. The setup activities are non-anticipatory, which means that the sublots must be available on a machine to start the setup activity. The setup operators are needed to perform a set of

setup tasks, including preparing the cutter and drillers on the tool holders, changing the fixture and tool holders, cleaning the machines, and setting the machine configurations according to the product specifications. Since setup operators are skilled in performing these setup tasks, they cannot be assigned to any other processing operation during the manufacturing process. It is assumed that there are sufficient workers available to perform the processing operations. Therefore, the setup operators are in charge of conducting only the setup operations. In practice, setup operations are performed by more than one operator in order to complete a set of tasks during the setup activity. Therefore, limited setup operators are allocated to setup activities; it is not easy to plan the setup activities to overlaps on different machines. There are some papers considering the limited setup resources, i.e., operators as in our problem. Please refer to (Fanjul-Peyro, 2020; Yepes-Borrero, Perea, Ruiz, & Villa, 2021; Yepes-Borrero, Villa, Perea, & Caballero-Villalobos, 2020). These studies consider limited setup resources in parallel machine scheduling problems.

There are also transport operations of sublots among the machines. Transport times are independent of the subplot size, and the transport resources are needed to carry the sublots between the machines. A transport resource can be allocated to only one subplot at a time. Since there are limited transport resources available, it is critical to schedule the transport operations considering the resource feasibility. Some papers consider the limited transport resources such as robots and transporters as in our problem. Please refer to (Nouri et al., 2016; Zhang et al., 2012). These studies consider limited transport resources in the FJSSP problems.

There are five decisions to be made in this problem: *i.* Determination of the number of sublots for each job, *ii.* Determination of the subplot sizes, *iii.* Determination of subplot-machine assignments considering the alternative machines, *iv.* Determination of the processing operation sequences on each machine, *v.* Determination of the setup and transport operation sequences on each resource. The basic assumptions of the problem are given below.

- There is a predetermined processing sequence for each job.

- All jobs, machines, setup, and transport resources are available at the beginning of the scheduling.
- The size of the jobs is known in advance. Therefore, product demand is known, and lot sizing is already done.
- Each machine is allocated to only one processing operation at a time.
- Each setup/transport resource is allocated to only one setup/transport operation at a time.
- Each transport resource can be used to transfer only one subplot at a time.
- Empty moves and unavailable times of transport resources are not considered.
- Input and output buffers are not considered (unlimited input/output buffers).
- Preemption is not allowed.
- Machine and vehicle breakdowns are ignored.
- Maintenance activities are not considered.

The optimization objective is defined as minimizing the maximum completion time for all sublots, i.e., the makespan, widely studied in the lot streaming literature. This form of the objective function is usually used in lot streaming problems to observe the time reduction obtained by lot streaming. Note that the FJSSP-LS with setup and transport resources dramatically increases the problem complexity. This variation of the FJSSP is more difficult to solve, so it turns out to be NP-hard in a strong sense.

CHAPTER 3

LITERATURE REVIEW

Since this thesis contributes to both the FJSSP and JSSP-LS literature, a review of the related literature is given under two headings. First, the FJSSP literature is reviewed under the problem characteristics in Section 3.1. Second, the related literature on the JSSP-LS, assembly JSSP-LS, and FJSSP-LS is given in Section 3.2. We also highlight the contributions of our thesis given the existing literature.

3.1 Flexible Job Shop Scheduling Problem

The FJSSP is an extension of the JSSP, where multi-purpose machines are adopted in job shop manufacturing environments. In the FJSSP, setup and transport activities and manufacturing resources required for these activities must be considered to reflect real-world manufacturing environments. Therefore, this thesis considers scheduling these activities and jobs together and investigates their effect in a flexible job shop environment. Accordingly, the existing literature on the FJSSP is reviewed subject to these problem characteristics. The related literature on the FJSSP with setup and transport activities is given in Table 3.1.

There are some studies that consider setup and transport times separately in the FJSSP. Chan et al. (2006) consider operation-dependent setup times in the resource-constrained FJSSP. They propose a genetic algorithm (GA) based solution approach for solving the problem. Saidi-Mehrabad and Fattahi (2007) develop a two-phase tabu search (TS) algorithm for solving the FJSSP with sequence-dependent setup times (SDSTs) to minimize the makespan of the resulting schedule. Defersha and Chen (2010) study the FJSSP with SDSTs, attached and detached setups, time lags, and machine release dates. They propose a mixed integer linear programming (MILP) model and a parallel GA to minimize makespan. Bagheri and Zandieh (2011) deal with the bi-criteria FJSSP with an aggregate objective, including minimizing makespan and mean tardiness. They consider SDSTs and propose a variable neighborhood search

(VNS) algorithm for solving the considered problem. Özgüven, Yavuz, and Özbakır (2012) consider process plan flexibility and separable/non-separable SDSTs in the FJSSP. They propose two mixed integer goal programming models, with the first goal to minimize makespan and the second goal to minimize total deviation from machines' workload from the most loaded machine. Mousakhani (2013) addresses the FJSSP with SDSTs to minimize total tardiness. They propose a MILP model and iterated local search (ILS) based algorithm to solve the problem. Abdelmaguid (2015) considers separable SDSTs in the FJSSP. They suggest a neighborhood search function for the TS algorithm to minimize makespan. Karimi, Ardalan, Naderi, and Mohammadi (2017) consider transport times in the FJSSP in which the number of available transporters is assumed to be unlimited. They propose sequence- and position-based MILP models and a hybrid imperialist competitive algorithm (ICA) with simulated annealing (SA) and local search. Shen, Dauzère-Pérès, and Neufeld (2018) consider SDSTs in the FJSSP. They propose a mathematical model and a TS algorithm with a disjunctive graph representation. Kress, Müller, and Nossack (2019) handle a real-world worker-constrained FJSSP with SDSTs. They propose decomposition-based solution approaches with two objectives: minimizing makespan and minimizing total tardiness. Defersha and Rooyani (2020) consider the model presented by Defersha and Chen (2010), and they propose a two-stage GA in which the first stage involves an encoding scheme for sequencing decisions and the second stage is a common procedure for GA. Tamssaouet, Dauzère-Pérès, Knopp, Bitar, and Yugma (2021) consider the FJSSP with SDSTs. They develop a greedy randomized adaptive search procedure and SA algorithm to optimize multiple criteria defined in the industrial context.

There are few studies that address both setup and transport activities in the FJSSP literature. However, these studies do not consider setup and transport resource restrictions; they only consider setup and transport times when making scheduling decisions. Zhang, Wu, Peng, and Jia (2019) consider setup and transport times in the FJSSP. They develop non-dominated sorting genetic algorithm-II to minimize makespan and total energy consumption. Zhang, Hu, Sun, and Zhang (2020) propose an improved GA to solve the JSSP with job-, operation-, and machine-dependent setup

times and transport times to minimize makespan and total setup and transport time. Li et al. (2020) extend the FJSSP with SDSTs and transport times. They propose an integer programming (IP) model that considers the machines' energy consumption and defines multiple objectives, including makespan and energy consumption minimization. Moreover, they develop an improved Jaya algorithm with effective local search operators. Rossi (2014) handles the FJSSP with separable (anticipatory) SDSTs and transport times. They represent the problem with a disjunctive graph model and suggest an ant colony algorithm (ACO) that minimizes makespan. Motivated by the steel furniture production in a job shop setting, Sun et al. (2021) address a multi-objective FJSSP considering SDSTs and transport times. They develop a hybrid evolutionary algorithm and use TS for the local search of the algorithm. Li and Lei (2021) consider energy-efficient FJSSP with SDSTs and transport times. They propose an ICA to minimize the makespan, total tardiness, and total energy consumption simultaneously.

There is scarce literature on the FJSSP that considers limited transport resources. Nouri et al. (2016) study the FJSSP with several transport robots in a robotic cell environment. They develop a neighborhood-based GA to solve related job scheduling and robot routing problems. Dai et al. (2019) develop a multi-objective GA to solve the FJSSP considering machine tool setup times and automated guided vehicle (AGV) transport constraints, aiming to minimize makespan and total energy consumption. Du et al. (2021) address the distributed FJSSP with crane transport constraints. In this problem, a crane is used to transport the jobs between machines, and there is only one crane available in each factory. They propose a hybrid algorithm based on VNS and estimation of distribution algorithm to minimize total energy consumption and makespan. Zhang et al. (2012) develop a hybrid metaheuristic approach based on GA and TS to solve the FJSSP with identical transport resources. They consider bounded processing times, and the aim is to minimize makespan and total storage induced by input/output buffers for all machines. Yan et al. (2021) develop a GA for solving the FSSP with AGV constraints. They consider job- and operation-dependent setup times and transport times and analyze the coupling relationship between transport and processing operations.

3.2 Lot Streaming Extension to Job Shop Scheduling Problems

In recent years, there is a growing interest in applying lot streaming to multistage processing environments. In the existing literature, the lot streaming is extended to the classical JSSP, assembly JSSP, and FJSSP. Henceforth, the application of lot streaming to these three scheduling problems is reviewed in this section. The related literature on the classical JSSP-LS, FJSSP-LS, and assembly JSSP-LS is given in Table 3.2.

Some studies consider the JSSP-LS without the setup and transport resource restrictions. Dauzere-Peres and Lasserre (1993) propose an iterative procedure for the lot streaming problem in a job shop environment. The procedure consists of two phases. The first phase is referred to as the upper level, which determines the lot streaming decisions, while the second phase is referred to as the lower level, which determines the scheduling decisions to minimize makespan. In the first phase, they calculate the optimal sizes of sublots by using a linear programming (LP) model. In the second phase, the classical JSSP is solved based on the subplot sizes determined by the LP model. Glass, Gupta, and Potts (1994) analyze the lot streaming problem in three-stage production processes: flow shop, job shop, and open shop. The consistent subplot type is adopted for scheduling a single product type in these manufacturing environments. They propose a similar procedure presented in Dauzere-Peres and Lasserre (1993). They divide the problem into two phases that deal with the lot streaming and scheduling decisions separately. In the first phase, they solve an LP model to determine the subplot sizes. Based on the subplot sizes, the classical JSSP is solved in the second phase to determine the sequencing decisions. Dauzere-Peres and Lasserre (1997) propose an iterative procedure based on LP for solving the lot streaming problem with SDSTs in job shops. They suggest an iterative solution approach consisting of two phases: In the first phase, an LP model is solved to determine optimal subplot sizes. Based on the fixed subplot sizes, the job shop scheduling problem is solved in the second phase. Buscher and Shen (2009) propose an integrated TS algorithm for solving the lot streaming problem in a job shop environment. They propose a three-phase algorithm that handles the computational difficulty of the

problem and enhances its ability of convergence to the lower bounds. The three-phase algorithm consists of the following phases: 1. Pre-determination of subplot sizes, 2. Determination of the schedules, 3. Variation of subplot sizes. In the first phase, initially, they consider consistent sublots with equal sizes and then let them deviate from their equal sizes by using a set of formulas. In the second phase, the TS algorithm determines the schedule based on the subplot sizes previously determined in the first phase. In the third phase, the subplot sizes are varied using the proposed variation procedure. In this procedure, two types of variations are considered: 1. The major variation is used for the sublots belonging to different jobs, 2. The minor variation is used to preserve the number of products included in each job. This variation is subject to the sublots belonging to the same jobs. Note that the minor variation can be less than or equal to the major variation. After the third phase, the makespan of the schedule is recalculated by using the proposed TS procedure in the second phase. If a better makespan is achieved, the process continues with these current varied sizes of sublots. Kesen and Güngör (2012) handle the JSSP-LS in virtual manufacturing cells. They consider consistent subplot sizes and propose a MILP model and GA to minimize the makespan of the schedule. Wong and Ngan (2013) consider the assembly JSSP-LS with job- and operation-dependent setup times and consistent sublots and develop a hybrid GA and particle swarm optimization (PSO) to minimize makespan. Liu, Chen, and Lin (2013) study the lot streaming problem in a job shop consisting of multiple jobs whose values exponentially deteriorate over time, and the objective is to maximize the total value of the jobs. They consider different subplot sizing policies, including equal size and varied size sublots. A GA-based job splitting approach, a fixed-number job-splitting approach, and simple dispatching rules (SDRs) are proposed to solve this problem. In Güçdemir and Selim (2018), the customer-oriented JSSP-LS with consistent and equal sublots is handled. A solution approach based on simulation and multi-criteria decision making to evaluate the performance measures, including makespan, mean flow time, percentage of tardy orders, and mean percentage deviation from the customer expectations in their study. Daneshamooz, Fattahi, and Hosseini (2021) address the assembly JSSP-LS to minimize makespan. They propose a MILP model for its formulation and develop two-level VNS-based algorithms as a solution approach.

There are some studies that consider the FJSSP-LS without the setup and transport resource restrictions. Defersha and Chen (2012) investigate the FJSSP-LS with unequal sublots, SDSTs, attached/detached setups, eligible machines, machine release dates, and lag time. They propose a MILP model and an island model parallel GA to minimize makespan. They claim that the extension of lot streaming to the FJSSP is more effective than its extension to the classical JSSP in terms of the makespan objective, which is attributed to the routing flexibility that provides alternative manufacturing routes in a job shop manufacturing system. Furthermore, they remark that minimizing the number of setups or the total setup time should also be an important goal as the lot streaming increases the number of setups in the FJSSP. Božek and Wysocki (2016) consider the FJSSP-LS with variable sublots, transport times, SDSTs, and machine calendars in offline scheduling. A Petri net formalism, TS metaheuristic, and CP model are proposed to minimize makespan. Furthermore, dispatching rules and a multi-agent system are used as two dynamic scheduling methods for the mentioned problem. Defersha and Movahed (2018) develop an LP-assisted GA for the problem handled in Defersha and Chen (2012). They divide the MILP model into two parts by considering the continuous and integer decision variables separately. An LP solves the continuous part of the MILP formulation after the GA has performed a certain number of iterations to refine the promising regions further. Meng and Pan (2018) propose an enhanced migrating birds optimization algorithm for solving the FJSSP-LS in which the objective is to minimize the total flowtime. Božek and Werner (2018) develop a two-stage optimization procedure to solve the FJSSP-LS with variable sublots. They also propose a MILP model, CP model, TS, and a greedy constructive heuristic to solve this problem. SDSTs and transport times are considered, and graph-based models are used to illustrate setup, transport, and processing activities. In the first stage of the optimization procedure, they derive a schedule with the minimum subplot sizes under the makespan minimization objective. In the second stage, the subplot sizes are maximized by preserving the makespan value found in the first stage to reduce transport activities. The second stage has two objectives: maximizing the sum of the subplot sizes of all operations and maximizing the number of non-split operations. This stage aims to determine the lot-sizing decisions by keeping the processing order and the makespan

obtained in the first stage. Novas (2019) propose a CP model for solving the FJSSP-LS to minimize makespan. They consider a shared intermediate storage area and different waiting and storage policies. Furthermore, they investigate different scenarios, including discrete sublots, consistent sublots, single or multiple products, non-idling or intermitted idling situations, wait or no-wait schedules, intermingling or non-intermingling situations, SDSTs or no-setups. Accordingly, four mutually exclusive operational policies are developed and solved using the proposed CP model. Abdollahzadeh Sangroudi and Ranjbar-Bourani (2021) propose a mathematical model and a self-adaptive COA for solving the FJSSP-LS. They consider SDSTs, assembly operations, inventory, and unequal and variable subplot sizes. The objective is to minimize total setup, production, and tardiness penalty costs. Xiuli et al. (2021) develop an improved multi-objective optimization algorithm for solving the FJSSP-LS with variable sublots. They consider multiple objectives including makespan and transferring time minimization. Zhang, Wang, Ma, Li, and Gao (2021) consider variable sublots in the FJSSP-LS. They develop a discrete grey wolf optimizer (DGWO) to minimize the makespan. The proposed DGWO outperforms the GA and PSO which are used as benchmark methods.

A few studies deal with setup resources and their restrictions within the JSSP-LS, assembly JSSP-LS, and FJSSP-LS. Chan, Wong, and Chan (2008) consider consistent sublots and fixture changeovers in an assembly job shop environment. They suggest a solution approach based on dispatching rules and a GA to minimize total lateness and inventory costs. In addition, they compare the performance of their solution method with sublots of equal and varied sizes. Chan, Wong, and Chan (2009b) address the assembly JSSP-LS with consistent sublots. They consider product-, job- and operation-dependent setup times and fixture changeovers and propose solution approaches based on GA and SDRs to minimize lateness cost. Moreover, they evaluate four system measurements: makespan, setup cost, inventory cost, and work-in-process inventory (WIP). Wong, Chan, and Chan (2009) suggest a resource-constrained assembly JSSP by appending an assembly stage to the JSSP-LS. In this problem, jobs are transferred to the assembly stage for final product assembly provided that they have already finished their processes in the JSSP-LS stage. The final assembly of the

product is performed for the jobs belonging to the same bill-of-material structure. Once the number of sublots and their size is determined, they remain unchanged in all the production processes. They also consider fixtures and tools to be renewable resources. The setup times are due to fixture changes and cannot be counted until a fixture change is necessary. Also, the setup times are job- and operation-dependent. Two levels of resources (high and low) are defined for this problem. A high level of resources means that there is no shortage of resources, whereas a low level of resources means that a single resource is available at the resource station. Consequently, two decisions should be taken on the problem: the order of jobs on the machines and assembly operations between jobs. To solve the problem mentioned above, they propose a mathematical model to minimize the total lateness cost of all final products. They determine four system measurements: makespan, total inventory cost, total setup cost, and WIP to interpret the impact of the lot streaming to solutions. Then, they propose a single GA to solve the two subproblems, the lot streaming and scheduling problem, with a new 3D chromosome structure. In addition, they implement the PSO as a benchmark method to compare solutions with the proposed GA. Computational analysis with equal size and varied size sublots is done to analyze the impacts of lot streaming on this scheduling problem. Chan, Wong, and Chan (2009a) handle the JSSP-LS with subplot- and machine-dependent setup times. Fixtures are considered a setup resource, and setups are counted when a fixture changeover is required. In addition, three lot streaming conditions are determined in this problem: without lot streaming, equal size splitting, and varied size splitting. The objective consists of overall penalty cost and total setup cost. The overall penalty cost includes the total cost of lateness, whereas the total setup cost includes the total machining cost during the setup time. Liu (2009) handles the JSSP-LS in an order-based production environment. They extend the lot streaming to the customer order scheduling (COS) problem in a job shop with fixture types as setup resources. A customer order is defined as a collection of various jobs, including more than one product, and called a batch. In a COS problem, the products cannot be delivered until the complete order is produced. In other words, to deliver an order, all sublots of the jobs included in this order must go through a series of operations in a manufacturing route. Accordingly, they define three objectives for this problem: minimization of lateness, finished goods flow time,

and makespan. In the proposed solution procedure, they determine the lot streaming conditions with GA and solve the problem using SDRs. In this context, arrival-, due date-, processing time-, job complexity-, and ratio-oriented dispatching rules are considered.

There is only one study that considers transport resources in the JSSP-LS. Lei and Guo (2013) consider consistent sublots in the JSSP-LS to minimize makespan. In this problem, transport time consists of the time spent in load/unload operations and trips. They assume that transport time for load/unload operations depends on the subplot sizes, and there is a single transporter available to transfer only one subplot at any time between two machines. In this context, to solve the mentioned problem, they propose a modified artificial bee colony algorithm with a two-phase decoding procedure and use discrete event simulation to dispatch the transport tasks after deriving a schedule.

As a result of the comprehensive review of the FJSSP, JSSP-LS, assembly JSSP-LS, and FJSSP-LS literature, it is observed that simultaneous consideration of setup and transport activities is scarce in the related studies. Also, the renewable resources required by setup and transport activities are not investigated simultaneously in the lot streaming problems. The contribution of this thesis to the related literature is to consider both setup and transport resource restrictions in the FJSSP-LS, which are mainly observed in real-world settings. Unlike the existing studies in the literature, lot streaming is investigated under additional resource restrictions apart from processing machines reflecting the real-world flexible job shop manufacturing environments with sequence-dependent setup and transport times. Since this problem is a highly constrained optimization problem reflecting the real-world scheduling restrictions, a novel CP model and CP-based LNS are proposed for its solution for the first time. Tables 3.1 and 3.2 illustrate the contributions of this thesis besides the related FJSSP and FJSSP-LS literature in terms of problem characteristics and solution methodologies.

Table 3.1 Related literature on the FJSSP with setup and transport activities

Author(s)	Problem Characteristics					Solution Methodologies	
	Setup Time	Transport Time	Setup Resource	Transport Resource	Objective	Exact	Heuristic
Chan et al. (2006)	√		√		$\min C_{max}$, MIC		GA
Saidi-Mehrabad and Fattahi (2007)	√				$\min C_{max}$		TS
Defersha and Chen (2010)	√				$\min C_{max}$	MILP Model	Parallel GA
Bagheri and Zandieh (2011)	√				$\min C_{max}$, Mean Tardiness		VNS
Özgüven et al. (2012)	√				$\min C_{max}$, TDMW	MIGP Models	
Zhang et al. (2012)		√		√	$\min C_{max}$, Total Storage		Hybrid Metaheuristic Based on GA and TS
Mousakhani (2013)	√				$\min$ Total Tardiness	MILP Model	ILS-based Algorithm
Rossi (2014)	√	√			$\min C_{max}$		ACO with RPR
Abdelmaguid (2015)	√				$\min C_{max}$		TS
Nouri et al. (2016)		√		√	$\min C_{max}$		Hybrid Metaheuristics Based on CHMM, GA, and TS
Karimi et al. (2017)		√			$\min C_{max}$	MILP Models	HICA with SA and LS
Shen et al. (2018)	√				$\min C_{max}$	MILP Model	TS
Kress et al. (2019)	√				$\min C_{max}$, Total Tardiness		Decomposition-based Solution Approaches
Dai et al. (2019)	√	√		√	$\min C_{max}$, TEC	Multi-Objective MILP Model	GA
Zhang et al. (2019)	√	√			$\min C_{max}$, TEC		NSGA-II
Zhang et al. (2020)	√	√			$\min C_{max}$, Total Setup and Transport Time		GA
Defersha and Rooyani (2020)	√				$\min C_{max}$		GA

Table 3.1 continues

Li et al. (2020)	√	√			min C_{max} , Energy Consumption	IP Model	Improved Jaya Algorithm
Sun et al. (2021)	√	√			min C_{max} , TW, TWCM, and Earliness/Tardiness Penalties		HMEA with TS
Du et al. (2021)		√		√	min C_{max} , TEC		Hybrid Algorithm Based on VNS and EDA
Tamssaouet et al. (2021)	√				min Multiple Criteria		SA, GRASP
Li and Lei (2021)	√	√			min C_{max} , Total Tardiness, and TEC		ICA
Yan et al. (2021)	√	√		√	min C_{max}	MILP Model	GA
This paper*	√	√	√	√	min C_{max}	CP Model	CP-based LNS

Abbreviations: ACO: Ant Colony Algorithm, AGV: Automated Guided Vehicle, CHMM: Clustered Holonic Multiagent Model, C_{max} : Makespan, EDA: Estimation of Distribution Algorithm, GA: Genetic Algorithm, HICA: Hybrid Imperialist Competitive Algorithm, HMEA: Hybrid Many-Objective Evolutionary Algorithm, ICA: Imperialist Competitive Algorithm, ILS: Iterated Local Search, IP: Integer Programming, LS: Local Search, MIC: Machine Idle Cost, MIGP: Mixed Integer Goal Programming, MILP: Mixed-Integer Linear Programming, NSGA-II: Non-Dominated Sorting Genetic Algorithm-II, RPR: Reinforced Pheromone Relationships, SA: Simulated Annealing, SDST: Sequence-Dependent Setup Time, TDMW: Total Deviation of the Machines' Workload, TEC: Total Energy Consumption, TS: Tabu Search, TW: Total Workload, TWCM: Total Workload of the Critical Machine, VNS: Variable Neighborhood Search.

Table 3.2 Related literature on the classical JSSP-LS, assembly JSSP-LS, and FJSSP-LS

Author(s)	Problem	Problem Characteristics							Solution Methodologies
		Consistent	Variable	Setup Time	Transport Time	Setup Resource	Transport Resource	Objective	
Dauzere-Peres and Lasserre (1993)	JSSP-LS	√						min C_{max}	Iterative Procedure Based on LP
Glass et al. (1994)	JSSP-LS, FSSP-LS, OSSP-LS	√						min C_{max}	Algorithms Based on LP Model
Dauzere-Peres and Lasserre (1997)	JSSP-LS	√		√				min C_{max}	Iterative Procedure Based on LP
Chan et al. (2008)	AJSSP-LS	√		√		√		min LC, IC	An Algorithm Based on GA and SDR
Liu (2009)	JSSP-LS & COS	√		√		√		min C_{max} FGFT, Lateness	GA, SDR
Chan et al. (2009a)	JSSP-LS	√		√		√		min OPC, TSC	GA
Buscher and Shen (2009)	JSSP-LS	√						min C_{max}	Three-phase Algorithm Based on TS
Chan et al. (2009b)	AJSSP-LS	√		√		√		min Lateness Cost	GA, SDR
Wong et al. (2009)	AJSP-LS	√		√		√		min TLC	MILP Model, GA, PSO
Defersha and Chen (2012)	FJSSP-LS	√		√				min C_{max}	MILP Model, Island-model Parallel GA
Kesen and Güngör (2012)	JSSP-LS	√						min C_{max}	MILP Model, GA
Lei and Guo (2013)	JSSP-LS	√			√		√	min C_{max}	MABC Algorithm

Table 3.2 (continues)

Liu, Chen, and Lin (2013)	JSSP-LS	√		√				max TVJ	GA, SDR
Wong and Ngan (2013)	AJSSP-LS	√		√				min C_{max}	A Hybrid GA, A Hybrid PSO
Bożek and Wysocki (2016)	FJSSP-LS		√	√	√			min C_{max}	Petri Net Formalism, TS, and CP, DR and MAS
Bożek and Werner (2018)	FJSSP-LS		√	√	√			min C_{max}	MILP Model, CP Model, TS
Güçdemir and Selim (2018)	JSSP-LS	√		√				min C_{max} MFT, PTO, MPDCE	Simulation, MCDM
Defersha and Movahed (2018)	FJSSP-LS	√		√				min C_{max}	MILP model, LP-assisted GA
Novas (2019)	FJSSP-LS	√		√				min C_{max}	CP Model
Abdollahzadeh Sangroudi and Ranjbar-Bourani (2021)	FJSSP-LS		√	√				min Total SC, TPC, PC.	MILP Model, Self-adaptive COA
Daneshamooz et al. (2021)	AJSSP-LS		√					min C_{max}	MILP Model, VNS-based Algorithms
Xiuli et al. (2021)	FJSSP-LS		√		√			min C_{max} , TT	IMOA
Zhang et. al. (2021)	FJSSP-LS		√					min C_{max}	DGWO
This paper*	FJSSP-LS	√		√	√	√	√	min C_{max}	CP Model, CP-based LNS

Abbreviations: AJSSP-LS: Assembly Job Shop Scheduling Problem with Lot Streaming, C_{max} : Makespan, COA: Cuckoo Optimization Algorithm, COS: Customer Order Scheduling, CP: Constraint Programming, DGWO: Discrete Grey Wolf Optimizer, DR: Dispatching Rules, FGFT: Finished Goods Flow Time, FJSSP-LS: Flexible Job Shop Scheduling Problem with Lot Streaming, IC: Inventory Cost, IMOA: An Improved Multi-Objective Optimization Algorithm, JSSP-LS: Job Shop Scheduling Problem with Lot Streaming, LC: Lateness Cost, LNS: Large Neighborhood Search, LP: Linear Programming, MABC: A Modified Artificial Bee Colony, MAS: Multi-Agent System, MCDM: Multi-Criteria Decision Making, MFT: Mean Flow Time, MILP: Mixed-Integer Linear Programming, MPDCE: Mean Percentage Deviation from the Customer Expectations, OPC: Overall Penalty (Earliness/Lateness) Cost, OSSP-LS: Open Shop Scheduling Problem with Lot Streaming, PC: Production Cost, PC: Tardiness Penalty Cost, PGA: Parallel Genetic Algorithm, PSO: Particle Swarm Optimization, PTO: Percentage of Tardy Orders, SC: Setup Cost, SDR: Simple Dispatching Rules, SDST: Sequence-dependent Setup Time, TLC: Total Lateness Cost, TS: Tabu Search, TSC: Total Setup Cost, TT: Transferring Time, TVJ: The Total Value of Jobs, VNS: Variable Neighborhood Search.

CHAPTER 4

CONSTRAINT PROGRAMMING SOLUTION APPROACH

Constraint Programming (CP) is an effective solution approach for solving highly constrained optimization problems (Yunusoglu & Topaloglu Yildiz, 2021). The solution approach presents variable and constraint expressions specifically designed for modeling scheduling constraints. The existing studies in the literature indicate that the CP approach is effective for solving constrained scheduling problems (Åstrand, Johansson, & Zanarini, 2020; Kreter, Schutt, Stuckey, & Zimmermann, 2018; Novas, 2019; Topaloglu & Ozkarahan, 2011; Yunusoglu & Topaloglu Yildiz, 2021). In this thesis, it is hard to develop a MILP model reflecting the real-world scheduling restrictions, especially for the resource scheduling decisions exclusively defined for the setup and transport activities. Also, the MILP model may not be effective in finding feasible solutions within strict time limits, since the problem is highly constrained. This problem can only be modeled using specialized scheduling constraints provided by the CP Optimizer. Hence, we proposed a CP model for the FJSSP-LS under limited setup and transport resources. In this context, first, the CP preliminaries will be given in this chapter. Later, the proposed CP model and branching strategies will be explained in detail.

4.1 The Preliminaries of CP

CP is an exact solution approach utilized by practitioners from operations research (OR) communities to solve highly constrained optimization problems. CP uses the contributions introduced by logic programming and artificial intelligence (AI) in a tree-search solution strategy; therefore, it is considered an interdisciplinary solution approach for the CP-AI-OR practitioners (Kacprzyk & Pedrycz, 2015). The tree-search strategy in CP is the branch and bound procedure, which is typically used in integer programming. CP strengthens the branch and bound procedure with effective constraint propagation algorithms. At each node of the branch and bound tree, the modeler uses the inferences of constraint propagation algorithms. These algorithms can capture the infeasibility induced by a value instantiation to a variable.

Constraint propagation is a procedure for solving a *constraint satisfaction problem* (CSP) that aims at finding a feasible solution with a complete assignment of values to variables such that whole problem constraints are satisfied. CSP consists of n decision variables $X = \{x_1, x_2, \dots, x_n\}$, and each variable is associated with a finite domain $D = \{D_1, D_2, \dots, D_n\}$, correspondingly (Rossi, Van Beek, & Walsh, 2006). The CSP is to find the appropriate values from the domain of the variables that are consistent with the given set of constraints. For each variable, each value in the corresponding domain is checked, and *domain reduction* is made (Rossi et al., 2006). The effective propagation algorithms remove the domain values that lead to inconsistencies for the set of constraints. The process continues with the reduced domain of the variables. If any variable with the current domain cannot achieve a feasible solution, CP returns fail and backtracks the previously instantiated variable in the branch and bound tree. The CP branches out the node that assigns a new value from the current domain to instantiate this variable. This procedure is repeated until a complete assignment is attained, implying a feasible solution for the optimization problem. If consistency is ensured with the current assignment that produces a better bound on the objective function, this provides a bound for the objective function of the CP model (Kacprzyk & Pedrycz, 2015).

The variable and value selection orders are important for propagation algorithms (Rossi et al., 2006). The order of variables for branching and the order of values for instantiating variables are decisions made by the CP Optimizer (IBM, 2019a). Employing problem-specific variable and value orderings is crucial to decide on effective branching decisions during the search progress. In this regard, The CP Optimizer enables practitioners to define the *variable and value orderings* based on heuristic criteria. Furthermore, search phases can be defined to guide the tree search. A search phase can contain variable and/or value ordering strategies and search types for traversing the search tree, such as depth-first, iterative diving, restart, multi-point, and auto search. Variable and value orderings, search phases, and search types are the decisions left to the practitioners. Detailed information about the search phases will be given in Section 4.3.

CP has a declarative programming language that includes variable and value definers and constraint expressions on the defined variables. IBM (2019a) presents expressions that are specifically designed to model scheduling constraints. Hence, it is easier to adapt the combinatorial nature of the problem by using these expressions. Besides, it is possible to model the non-linear constraints by the declaration of scheduling constraint expressions. Since there is no assumption of linearity, it is easier to model highly constrained optimization problems than the traditional mathematical modeling approach.

IBM ILOG CP Optimizer 12.10 has been used in this thesis for modeling the proposed CP model. A brief introduction to the related structures on variable and constraint expressions is provided with reference to IBM (2019a) as follows. The basic variable definers consist of integer variables, interval variables, and sequence variables. *Interval Variable* represents a time interval of a scheduled job and has a specific size and has start/end times that are accessible by the user. An interval variable can be considered *optional* when there is no requirement to be present during model execution. *Sequence variable* defines the order of the interval variables as a sequence. In scheduling problems, interval variables are mainly used to model processing activities of jobs and can be used to define the sequence variables to decide on the order of the jobs on each machine. Furthermore, there are specific scheduling expressions based on timing decisions of interval variables such as *startOf*, *endOf*, *sizeOf*, *endAtEnd*, *endAtStart*, *overlapLength*; constraints defined for precedence relations such as *endBeforeStart*, *endBeforeEnd*, *startBeforeEnd*, *startBeforeStart*; constraints to decide on sequence decisions such as *first*, *noOverlap*, *typeOfNext*, *typeOfPrev*, *endOfNext*, *endOfPrev*, *sizeOfNext*, *sizeOfPrev*, *startOfNext*, *startOfPrev*; constraints including group of interval variables such as *span*, *alternative*, *synchronize*; logical constraints such as *presenceOf* and global constraints that include group of variables such as *allDifferent*.

Furthermore, resource scheduling constraints introduced by The CP Optimizer (IBM, 2019a) are effective in modeling manufacturing resources in restricted amounts. These constraints are based on the *cumulFunction* introduced by the CP Optimizer to

model the resources. In addition, there are specific expressions to model renewable resources and resource consumptions, i.e., *pulse*, *step*, *stepAtEnd*, *stepAtStart*, *heightAtEnd*. In this way, the CP Optimizer enables practitioners to model the resource scheduling constraints more compactly. Please refer to more detailed information for the CP Optimizer scheduling constraints and expressions (IBM, 2019a).

4.2 Proposed Constraint Programming Model

4.2.1 Notation

Indices

i, j	Index for jobs
s	Index for sublots
o	Index for operations
m, m'	Index for machines

Sets

$J = \{0, 1, 2, \dots, j, \dots J \}$	Set of jobs including dummy job 0
$M = \{1, 2, \dots, m, \dots M \}$	Set of machines
$S_j = \{1, 2, \dots, s, \dots S_j \}$	Set of sublots for job j where $ S_j $ is an upper bound on the possible number of sublots
$O_j = \{1, 2, \dots, o, \dots O_j \}$	Set of operations for job j
$M_{j,o}$	Set of eligible machines to process operation o of job j , such that $M_{j,o} \subseteq M$

Parameters

js_j	Size of job j
$pt_{j,o,m}$	Unit processing time for operation o of job j , when it is processed on machine m
$st_{i,j,m}$	SDST required to change from processing job i to job j on machine m

$tt_{m,m'}$	Transport time between machine m and m'
sRq_j	Setup resource requirement for the setup activity of job j
tRq_j	Transport resource requirement for job j
$sCap$	Available number of setup resources
$tCap$	Available number of transport resources
L	Sufficiently large number

Integer Decision Variable

$x_{j,s,o}$	Integer variable regarding the size of the sublots belonging to job j for each operation o
-------------	--

Interval Decision Variables

$pOp_{j,s,o}$	Optional interval variable that represents operation o of subplot s belonging to job j
$sOp_{j,s,o}$	Optional interval variable that represents the setup activity of subplot s belonging to operation o of job j
$tOp_{j,s,o}$	Optional interval variable that represents the transport activity of subplot s belonging to operation o of job j
$xM_{j,s,o,m}$	Optional interval variable indicating the $pOp_{j,s,o}$ assigned to machine m where $m \in M_{j,o}$

Sequence Variable

mch_m	Sequence variable that determines the order of the operations $xM_{j,s,o,m}$ assigned to machine m
---------	--

4.2.2 Objective Function and the Objective Function Related Constraint

The objective function and the objective function-related constraint indicate the aim of the optimization problem and the computation of the objective function value.

$$\text{Minimize } C_{max} \quad (2.1)$$

$$C_{max} = \max\{endOf(pOp_{j,s,o}) | \forall j \in J, \forall s \in S_j, o = |O_j|\} \quad (2.2)$$

The objective function (2.1) minimizes the makespan. Constraint (2.2) calculates the makespan, which is the maximum time to complete all operations of the sublots. Here, the *endOf* keyword is used to access the end time of the interval variable $pOp_{j,s,o}$. The end times of all operations are included in a list, and the maximum of them indicates the makespan. Thereby, the form of the objective function is non-linear.

4.2.3 Constraints Related to the Lot Streaming Decisions

The constraints (2.3)-(2.5) related to lot streaming decisions determines the number of sublots to be formed from each job and the size of the consistent sublots subject to the scheduling constraints of the CP model in an integrated manner. These constraints are adapted from Novas (2019). Constraint (2.3) calculates the processing time of the sublots denoted by the size of the $xM_{j,s,o,m}$ interval variable. Whenever a subplot is allocated to a machine, which implies that the related optional interval variable $xM_{j,s,o,m}$ is present in the solution, the processing time of this subplot is calculated by multiplying the unit processing time and the subplot size. Here, the *presenceOf* keyword is used to control the presence status of the $xM_{j,s,o,m}$ optional interval variable. When this variable is present in the solution, it returns “1”; otherwise, it returns “0”. Therefore, this constraint is active only when a subplot is assigned to a machine; in other words, the $xM_{j,s,o,m}$ is present in the solution.

$$sizeOf(xM_{j,s,o,m}) = presenceOf(xM_{j,s,o,m}) \times x_{j,s,o} \times pt_{j,o,m} \quad (2.3)$$

$$\forall j \in J, \forall s \in S_j, \forall o \in O_j, \forall m \in M_{j,o}$$

$$\sum_{s=1}^{S_j} x_{j,s,o} = js_j \quad \forall j \in J, \forall o \in O_j \quad (2.4)$$

$$x_{j,s,o} = x_{j,s,o'} \quad \forall j \in J, \forall s \in S_j, \forall o, o' \in O_j | o \neq o' \quad (2.5)$$

The determination of the subplot sizes is a decision made by the CP model subject to a set of constraints. Constraint (2.4) ensures the consistency between the job size and the size of all sublots belonging to this job. The total size of all sublots created for each operation of a job must be equal to its size. Constraint (2.5) guarantees the consistency of sublots by which the subplot sizes remain the same in all operations throughout the manufacturing process.

4.2.4 Constraints Related to the Scheduling of Dummy Sublots

The constraints (2.6)-(2.7) related to the scheduling of the dummy job are necessary to determine the initial setup time of the first subplot scheduled on the machines. The first subplot scheduled on each machine must be the dummy subplot whose setup time is zero. Note that the dummy job has only one operation that can be processed on all machines.

$$\sum_{s=1}^{S_0} \text{presenceOf}(pOp_{0,s,1}) = |M| \quad (2.6)$$

$$\text{first}(mch_m, xM_{0,s,1,m}) \quad \forall s \in S_0, \forall m \in M \quad (2.7)$$

According to Constraint (2.6), the number of sublots for the dummy job is equal to the number of machines $|M|$ available. Constraint (2.7) enforces the dummy job to take the first place in sequencing sublots assigned to each machine. Therefore, we employ the *first* constraint to determine the first interval variable in a machine sequence. In this way, the dummy subplot operation is scheduled on each machine.

4.2.4 Constraints Related to the Presence Status of the Operations

The constraints (2.8)-(2.9) determine the presence status of the optional interval variables $pOp_{j,s,o}$ associated with the operations of a subplot based on its size. The zero-size sublots are not taken into account because they do not actually exist. Thus, the operations of these sublots should not be included in the solution as their scheduling will lead to improper scheduling of setup and transport activities.

$$x_{j,s,o} \geq \text{presenceOf}(pOp_{j,s,o}) \quad \forall j \in J | j \neq 0, \forall s \in S_j, \forall o \in O_j \quad (2.8)$$

$$\text{presenceOf}(pOp_{j,s,o}) \times L \geq x_{j,s,o} \quad \forall j \in J | j \neq 0, \forall s \in S_j, \forall o \in O_j \quad (2.9)$$

Constraint (2.8) enforces the presence status of the optional interval variables $pOp_{j,s,o}$ that correspond to all operations of the subplot to become *absent* when the value of the integer variable $x_{j,s,o}$ becomes zero during the execution of the CP model. Thus, zero-size sublots are not considered in the scheduling decisions. Constraint (2.9) enforces the presence status of the optional interval variables $pOp_{j,s,o}$ to become *present* when the related subplot size is greater than zero, which means the operations of this subplot must be scheduled.

4.2.5 Constraints Related to the Allocation of Sublots to the Machines

The constraints (2.10)-(2.11) determine the assignment of sublots to one of the eligible machines and also guarantee that a machine as a unary resource can process only one subplot at a time.

$$\text{alternative}(pOp_{j,s,o}, \{xM_{j,s,o,m} | m \in M_{j,o}\}) \quad \forall j \in J, \forall s \in S_j, \forall o \in O_j \quad (2.10)$$

$$\text{noOverlap}(mch_m) \quad m \in M \quad (2.11)$$

Constraint (2.10) assures that each subplot must be assigned to a machine. Here, we employ the *alternative* constraint, which selects an optional $xM_{j,s,o,m}$ interval variable from the set of candidate interval variables to match up with the interval variable $pOp_{j,s,o}$. These two variables have the same start and end values in a feasible schedule. Constraint (2.11) assures that each machine can handle at most one subplot at a time. The machines are unary resources for the subplot operations and cannot process more than one subplot at a time. The *noOverlap* constraint is used to prevent the overlapping of subplot operations, $xM_{j,s,o,m}$ variables on a machine. This constraint is imposed on the sequence variables to derive a non-overlapping schedule of the interval variables involved in the related sequence variable.

4.2.6 Constraints Related to the Scheduling of Setup Activities

The constraints (2.12)-(2.16) are related to the scheduling of setup activities. They determine the presence status of the setup activities and their scheduling decisions, including the subplot-attached setups. The optional variable for the setup activity must be present in the solution when SDST is counted between consecutive sublots assigned to the same machine, and the setup activities must be scheduled between them considering the subplot-attached setups.

$$typeOfPrevious(mch_m, xM_{j,s,o,m}, j, j) \neq j \Rightarrow \quad (2.12)$$

$$presenceOf(sOp_{j,s,o}) = 1$$

$$\forall j \in J, \forall s \in S_j, \forall o \in O_j, \forall m \in M_{j,o}$$

$$presenceOf(xM_{j,s,o,m}) = 1 \Rightarrow$$

$$sizeOf(sOp_{j,s,o}) = presenceOf(sOp_{j,s,o}) \times st_{i,j,m} \quad (2.13)$$

$$\forall i, j \in J | i = typeOfPrevious(mch_m, subplotM_{j,s,o,m}, j, j), \forall s \in S_j, \forall o \in O_j, \forall m \in M_{j,o}$$

$$presenceOf(pOp_{j,s,o}) \geq presenceOf(sOp_{j,s,o}) \quad (2.14)$$

$$\begin{aligned}
& \forall j \in J, \forall s \in S_j, \forall o \in O_j \\
& startOf(sOp_{j,s,o}, L) \geq endOfPrevious(mch_m, xM_{j,s,o,m}, 0) \\
& \forall j \in J, \forall s \in S_j, \forall o \in O_j, \forall m \in M_{j,o}
\end{aligned} \tag{2.15}$$

$$\begin{aligned}
& startOf(pOp_{j,s,o}) \times presenceOf(sOp_{j,s,o}) = endOf(sOp_{j,s,o}) \\
& \forall j \in J, \forall s \in S_j, \forall o \in O_j
\end{aligned} \tag{2.16}$$

Constraint (2.12) determines the presence status of the setup activity. The setup activity must be counted between each pair of sublots from different jobs if scheduled on the same machine successively. In this context, the *typeOfPrevious* keyword is used to detect the sequence-dependent setup activities. This keyword is effective in finding the job type of the previous subplot scheduled on a machine using the related sequence variable. If the previous subplot type is different from the current subplot type, the sequence-dependent setup activity must be present in the solution. Note that this constraint is ineffective when $xM_{j,s,o,m}$ is the first interval variable in the machine sequence or $xM_{j,s,o,m}$ is absent in the current solution. These two cases are captured in the third and fourth inputs of the *typeOfPrevious* keyword. Constraint (2.13) calculates the SDST between consecutively scheduled sublots of different job types scheduled on the same machine. The size of the $sOp_{j,s,o}$ interval variable denotes the time of setup activity for the subplot. Whenever a subplot is allocated to a machine, which implies that the optional interval variable $xM_{j,s,o,m}$ is present in the solution, and the setup activity is present between consecutively scheduled sublots, the size of the $sOp_{j,s,o}$ interval variable corresponds to the SDST between the sublots. Constraint (2.14) links the presence status of the setup activity and subplot operation. When a subplot operation is not present, this constraint sets the absent status to the setup activities planned for this subplot operation. Constraint (2.15) defines the possible start times of the setup activities. The setup activity can only start after the previously scheduled subplot operation finishes on the same machine. Since $sOp_{j,s,o}$ and $xM_{j,s,o,m}$ are the optional interval variables, two cases must be considered to validate this constraint. In the first case, $sOp_{j,s,o}$ may not be present in the solution, so the *startOf* keyword returns a sufficiently large number L to validate the inequality. In the second

case, $xM_{j,s,o,m}$ may be the first interval variable in the sequence mch_m , so the *endOfPrevious* keyword returns zero to validate the inequality. Finally, Constraint (2.16) imposes the attached setup times. The setup activity of the subplot operation must be completed at the same time that the operation starts on the machine.

4.2.7 Constraints Related to the Scheduling of Transport Activities

The constraints (2.17)-(2.21) are related to the scheduling of transport activities. They determine the presence status of the transport activities and their scheduling decisions. The optional variable for the transport activity must be present in the solution when the consecutive operations of a subplot are assigned to different machines, and transport activities must be scheduled between these operations in such a way that the precedence relations defined for transport activities are taken into account.

$$\begin{aligned} presenceOf(xM_{j,s,o,m}) \times presenceOf(xM_{j,s,(o+1),m}) &= 1 \\ \Rightarrow presenceOf(tp_{j,s,o}) &= 0 \end{aligned} \quad (2.17)$$

$$\begin{aligned} \forall j \in J, \forall s \in S_j, \forall o \in O_j | o \neq |O_j|, \forall m \in M_{j,o} \\ presenceOf(xM_{j,s,o,m}) \times presenceOf(xM_{j,s,(o+1),m'}) &= 1 \\ \Rightarrow sizeOf(tp_{j,s,o}) &= tt_{m,m'} \end{aligned} \quad (2.18)$$

$$\begin{aligned} \forall j \in J, \forall s \in S_j, \forall o \in O_j | o \neq |O_j|, \forall m, m' \in M_{j,o} | m \neq m' \\ presenceOf(pOp_{j,s,o}) \geq presenceOf(tp_{j,s,o}) \end{aligned} \quad (2.19)$$

$$\begin{aligned} \forall j \in J, \forall s \in S_j, \forall o \in O_j | o \neq |O_j| \\ endBeforeStart(pOp_{j,s,o}, tp_{j,s,o}) \end{aligned} \quad (2.20)$$

$$\begin{aligned} \forall j \in J, \forall s \in S_j, \forall o \in O_j | o \neq |O_j| \\ endBeforeStart(tp_{j,s,o}, pOp_{j,s,(o+1)}) \end{aligned} \quad (2.21)$$

Constraint (2.17) links the presence status of the transport activity and the consecutive operations of a subplot. If consecutive operations of a subplot are assigned to the same machine, the transport activity must not be counted between these operations. Constraint (2.18) calculates the transport time between the consecutive operations of the subplot assigned to different machines. The size of the $tOp_{j,s,o}$ interval variable denotes the transport time of the subplot. Whenever consecutive operations of a subplot are allocated to different machines, which implies that the optional interval variables $xM_{j,s,o,m}$ and $xM_{j,s,(o+1),m'}$ are present in the solution, the size of the $tOp_{j,s,o}$ interval variable is determined as the transport time between those machines.

Constraint (2.19) links the presence status of the transport activity and the subplot operation. This constraint sets the absent status to the transport activity for the subplot operation that is absent in the solution. Constraints (2.20)-(2.21) are concerned with scheduling the transport activities between the consecutive operations of the sublots. The transport activity can only be scheduled when the previous operation finishes. Moreover, it must be scheduled before the successive operation starts. In this context, we employ the *endBeforeStart* constraint to define the precedence relationships between the transport activity and each of the consecutive operations assigned to different machines.

4.2.8 Constraint Related to the Successive Operations of a Sublot

Constraint (2.22) indicates the precedence relationships between the successive subplot operations of a job. The successive subplot operation can only start when the previous operation has been completed.

$$startOf(pOp_{j,s,(o+1)}) \geq endOf(pOp_{j,s,o}) \quad (2.22)$$

$$\forall j \in J, \forall s \in S_j, \forall o \in O_j | o \neq |O_j|$$

4.2.9 Constraints Related to the Usage of Renewable Setup and Transport Resources

The constraints (2.23)-(2.24) are related to the usage of renewable setup and transport resources. They determine the resource restrictions for setup and transport activities correspondingly. These constraints are required to model the manufacturing resource restrictions. The feasibility of the problem depends on considering these resource restrictions.

$$Cumulative(sOp_{j,s,o} | \forall j \in J, \forall s \in S_j, \forall o \in O_j, sRq_j, sCap) \quad (2.23)$$

$$Cumulative(tOp_{j,s,o} | \forall j \in J, \forall s \in S_j, \forall o \in O_j, tRq_j, tCap) \quad (2.24)$$

Compared to traditional mathematical programming approaches, it is easier to model the renewable resources in the CP Optimizer. Furthermore, the *Cumulative* function enables handling the resource restrictions in scheduling problems in a more compact way. This expression is used to model quantities that vary over time based on an activity represented as an interval decision variable (IBM, 2021a). In this sense, Constraints (2.23)-(2.24) impose the usage of renewable setup and transport resources via the *Cumulative* function. At any time, the demand for the setup/transport resources cannot exceed the available number of resources exclusively defined for setup/transport activities. Resource usage increases by the resource requirement at the start of the activity, and then the resources are released when the activity ends.

4.3 Proposed Branching Strategies

The CP Optimizer enables to guide the search progress through efficient branching strategies defined by the practitioners (Rossi et al., 2006). Furthermore, it is possible to tune the search with problem-specific branching strategies to enhance constraint propagation. The aim is to expedite the search and direct the search to promising regions during the search progress.

The practitioner can define the search phases and parameters, considering the structure of the optimization problem. For example, the CP Optimizer presents effective search phases for scheduling problems. The scheduling search phases are based on the interval and sequence variables (IBM, 2019b). The search phases for the interval variables fix each interval variable to their presence status and start/end times. The search phases defined for the sequence variables fix the presence status and the interval variables' order in the sequence variables. Note that the start/end times of the interval variables are not fixed in the search phases defined for the sequence variables.

The variable and value ordering strategies have a crucial role in solving a constrained optimization problem. A variable ordering strategy effectively determines the variable to be branched on next, whereas a value ordering strategy effectively instantiates the selected variable to a value (Rossi et al., 2006). There are effective variable and value evaluators that utilize the performance of the CP model during the search progress. The evaluators are based on the history of the search. These evaluators can reference the selection of a variable for creating branches or an instantiated value of the selected variable. In this way, it is possible to detect infeasibility earlier in the search progress and direct the search toward promising regions. Moreover, search strategies can be employed, and the search parameters can be tuned depending on the problem structure to reduce the time spent in unpromising regions of the solution space.

The variable orderings presented in this thesis have been developed depending on the characteristics of the problem. Several decisions have to be made in the considered problem, such as lot streaming, sequencing, timing, and resource scheduling decisions. In this study, it is sensible to determine the lot streaming decisions before any other decisions made by the CP model. Sequencing and timing decisions are made based on the fixed sublots allocated to each machine. First, the CP model determines the subplot sizes and fixes them to their current value. Second, based on the fixed subplot sizes, the CP model determines the sequence of the sublots on each machine, and lastly, the timing decisions of the sublots on each machine are fixed based on the fixed machine sequence.

Considering the order of the lot streaming and scheduling decisions explained above, we propose the variable ordering in *Branching Strategy-1*. Additionally, based on the variable ordering in *Branching Strategy -1*, two variable orderings are proposed in *Branching Strategy-2* and *-3* taking into account the resource scheduling decisions of setup and transport activities. *Branching Strategy-1* proved to be the best variable ordering due to the computational experiments presented in Chapter 6. Based on the best variable ordering, three additional branching strategies are developed to enhance the constraint propagation in the CP model. These branching strategies make use of integer variable and value selectors and evaluators for lot streaming decisions handled by the integer variable $x_{j,s,o}$.

4.3.1 Branching Strategy-1

This branching strategy considers the lot streaming decisions handled by $x_{j,s,o}$ variable, sequencing decisions handled by mch_m variable, and machine allocation decisions handled by $xM_{j,s,o,m}$ variable correspondingly. The resource scheduling decisions of setup and transport activities are left to the CP model. The C# code for the *Branching Strategy-1* is given in Table 4.1. Three search phases are declared in Row (1) to define the proposed variable ordering strategy. The mentioned variable ordering is shown in Rows (2)-(4) accordingly, and finally, we set the search phases in Row (5).

Table 4.1 The C# code for the branching strategy-1

Branching Strategy-1	
1:	<code>ISearchPhase[] phases = new ISearchPhase[3];</code>
2:	<code>phases[0] = cp.SearchPhase($x_{j,s,o}$);</code>
3:	<code>phases[1] = cp.SearchPhase(mch_m);</code>
4:	<code>phases[2] = cp.SearchPhase($xM_{j,s,o,m}$);</code>
5:	<code>cp.SetSearchPhases(phases);</code>

4.3.2 Branching Strategy-2

This branching strategy considers the lot streaming decisions handled by $x_{j,s,o}$ variable, sequencing decisions handled by mch_m variable, and machine allocation decisions handled by $xM_{j,s,o,m}$ variable correspondingly. The scheduling decisions of setup activities handled by $sOp_{j,s,o}$ variable are taken before the scheduling decisions of transport activities handled by $tOp_{j,s,o}$ variable. The transport activities are scheduled based on the fixed setup activities. The C# code for the *Branching Strategy-2* is given in Table 4.2. Five search phases are declared in Row (1) to define the proposed variable ordering strategy. The mentioned variable ordering is shown in Rows (2)-(6) accordingly, and finally, we set the search phases in Row (7).

Table 4.2 The C# code for the branching strategy-2

Branching Strategy-2
1: <i>ISearchPhase[] phases = new ISearchPhase[5];</i>
2: <i>phases[0] = cp.SearchPhase($x_{j,s,o}$);</i>
3: <i>phases[1] = cp.SearchPhase(mch_m);</i>
4: <i>phases[2] = cp.SearchPhase($xM_{j,s,o,m}$);</i>
5: <i>phases[3] = cp.SearchPhase($sOp_{j,s,o}$);</i>
6: <i>phases[4] = cp.SearchPhase($tOp_{j,s,o}$);</i>
7: <i>cp.SetSearchPhases(phases);</i>

4.3.3 Branching Strategy-3

This branching strategy considers the lot streaming decisions handled by $x_{j,s,o}$ variable, sequencing decisions handled by mch_m variable, and machine allocation decisions handled by $xM_{j,s,o,m}$ variable correspondingly. The scheduling decisions of the transport activities handled by $tOp_{j,s,o}$ variable are taken before the decisions made for the setup activities handled by $sOp_{j,s,o}$ variable. The setup activities are scheduled based on the fixed transport activities. The C# code for the *Branching Strategy-3* is given in Table 4.3. Five search phases are declared in Row (1) to define the proposed variable ordering strategy. The mentioned variable ordering is shown in Rows (2)-(6) accordingly, and finally, we set the search phases in Row (7).

Table 4.3 The C# code for the branching strategy-3

Branching Strategy-3	
1:	<i>ISearchPhase[] phases = new ISearchPhase[5];</i>
2:	<i>phases[0] = cp.SearchPhase($x_{j,s,o}$);</i>
3:	<i>phases[1] = cp.SearchPhase(mch_m);</i>
4:	<i>phases[2] = cp.SearchPhase($xM_{j,s,o,m}$);</i>
5:	<i>phases[3] = cp.SearchPhase($tOp_{j,s,o}$);</i>
6:	<i>phases[4] = cp.SearchPhase($sOp_{j,s,o}$);</i>
7:	<i>cp.SetSearchPhases(phases);</i>

4.3.4 Branching Strategy-4

This branching strategy evaluates the integer variables $x_{j,s,o}$ taking into account the impact of these variables in the last search. *ImpactOfLastBranch* is an integer variable evaluator that provides access to the impact of the variables on the last branch. This evaluator is a measure of the domain reduction made at the last instantiation of the evaluated variable, and it ranges from zero to one (IBM, 2021b). If the *ImpactOfLastBranch* evaluation for a variable is high, this means that branching on that variable has a strong effect on the domain reduction performed in the last search. In general, it is preferable to select the variable that leads to strong domain reduction to guide search moves to the promising regions. Therefore, the variables having the largest impact on the last branch are selected in the proposed *Branching Strategy-4*. After the variable has been selected to be fixed, the integer value evaluator *ValueImpact* is used to decide the value for the selected variable. This evaluator calculates the average reduction of the search space when the selected variable is instantiated to this value. Therefore, a value with the smallest impact on reducing the search space is selected in case of early failure. The C# code for the *Branching Strategy-4* is given in Table 4.4.

Table 4.4 The C# code for the branching strategy-4

Branching Strategy-4	
1:	<i>ISearchPhase[] phases = new ISearchPhase[3];</i>
2:	<i>IVarSelector varSel = cp.SelectLargest(cp.ImpactOfLastBranch());</i>
3:	<i>IValueSelector valSel = cp.SelectSmallest(cp.ValueImpact());</i>
4:	<i>phases[0] = cp.SearchPhase($x_{j,s,o}$, cp.IntVarChooser(varSel), cp.IntValueChooser(valSel));</i>
5:	<i>phases[1] = cp.SearchPhase(mch_m);</i>
6:	<i>phases[2] = cp.SearchPhase($xM_{j,s,o,m}$);</i>
7:	<i>cp.SetSearchPhases(phases);</i>

4.3.5 Branching Strategy-5

This branching strategy evaluates the integer variables considering the current domain sizes. According to Rossi et al. (2006), the most efficient variable ordering strategies are associated with the current domain sizes of the variables. Thereby, we propose a dynamic ordering of the interval variables based on the current domain sizes of the variables. The integer variable evaluator *DomainSize* is used to access the current domain sizes of the integer variables. This evaluator returns the number of values left in the integer variable domains. The evaluator is dynamic, and the current domain size of the variables is updated as the search progresses. Therefore, the variables having the smallest domain size are selected in the proposed *Branching Strategy-2*. After the variable has been selected to be fixed, the value having the smallest value impact on the search space reduction is chosen to instantiate the selected variable. In this way, the domain of the selected variable is greatly reduced by experiencing early failures. The C# code for the *Branching Strategy-5* is given in Table 4.4.

Table 4.5 The C# code for the branching strategy-5

Branching Strategy-5	
1:	<i>ISearchPhase[] phases = new ISearchPhase[3];</i>
2:	<i>IVarSelector varSel = cp.SelectSmallest(cp.DomainSize());</i>
3:	<i>IValueSelector valSel = cp.SelectSmallest(cp.ValueImpact());</i>
4:	<i>phases[0] = cp.SearchPhase($x_{j,s,o}$, cp.IntVarChooser(varSel), cp.IntValueChooser(valSel));</i>
5:	<i>phases[1] = cp.SearchPhase(mch_m);</i>
6:	<i>phases[2] = cp.SearchPhase($xM_{j,s,o,m}$);</i>
7:	<i>cp.SetSearchPhases(phases);</i>

4.3.6 Branching Strategy-6

This branching strategy evaluates the integer variables by considering the success rate of the variables. *VarSuccessRate* is an integer variable evaluator that calculates the rate of success (no-failure) of a variable. When the variable is fixed n times by which result in f failures during the execution of the CP model, the variable success rate is calculated as $(n - f)/n$. In the proposed branching strategy, the variable having the smallest success rate is selected to be instantiated to a value having the smallest value impact. In this way, as early failures occur, the smallest impact values are removed from the domain of the variables that typically lead to failures. The C# code for the *Branching Strategy-6* is given in Table 4.6.

Table 4.6 The C# code for the branching strategy-6

Branching Strategy-6	
1:	<i>ISearchPhase[] phases = new ISearchPhase[3];</i>
2:	<i>IVarSelector varSel = cp.SelectSmallest(cp.VarSuccessRate());</i>
3:	<i>IValueSelector valSel = cp.SelectSmallest(cp.ValueImpact());</i>
4:	<i>phases[0] = cp.SearchPhase($x_{j,s,o}$, cp.IntVarChooser(varSel), cp.IntValueChooser(valSel));</i>
5:	<i>phases[1] = cp.SearchPhase(mch_m);</i>
6:	<i>phases[2] = cp.SearchPhase($xM_{j,s,o,m}$);</i>
7:	<i>cp.SetSearchPhases(phases);</i>

CHAPTER 5

CONSTRAINT PROGRAMMING-BASED LARGE NEIGHBORHOOD SEARCH

The CP-based Large Neighborhood Search (LNS) is an effective hybrid method that exploits the convergence ability of CP in large neighborhoods. This heuristic is particularly effective for constrained optimization problems with large feasible spaces. Hence, we propose a CP-based LNS for the FJSSP-LS under limited setup and transport resources. In this chapter, firstly, the hybridization of CP and metaheuristics is given. Later, the proposed CP-based LNS and its adaptive parameters are explained in detail.

5.1 Hybridization of CP and Metaheuristics

In recent years, the hybridization of CP and metaheuristics has become increasingly popular for OR practitioners. The metaheuristics are capable of diversifying the large feasible spaces, while CP is effective in finding high-quality feasible solutions subject to problem constraints. Therefore, CP and metaheuristics are complimentary solution approaches for solving constrained optimization problems (Kacprzyk & Pedrycz, 2015).

The integration of CP and metaheuristics is attracting growing attention in the literature. The existing work focuses on the hybridization of CP and genetic algorithms (Cormier, O'Grady, & Sanii, 1998; Deris, Omatu, Ohta, & Saad, 1999; Nguyen, Thiruvady, Zhang, & Tan, 2021; Sioud, Gravel, & Gagne, 2012; Wang, Chen, & Wang, 2007), ant colony optimization algorithms (Thiruvady, Meyer, & Ernst, 2011; Thiruvady, Singh, Ernst & Meyer, 2013; Zhang, & Zhang, 2018), and local search-based heuristics (Beck, Feng, & Watson, 2011; Rossi, Tarim, & Bollapragada, 2012; Jussien & Lhomme, 2002; Pesant & Gendreau, 1999; Prestwich, 2008; Valoux & Housos, 2003). In these hybridizations, CP has different roles in the exploration of the search space. For example, CP can create neighborhoods or be used as a local search procedure to seek solutions in neighborhoods. Accordingly, there are two kinds of

integration for CP and local search-based heuristics. The first kind of integration is designed to enhance the performance of CP through a local search (Kacprzyk & Pedrycz, 2015). Local search is used to explore the promising regions by refining the feasible search space. Limited Discrepancy Search is an example of this kind of integration. *Limited Discrepancy Search* (LDS) is an incomplete method to explore the neighboring paths by handling the discrepancies in the CP search tree (Kacprzyk & Pedrycz, 2015). This solution approach limits the exploration of the CP search tree and directs the search moves to the promising regions. The second kind of integration is designed to explore large feasible spaces with CP embedded local search-based heuristics (Kacprzyk & Pedrycz, 2015). This kind of integration is used to intensify the search with a CP model. The CP model acts as a local search procedure in a metaheuristic. *CP-based Large Neighborhood Search* (CP-based LNS) is an example of this kind of integration.

The CP-based LNS is an iterative improvement method for solving constrained optimization problems with large feasible spaces. At each iteration of the algorithm, the *global problem*, which includes all the decision variables of the problem, is transformed into a simplified *subproblem* by fixing the values of certain variables in each neighborhood. The subproblems defined in large neighborhoods are re-optimized in each iteration of the algorithm. This method improves the exploration of the search space by applying small perturbations to the current solution. A full CP model iteratively solves a large number of neighborhoods to extend the current solution to an improved one. The successful implementations of CP-based LNS can be observed in the resource-constrained project scheduling problem (Palpant, Artigues, & Michelon, 2004), the JSSP (Carchrae & Beck, 2009), underground mine scheduling problem (Åstrand et al., 2020), and vehicle routing problems (Hojabri, Gendreau, Potvin, & Rousseau, 2018; Kilby & Urli, 2016).

The CP-based LNS overcomes the challenges of a single CP optimization model. The CP model itself is effective in solving highly constrained scheduling problems. However, as the search progresses, the CP model may get stuck in a solution and may not improve the solution within a reasonable computation time. In this point, the

subproblems defined in the CP-based LNS have a reduced search space that a full CP model can handle. The subproblems must be large enough to include promising solutions and small enough to be handled by a full CP model in a reasonable time (Blum, Puchinger, Raidl, & Roli, 2011). Therefore, the determination of the subproblem size is a critical issue. The most straightforward implementation is based on the percentage of the problem variables to define the subproblem or namely, the neighborhood. A specified percent of the problem variables is fixed to their current values, and the remaining problem variables define the size of the subproblem. Consequently, the subproblem has a considerably reduced search space for the global problem. The CP model finds the feasible solution of the subproblem by extending the partial assignment to the complete assignment of the values for the unassigned variables. Note that a solution to the subproblem is also a feasible solution to the global problem.

The CP-based LNS is an effective solution approach for tackling complex optimization problems when it is possible to define the subproblems to be effectively solved by CP. The execution of CP is faster than the enumeration, contrary to the local search heuristics, which enumerates the solutions in confined areas of the search space (Blum et al., 2011). Hybridization of CP and LNS enhances the performance of the two solution methodologies. The LNS exploits the exploration advantage of the metaheuristics by defining large neighborhoods, while CP effectively tackles the problem's constraints. Hence, this hybridization is effective in solving highly constrained optimization problems that have large feasible spaces.

5.2 The Proposed CP-Based LNS

This section proposes a CP-based LNS for solving the FJSSP-LS with setup and transport resources. This solution approach is based on a re-optimization idea and outperforms CP by tackling the problem's computational complexity. The idea is to explore the search space by solving large neighborhoods with a full CP model.

The solution approaches based on LNS have received attention for solving transportation and scheduling problems in recent years (Gendreau & Potvin, 2010). This metaheuristic creates the neighborhood by *destroying* and *repairing* the incumbent solution. The destroy method creates an infeasible solution, which then turns out to be a feasible solution by repairing it. The destroy and repair method can be regarded as a fix and optimize operation when a part of the solution is fixed to its current value, and the remaining variables are optimized (Gendreau & Potvin, 2010). In this view, the fixing operation corresponds to the destroy method, while the optimizing operation corresponds to the repair method. Also, the repair method may be handled using a MILP or CP solver as well as a heuristic method (Gendreau & Potvin, 2010). In this context, the basic pseudocode of the LNS is given in Table 5.1.

Table 5.1 The basic pseudocode of the LNS

LNS	
1:	Compute an initial solution: $\theta \leftarrow \theta_0$
2:	while stopping criteria is not satisfied do
3:	Create neighborhood by destroying and repairing θ
4:	Update the best solution: $\theta \leftarrow \theta_{best}$
5:	end while

In the proposed CP-based LNS, the initial solution is first derived by solving a full CP model. Then, in each iteration of the algorithm, certain variables are fixed to their current values, and a subproblem is created with the remaining variables. A full CP model solves the subproblem, and the current solution is updated. In this way, the current solution is iteratively improved by a full CP model. At this point, the proposed branching strategy for the CP model is effective in finding feasible solutions within the time limit defined for each iteration. Moreover, the subproblem is easier to solve since the global problem is simplified by fixing a part of the solution.

In the proposed CP-based LNS, a fixing strategy based on subplot size and machine allocation decisions is developed. In the FJSSP-LS, the problem size can be controlled by the total number of operations, denoted as (j, s, o) triples where j represents the job, s represents the associated subplot, and o represents the related operations of the job for the relevant subplot. The variable $x_{j,s,o}$ that denotes the size of the subplot and

the corresponding size of interval variables $xM_{j,s,o,m}$ that represent the subplot operations on the machines are fixed initially. The notation used in the CP-based LNS is given in Table 5.2.

Table 5.2 The notation used in the pseudocode of the CP-based LNS

$\mathcal{P}$	Global Problem
$\mathcal{X}$	Subproblem
θ	Current and best solution
θ'	Intermediate solution
$\mathcal{P}_\theta$	The current size of the global problem denoted as (j, s, o) triples
Z_θ	The objective value of the best solution
$Z_{\theta'}$	The objective value of the intermediate solution
ρ_t	Number of the (j, s, o) triples to be fixed in iteration t
$\rho_t^{min}, \rho_t^{max}$	The lower and upper limits defined for ρ_t
ℓ_t	Time limit defined for solving the subproblem by the full CP model in iteration t
$\ell_{initial}$	Time limit defined for the CP model to find an initial solution
V	Set of the index values (j, s, o) of the variables of the global problem
$\mathcal{F}_{(j,s,o)}$	Set of the index values (j, s, o) of the variables to be fixed
$\mathcal{F}x_{(j,s,o)}$	Set of the variables $x_{j,s,o}$ to be fixed
$\mathcal{F}p_{(j,s,o,m)}$	Set that includes the size of the variables $xM_{j,s,o,m}$ to be fixed

As mentioned earlier, the subproblems must be small enough to be handled by a CP model and large enough to contain promising solutions. Therefore, the number of variables to be fixed in each iteration is determined according to the ρ_t parameter, considering the current size of the global problem $\mathcal{P}_\theta$, regarding (j, s, o) triples and these variables are randomly chosen from the global problem for the successive iteration. For each triple included in the set $\mathcal{F}_{(j,s,o)}$, the subplot sizes $x_{j,s,o}$ and the size of the variables $xM_{j,s,o,m}$ are fixed to their current values in the current solution θ . The subproblem is derived from the remaining variables; then, the full CP model is solved for the resulting subproblem. Here, the aim is to relax the problem by fixing a part of the solution and improve the current solution by solving the problem for the free variables. Note that we give the current solution as a starting point for each successive iteration. The *ISolution* interface, which represents solutions of the optimization

problem in C# (IBM, 2021c), creates an object for the CP solution and instantiates all decision variables to their current values. In this way, we provide a starting point to CP so that the current solution is iteratively improved in each iteration. At the end of each iteration, the adaptive parameters ρ_t^{min} , ρ_t^{max} , ρ_t , and ℓ_t are updated regarding the solution of the subproblem. The ρ_t^{min} and ρ_t^{max} are the lower and upper limits defined for ρ_t , and ℓ_t is the time limit defined for solving the subproblem by the full CP model at iteration t . The procedure for updating the adaptive parameters is explained in Section 5.2.1. This process iteratively continues until the algorithm terminates within a time limit defined for the overall algorithm. The pseudocode for the proposed CP-based LNS is given in Table 5.3.

Table 5.3 The pseudocode for the proposed CP-based LNS

CP-based LNS	
1:	Compute an initial solution: $\theta \leftarrow CP_{Full}$
2:	Set $\mathcal{P}_\theta$ for the initial solution θ , $t \leftarrow 1$
3:	Initialize ρ_t^{min} , ρ_t^{max} , ρ_t , ℓ_t
4:	while not terminated do
5:	Initialize the set V : $V \leftarrow \{(j, s, o) \in \mathcal{P}\}$
6:	for 1 to ρ_t do
7:	Randomly choose a (j, s, o) triple from V in where $j \neq 0$
8:	$\mathcal{F}_{(j,s,o)} \leftarrow (j, s, o)$
9:	$V \leftarrow V \setminus \{(j, s, o) \in \mathcal{F}_{(j,s,o)}\}$
10:	end for
11:	for all $(j, s, o) \in \mathcal{F}_{(j,s,o)}$
12:	$\mathcal{F}x_{(j,s,o)} \leftarrow x_{j,s,o}$
13:	for all $m \in M$
14:	$\mathcal{F}p_{(j,s,o,m)} \leftarrow sizeOf(xM_{j,s,o,m})$
15:	end for
16:	end for
17:	Derive the subproblem $\mathcal{X}$ by fixing a part of the solution
18:	$\mathcal{X} \leftarrow \{\mathcal{P} x_{j,s,o} \leftarrow \mathcal{F}x_{(j,s,o)}, sizeOf(xM_{j,s,o,m}) \leftarrow \mathcal{F}p_{(j,s,o,m)}\}$
19:	Give intermediate solution and solve the CP_{Full}
20:	$\theta' \leftarrow solveCP_{Full}\{\mathcal{X}\}$
21:	Update the adaptive parameters ρ_t^{min} , ρ_t^{max} , ρ_t , and ℓ_t
22:	if $Z_{\theta'} < Z_\theta$
23:	$\theta \leftarrow \theta'$
24:	$\mathcal{P}_\theta \leftarrow \mathcal{P}_{\theta'}$
25:	end if
26:	$t \leftarrow t + 1$
27:	end while

5.2.1 Adaptive Parameters of the CP-based LNS

We have noted that the size of the problem changes as the search progresses with the optional interval variables scheduled. While only a few variables are involved at the start of the search, the number of variables increases over time as a result of lot streaming decisions during the search. Therefore, adaptive parameters are required to maintain the size of the subproblem within certain limits so that a full CP model can handle it. These parameters update themselves according to the best solution found so far. The proposed adaptive parameters effectively capture the complexity of the lot streaming problem and enhance the effectiveness of the solution approach. Therefore, we have proposed two adaptive parameters for the CP-based LNS; one for controlling the subproblem size and the other for controlling the time limit for the CP model in each iteration, given in Section 5.2.1.1 and Section 5.2.1.2, respectively.

5.2.1.1 Adaptive Parameter for Defining the Subproblem Size

We propose an adaptive parameter for defining the size of the neighborhood, or namely the subproblem size, to manage the complexity of the subproblem by a full CP model. Accordingly, we define a lower limit ρ_t^{min} and upper limit ρ_t^{max} for the number of (j, s, o) triples to be fixed in the problem. These limits are initialized by using two parameters α and β , and calculated with the formulas presented in Equation (5.1) and Equation (5.2) in where $\mathcal{P}_{\theta_{t=0}}$ denotes the size of the global problem in the initial solution. The preliminary experiments indicate that the best values for the α and β parameters are 0.07 and 0.15, respectively. The number of (j, s, o) triples to be fixed in each iteration t is determined by the ρ_t value, and this value cannot exceed these lower and upper limits. It must be within the interval $[\rho_t^{min}, \rho_t^{max}]$. We randomly select a value within the interval $[\rho_{t=1}^{min}, \rho_{t=1}^{max}]$ for initializing $\rho_{t=1}$ for the first iteration (Hojabri et al., 2018).

$$\rho_{t=1}^{min} = \alpha \mathcal{P}_{\theta_{t=0}} \quad (5.1)$$

$$\rho_{t=1}^{max} = \beta \mathcal{P}_{\theta_{t=0}} \quad (5.2)$$

It is noted that a few optional variables are present in the solution at the start of the algorithm; however, as the algorithm proceeds, lot streaming decisions become effective to improve the current solution. Consequently, the problem size increases, making it harder to find a feasible solution within a given time limit. Therefore, we consider the elapsed time of the algorithm for adaptively updating the ρ_t values. In addition, neighborhoods must have promising solutions for the problem. Over time, as the current solution improves, it becomes harder to find a better solution for subproblems of small size. Therefore, a smaller ρ_t value is useful to include promising solutions in the neighborhood for the successive iterations. The calculation of the ρ_t^{min} and ρ_t^{max} values are given in Equation (5.3) and Equation (5.4), where ET denotes the elapsed time of the algorithm and TL denotes the time limit for the overall algorithm, which is 3600 s. in our case.

$$\rho_t^{min} = \alpha \left[1 - \left(\frac{ET}{TL} \right)^2 \right] \mathcal{P}_{\theta_{t-1}} \quad (5.3)$$

$$\rho_t^{max} = \beta \left[1 - \left(\frac{ET}{TL} \right)^2 \right] \mathcal{P}_{\theta_{t-1}} \quad (5.4)$$

The updating procedure for the ρ_t value is given in Hojabri et al. (2018), which is based on the improvement of solutions. If no improvement is achieved in the last performed iteration, the ρ_t parameter is decreased by one. The ρ_t parameter is increased by one when the problem size (j, s, o) and ρ_t have remained unchanged, and the algorithm has made an improvement on the objective function in the last N iterations, which is considered as three in our case.

5.2.1.2 Adaptive Parameter for Time Limitation of the CP Model

An adaptive parameter is provided to determine the time limit of the CP model in each iteration. As the search progresses, the algorithm converges to a solution, and it becomes harder to find a better solution within a strict time limit. When the size of the subproblem increases, the subproblem becomes more difficult and requires more time to find a feasible solution. Therefore, we propose an adaptive time limit based on the changes in subproblem sizes between consecutive iterations. The parameters λ_1 and λ_2 determine the rate at which the time limit is increased for each iteration. The λ_1 and λ_2 values are set to 0.5 and 2, respectively, in this study. The formulas are given in Equation (5.5) and Equation (5.6), in which $\mathcal{X}_t$ represents the current subproblem size in iteration t , defined by the unassigned number of (j, s, o) triples, namely the subproblem size, ℓ_t is a time limit for solving the neighborhood by the CP model, and $\mathcal{T}$ is a scalar or step size. The initial time limit for solving the neighborhood by the CP model $\ell_{t=1}$ is given as 40 s. for the first iteration.

$$\mathcal{T} = \frac{\mathcal{X}_t - \mathcal{X}_{t-1}}{\mathcal{X}_{t-1}} \quad (5.5)$$

$$\ell_{t+1} = \begin{cases} \ell_t + \mathcal{T}^{\lambda_1} \ell_t, & \text{if } \mathcal{T} \geq 0 \\ \ell_t - \mathcal{T}^{\lambda_2} \ell_t, & \text{otherwise} \end{cases} \quad (5.6)$$

Note that while ρ_t is updated based on an improvement in previous iterations, the time limit is always constant for problems with certain difficulty as long as the number of operations (j, s, o) included in the subproblem remains the same.

CHAPTER 6

COMPUTATIONAL STUDY

In this chapter, the computational study is performed to test the performances of the CP model and the CP-based LNS by solving the generated test problems. Initially, an experimental analysis of the proposed branching strategies and the parameters of the CP-based LNS is performed so that the performance of the CP-based LNS is tested using the best branching strategy and the best parameters determined. The parameters are tuned using the Taguchi method, and the computational results are analyzed within the perspective of the lot streaming problem. Subsequently, the effects of lot streaming on the FJSSP, the convergence behavior of the CP-based LNS are investigated, and the proposed solution methodologies are compared in terms of objective value and computation time.

6.1 Generation of Test Instances

There is a lack of lot streaming benchmark instances for the lot streaming problems in the FJSSP literature. Therefore, we consider generating our test instances according to the proposed procedure Novas (2019). Novas (2019) extends the FJSSP benchmark instances (Fattahi, Mehrabad, & Jolai, 2007) by considering these processing times for a single part included in a job. The characteristics of these instances are shown in Table 6.1 where $|J|$ denotes the number of jobs, $|O_j|$ denotes number of operations required for job j , $|M|$ denotes number of machines, and “Avg $\#M_{j,o}$ ” denotes the average number of machines per operation. The job size js_j is randomly generated using uniform distribution between limits [5,50] and the maximum number of sublots from each job S_j is given as either 3 or 5 (Novas, 2019). These values are given based on the maximum possible number of operations of sublots that can be processed in parallel. Note that the number of operations for each job is the same for all the FJSSP benchmark instances of Fattahi et al. (2007). We extend these instances given in Table 6.1, considering the special characteristics of our problem.

Each extended benchmark set includes data for four different scenarios to illustrate different setup and transport time cases, similar to those of Ahmadizar and Shahmaleki (2014). Based on each benchmark instance presented in Fattahi et al. (2007), we exclusively generate SDSTs and transport times for these four scenarios.

Table 6.1 Characteristics of Fattahi et al.'s (2007) benchmark instances

No	Instance	$ J $	$ O_j $	$ M $	Avg $\#M_{j,o}$
1	SFJS2	2	2	2	1.5
2	SFJS6	3	3	2	1.6
3	SFJS9	3	3	3	2
4	MFJS1	5	3	6	2.2
5	MFJS2	5	3	7	2.7
6	MFJS3	6	3	7	2.7
7	MFJS4	7	3	7	2.7
8	MFJS5	7	3	7	2.6
9	MFJS6	8	3	7	2.6
10	MFJS7	8	4	7	2.4
11	MFJS8	9	4	8	2.4
12	MFJS9	11	4	8	2.3
13	MFJS10	12	4	8	2.3

The SDSTs and transport times are generated regarding the processing times of these instances. The upper bound for the SDST generation is based on the percentage of the maximum processing time (Ruiz & Maroto, 2006; Ruiz & Stützle, 2008). Thus, the SDSTs are randomly generated between the minimum possible processing time and 50%, 75%, 100%, 125% of the maximum possible processing time of the FJSSP small-sized benchmark instances for the small-sized instances. Accordingly, the SDSTs are generated from a uniform distribution in the given intervals [17,107], [17,161], [17,214], and [17,268] respectively, and the following triangle inequality in Equation (6.1) holds in which i , j , and k represent the jobs and m represents the machines. This triangle inequality is necessary to generate balanced setup times for the operations. Thus, no operation can dominate other operations in the manufacturing process in terms of setup time.

$$st_{i,j,m} \leq st_{i,k,m} + st_{k,j,m} \quad \forall(i,j,k,m) \quad (6.1)$$

The transport times $tt_{m,m'}$ are randomly generated between the minimum possible processing time and 40%, 60%, 80%, 100% of the maximum possible processing time of the FJSSP small-sized instances for scenarios S50, S75, S100, and S125, respectively. The transport times are sampled from a uniform distribution in the given intervals [17,86], [17,128], [17,171], and [17,214] respectively. Similarly, the SDSTs are generated randomly between the minimum possible processing time and 50%, 75%, 100%, 125% of the maximum possible processing time for the FJSSP medium- and large-sized benchmark instances. As a result, for each scenario, the SDSTs are sampled from a uniform distribution in the given intervals [40,179], [40,268], [40,357], and [40,446], respectively, for which the triangle inequality also holds. Transport time $tt_{m,m'}$ is randomly generated between the minimum possible processing time and 40%, 60%, 80%, 100% of the maximum possible processing time of the FJSSP medium- and large-sized benchmark instances for scenarios S50, S75, S100, and S125, respectively. Thus, the transport times are sampled from a uniform distribution in the given intervals [40,143], [40,214], [40,286], and [40,357], respectively. It should be noted that the upper bound for the intervals is set to 80% of the largest possible SDSTs for each scenario.

The available number of setup resources $sCap$ is randomly generated in the given intervals [1,2] for small-sized instances and [2,3] for medium- and large-sized instances, respectively. According to Yepes-Borrero et al. (2020), setup resource requirement for the setup activity sRq_j is randomly generated between [1, $sCap$] for small-sized instances. For medium- and large-sized instances, sRq_j is randomly generated between [1,2]. Lastly, the available number of transport resources is randomly generated [1,2] for small-sized instances and [2,3] for medium- and large-sized instances. Since each subplot requires a single transport resource, transport resource requirement tRq_j is one for all instances.

The following scheme encodes the representation of the instances throughout the thesis: $\{Original\ Fattahi\ et\ al.\ (2007)\ instance\ name\}: \{S\%\}$. The first part represents the original name of the benchmark instance, while the second part

represents the percentage determined for the SDST generation for the related scenario. Furthermore, we divide the medium and large-sized benchmark instances into two problem classes considering the complexity of the lot streaming problem. The medium-sized instances are extended from MFJS1, MFJS2, MFJS3, MFJS4, MFJS5 instances, while large-sized instances are extended from MFJS6, MFJS7, MFJS8, MFJS9, MFJS10 instances. Since each instance is extended with four scenarios, the computational study is performed with 12 small-sized, 20 medium-sized, and 20 large-sized instances. The generating conditions for the instances are given in Table 6.2.

Table 6.2 Generating conditions for the test instances

Instance	Scenario	js_j	S_j	$st_{i,j,m}$	$tt_{m,m'}$	sRq_j	tRq_j	$sCap$	$tCap$
Small-sized	S50	[5,50]	3 or 5	[17,107]	[17,86]	[1, $sCap$]	1	[1,2]	[1,2]
	S75	[5,50]	3 or 5	[17,161]	[17,128]	[1, $sCap$]	1	[1,2]	[1,2]
	S100	[5,50]	3 or 5	[17,214]	[17,171]	[1, $sCap$]	1	[1,2]	[1,2]
	S125	[5,50]	3 or 5	[17,268]	[17,214]	[1, $sCap$]	1	[1,2]	[1,2]
Medium- & large-sized	S50	[5,50]	3 or 5	[40,179]	[40,143]	[1,2]	1	[2,3]	[2,3]
	S75	[5,50]	3 or 5	[40,268]	[40,214]	[1,2]	1	[2,3]	[2,3]
	S100	[5,50]	3 or 5	[40,357]	[40,286]	[1,2]	1	[2,3]	[2,3]
	S125	[5,50]	3 or 5	[40,446]	[40,357]	[1,2]	1	[2,3]	[2,3]

6.2 Experimental Analysis for Proposed Branching Strategies and Parameters

In this section, firstly, the preliminary experiments on the proposed branching strategies are given, then the parameter settings for the CP-based LNS are done using the Taguchi method.

6.2.1 The Comparison of Branching Strategies for the CP Model

The best branching strategy is determined using a series of computational experiments. The experiments are conducted for 20 medium- and large-sized instances, including all scenarios. The CP model is executed under a time limit of 1200 s. Table 6.3 presents the solutions obtained within the specified time limit. The performance of the six branching strategies is compared in terms of objective value and computation time. The relative percentage deviation (RPD) compares the

performance of branching strategies in terms of objective values. The procedure for calculating the RPD is shown in Equation (6.2), where CP_{obj}^{BS} denotes the objective value of the CP model using the branching strategy being evaluated, and CP_{obj}^{Best} denotes the best objective value obtained by any of the six branching strategies.

$$RPD = \frac{CP_{obj}^{BS} - CP_{obj}^{Best}}{CP_{obj}^{Best}} \times 100 \quad (6.2)$$

Furthermore, we also report the improvements brought by each of the proposed branching strategy-4, -5, and -6 subject to the best variable ordering strategy defined in branching strategy-1. The improvements of the objective values are calculated using Equation (6.3), where $CP_{obj}^{BS(1)}$ denotes the objective value of the CP model using branching strategy-1, and $CP_{obj}^{BS(X)}$ denotes the objective value of the CP model using branching strategy- X being evaluated.

$$Imp \% = \frac{CP_{obj}^{BS(1)} - CP_{obj}^{BS(X)}}{CP_{obj}^{BS(1)}} \times 100 \quad (6.3)$$

The computational results for the proposed branching strategies are shown in Table 6.3. The CP model with no branching strategy could not find any feasible solution within the 1200 s. time limit, while the proposed branching strategies are effective in finding feasible solutions for the given instances. We also note that some of the best solutions are superior in computation time. These computation times are shown in bold in Table 6.3 to highlight the advantage over the branching strategies in terms of computation time.

Table 6.3 Computational results for the test instances using the proposed branching strategies

Instance	CP (with BS-1)		CP (with BS-2)		CP (with BS-3)		CP (with BS-4)		CP (with BS-5)		CP (with BS-6)	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
MFJS2:S50	13520	448.64	13445	526.15	13476	742.87	13310	614.81	13302	1085.82	13363	681.21
MFJS2:S75	12842	455.07	12782	544.48	12937	921.28	12717	979.21	12717	865.38	12657	417.55
MFJS2:S100	6171	735.96	5916	485.15	6006	939.11	6004	306.54	5856	488.26	5882	241.64
MFJS2:S125	14004	237.44	13688	956.31	13559	993.76	13860	619.69	13501	1117.18	13418	1160.79
MFJS5:S50	12780	1051.05	12869	868.96	12933	422.54	12369	863.57	12600	290.90	12620	892.90
MFJS5:S75	12801	938.71	12839	1130.17	12872	1138.36	11859	1120.06	12779	1076.98	11651	1134.45
MFJS5:S100	11768	1064.25	11893	400.42	12177	673.54	11495	725.54	11516	791.69	11556	930.42
MFJS5:S125	13985	710.52	14022	735.08	13712	1154.82	13621	354.23	13729	1158.29	13580	768.14
MFJS7:S50	33126	1011.47	34823	874.89	32648	960.85	31109	974.40	31876	1145.93	32541	981.82
MFJS7:S75	25372	1193.40	24658	1082.78	26836	1144.65	23193	807.70	23227	1166.77	23366	610.54
MFJS7:S100	24136	936.05	24932	840.80	23926	1092.73	22619	629.49	22953	1016.36	23856	689.78
MFJS7:S125	31117	1042.21	29103	792.83	29168	1155.31	29194	847.47	28972	1038.11	31034	1158.87
MFJS9:S50	30854	968.49	32788	102.46	32684	371.14	28574	1081.48	28528	1093.00	28421	943.62
MFJS9:S75	26491	1123.46	24549	916.19	25496	1106.82	24063	1180.79	25607	1152.58	25587	1069.96
MFJS9:S100	31245	1175.59	32541	996.66	34193	1037.36	30887	1105.43	30851	1072.48	30912	1102.80
MFJS9:S125	39836	98.70	40495	518.38	40043	433.60	36631	1076.55	37378	520.80	38362	953.92
MFJS10:S50	32079	1196.38	35018	244.47	31934	997.99	30921	852.17	31414	1195.24	31238	830.80
MFJS10:S75	36669	1189.80	36814	1118.27	37019	1070.22	36657	310.06	36228	1130.54	35997	1175.18
MFJS10:S100	31589	1121.08	33565	755.33	33905	1021.23	30239	759.19	29935	955.19	31286	1055.06
MFJS10:S125	45278	990.64	45536	1028.25	45542	1197.05	42705	1150.48	44686	1005.36	43882	1007.42
Average	24283.15	884.45	24613.80	745.90	24553.30	928.76	23101.35	817.94	23382.75	968.34	23560.45	890.34

Time: the time required to instantiate the best objective value within 1200 s.; best results in **bold**

The paired t-tests are conducted to evaluate the difference between the objective values of the selected branching strategy from each of the other branching strategies over the given instances. The paired t-tests for the branching strategies-1, -2, and -3 are reported in Table 6.4 for these instances. There is insufficient statistical evidence that the objective values obtained by these branching strategies are significantly different from each other. Therefore, we select the branching strategy that has the lowest average RPD (ARPD) value.

Table 6.4 The results of paired t-tests for branching strategies-1, -2, and -3

Comparison	t-Stat	df	p-value	95% CI
BS1-BS2	-1.21	19	0.24	[-902, 241]
BS1-BS3	-1.08	19	0.30	[-795, 255]
BS2-BS3	0.24	19	0.81	[-467, 588]

As shown in Table 6.5, branching strategy-1 has the lowest ARPD value of 5.36 among the branching strategies-1, -2, and -3. Therefore, the variable ordering proposed in this branching strategy is used to develop three branching strategies: branching strategy-4, -5, and -6.

In addition to variable ordering proposed in branching strategy-1, the branching strategies-4, -5, and -6 make use of integer variable and value evaluators for the variable $x_{j,s,o}$, as mentioned previously in Chapter 4. Hence, paired t-tests are performed to analyze the effect of integer variable and value evaluators in the branching strategies-4, -5, and -6. The results of paired t-tests are reported in Table 6.6. It is statistically proven that the mean objective value of the branching strategy-1 is significantly different from the mean objective values of branching strategies-4, -5, and -6. Therefore, it is concluded that each of the proposed branching strategies 4, 5, and 6 is superior to the branching strategy-1, which implies that the use of integer variable and value evaluators are effective to expedite the search progress.

Table 6.5 Comparison of the proposed branching strategies according to objective value

Instance	RPD						Improvements on objective value		
	BS-1	BS-2	BS-3	BS-4	BS-5	BS-6	Imp(1)%	Imp(2)%	Imp(3)%
MFJS2:S50	1.64	1.08	1.31	0.06	0.00	0.46	1.55	1.61	1.16
MFJS2:S75	1.46	0.99	2.21	0.47	0.47	0.00	0.97	0.97	1.44
MFJS2:S100	5.38	1.02	2.56	2.53	0.00	0.44	2.71	5.10	4.68
MFJS2:S125	4.37	2.01	1.05	3.29	0.62	0.00	1.03	3.59	4.18
MFJS5:S50	3.32	4.04	4.56	0.00	1.87	2.03	3.22	1.41	1.25
MFJS5:S75	9.87	10.20	10.48	1.79	9.68	0.00	7.36	0.17	8.98
MFJS5:S100	2.37	3.46	5.93	0.00	0.18	0.53	2.32	2.14	1.80
MFJS5:S125	2.98	3.25	0.97	0.30	1.10	0.00	2.60	1.83	2.90
MFJS7:S50	6.48	11.94	4.95	0.00	2.47	4.60	6.09	3.77	1.77
MFJS7:S75	9.40	6.32	15.71	0.00	0.15	0.75	8.59	8.45	7.91
MFJS7:S100	6.71	10.23	5.78	0.00	1.48	5.47	6.29	4.90	1.16
MFJS7:S125	7.40	0.45	0.68	0.77	0.00	7.12	6.18	6.89	0.27
MFJS9:S50	8.56	15.37	15.00	0.54	0.38	0.00	7.39	7.54	7.89
MFJS9:S75	10.09	2.02	5.96	0.00	6.42	6.33	9.17	3.34	3.41
MFJS9:S100	1.28	5.48	10.83	0.12	0.00	0.20	1.15	1.26	1.07
MFJS9:S125	8.75	10.55	9.31	0.00	2.04	4.73	8.05	6.17	3.70
MFJS10:S50	3.75	13.25	3.28	0.00	1.59	1.03	3.61	2.07	2.62
MFJS10:S75	1.87	2.27	2.84	1.83	0.64	0.00	0.03	1.20	1.83
MFJS10:S100	5.53	12.13	13.26	1.02	0.00	4.51	4.27	5.24	0.96
MFJS10:S125	6.03	6.63	6.64	0.00	4.64	2.76	5.68	1.31	3.08
Average	5.36	6.13	6.17	0.64	1.69	2.05	4.41	3.45	3.10

Imp(1)%: Percent improvement in objective value with BS-4 with respect to BS-1; Imp(2)%: Percent improvement in objective value with BS-5 with respect to BS-1; Imp(3)%: Percent improvement in objective value with BS-6 with respect to BS-1.

Table 6.6 The results of paired t-tests for the comparison of branching strategy-1, and branching strategies -4, -5, and -6

Comparison	t-Stat	df	p-value	95% CI
BS1-BS4	5.21	19	0.00	[707, 1656]
BS1-BS5	4.94	19	0.00	[519, 1282]
BS1-BS6	4.90	19	0.00	[414, 1031]

Furthermore, paired t-tests are performed to determine the best branching strategy among the branching strategies 4, 5, and 6. The results of paired t-tests are reported in Table 6.7. It is statistically proven that the mean objective value of branching strategy-4 is significantly different from the mean objective value of branching strategy-6. However, there is insufficient statistical evidence that the mean objective value of branching strategy-4 is significantly different from the mean objective value of the branching strategy-5. Thus, we select the branching strategy that has the lowest ARPD value for further computational study. As shown in Table 6.5, the branching strategy-4 has the lowest ARPD value of 0.64 and provides the best improvement of 4.41 on

average. Since the branching strategy-4 has the lowest ARPD value for the test instances, it is selected to be used in the proposed CP-based LNS.

Table 6.7 The results of paired t-tests for the comparison of branching strategy -4, -5, and -6

Comparison	t-Stat	df	p-value	95% CI
BS4-BS5	-1.99	19	0.06	[-578, 15]
BS4-BS6	-2.64	19	0.02	[-823, -95]
BS5-BS6	-1.11	19	0.28	[-513, 158]

6.2.2 The Parameter Settings for the CP-based LNS

Determining the best combination of parameter levels requires a large number of computational experiments when the full factorial design is adopted to validate the best parameters for which robustness is obtained in the CP-based LNS. For this reason, the Taguchi method is used to determine the best combination of parameter levels that provide the robustness of the CP-based LNS. The parameters and their levels are given in Table 6.8, where α denotes the initial lower limit for the percentage of the (j, s, o) triples to be fixed to define the least number of variables to be fixed ($\rho_{t=1}^{min}$) at iteration one, β denotes the initial upper limit for the percentage of the (j, s, o) triples to be fixed to define the highest number of variables to be fixed ($\rho_{t=1}^{max}$) at iteration one, $\ell_{initial}$ denotes the time limit defined for the CP model to initialize the algorithm, $\ell_{t=1}$ denotes the initial time limit for solving the neighborhood solution by the CP model at the first iteration, and N denotes the number of iterations to be evaluated for updating the number of variables to be fixed at iteration t , which is denoted by the ρ_t parameter.

Table 6.8 Parameter levels for the Taguchi design

Level	α	β	$\ell_{initial}$	$\ell_{t=1}$	N
1	0.06	0.13	150	20	1
2	0.07	0.14	200	30	2
3	0.08	0.15	250	40	3

Taguchi (1986) proposes a robust parameter design that evaluates specific parameter combinations so that the number of required experiments can be greatly

reduced compared to the full factorial experiments. The experiments consist of factors that are classified into two categories: controllable factors and noise factors. The controllable factors are the factors under our control, while the noise factors are the response variables of the experiments. In this thesis, the control factors correspond to the parameters of the CP-based LNS, and the noise factors correspond to the objective value obtained by conducting the experiments. In this context, the Taguchi method aims to minimize the variation of the noise factors induced by the control factors to create a robust design. Signal-to-noise (S/N) ratios are used to evaluate the influence of control factors on noise factors. The goal is to achieve optimal combinations of parameter settings by maximizing the S/N ratio under an objective. Accordingly, Taguchi (1986) provides three options for the objectives “smaller is better,” “larger is better,” and “nominal is best.” Since the optimization objective of our problem is minimizing the makespan of the schedule, we have chosen the “smaller is better” option while creating the Taguchi design. S/N ratio used with the “smaller is better” option is given in Equation (6.4).

$$S/N \text{ ratio} = -10 \log_{10} \left(\frac{1}{n} \sum_{i=1}^n (\text{objective})_i^2 \right) \quad (6.4)$$

Taguchi (1986) proposes an orthogonal array on which the inner array includes the control factors, and the outer array includes the noise factors. The method analyzes the control factors independently from each other. A large-sized problem instance (MFJS10:S75) is considered to conduct the Taguchi experiments. The appropriate design for the experiments is L27. Five replications are taken for each parameter combination in the designed experiments, and the objective values obtained are reported. Accordingly, the orthogonal array generated for the Taguchi design is given in Table 6.9, where the parameters α , β , ℓ_{initial} , $\ell_{t=1}$, and N are the control factors; and the objective value of the replications are the noise factors for the Taguchi design.

Table 6.9 The Taguchi design for the computational experiments

Exp #	α	β	$\ell_{initial}$	$\ell_{t=1}$	N	Obj-1	Obj-2	Obj-3	Obj-4	Obj-5
1	0.06	0.13	150	20	1	34966	34722	35039	34966	34674
2	0.06	0.13	150	20	2	34628	35253	34606	34303	34746
3	0.06	0.13	150	20	3	34509	35493	34311	34817	35203
4	0.06	0.14	200	30	1	34999	34913	34459	34497	34746
5	0.06	0.14	200	30	2	34628	35146	34406	35493	35160
6	0.06	0.14	200	30	3	34795	35159	34342	34229	34311
7	0.06	0.15	250	40	1	34459	34430	34930	34342	34628
8	0.06	0.15	250	40	2	34666	34459	34303	34517	35112
9	0.06	0.15	250	40	3	34687	35054	34958	34229	34913
10	0.07	0.13	200	40	1	34905	34581	35382	34930	34817
11	0.07	0.13	200	40	2	34966	34722	34632	34342	34571
12	0.07	0.13	200	40	3	34430	34199	34666	34292	34628
13	0.07	0.14	250	20	1	34517	35195	34722	35132	35146
14	0.07	0.14	250	20	2	34905	35078	34623	35128	34628
15	0.07	0.14	250	20	3	34560	34656	34539	34581	34571
16	0.07	0.15	150	30	1	34529	35112	34485	35112	34913
17	0.07	0.15	150	30	2	34229	34430	34880	34311	34342
18	0.07	0.15	150	30	3	34234	34722	35008	34303	35039
19	0.08	0.13	250	30	1	34687	34930	34795	35078	34656
20	0.08	0.13	250	30	2	34209	34539	35104	35390	34209
21	0.08	0.13	250	30	3	34687	35382	35054	34342	34746
22	0.08	0.14	150	40	1	34833	34606	35008	34817	34459
23	0.08	0.14	150	40	2	34571	34581	34999	34656	34882
24	0.08	0.14	150	40	3	34238	34623	34229	35039	35078
25	0.08	0.15	200	20	1	34311	34913	34209	34930	34292
26	0.08	0.15	200	20	2	35167	35253	34571	34674	34406
27	0.08	0.15	200	20	3	34497	35026	35165	35203	35146

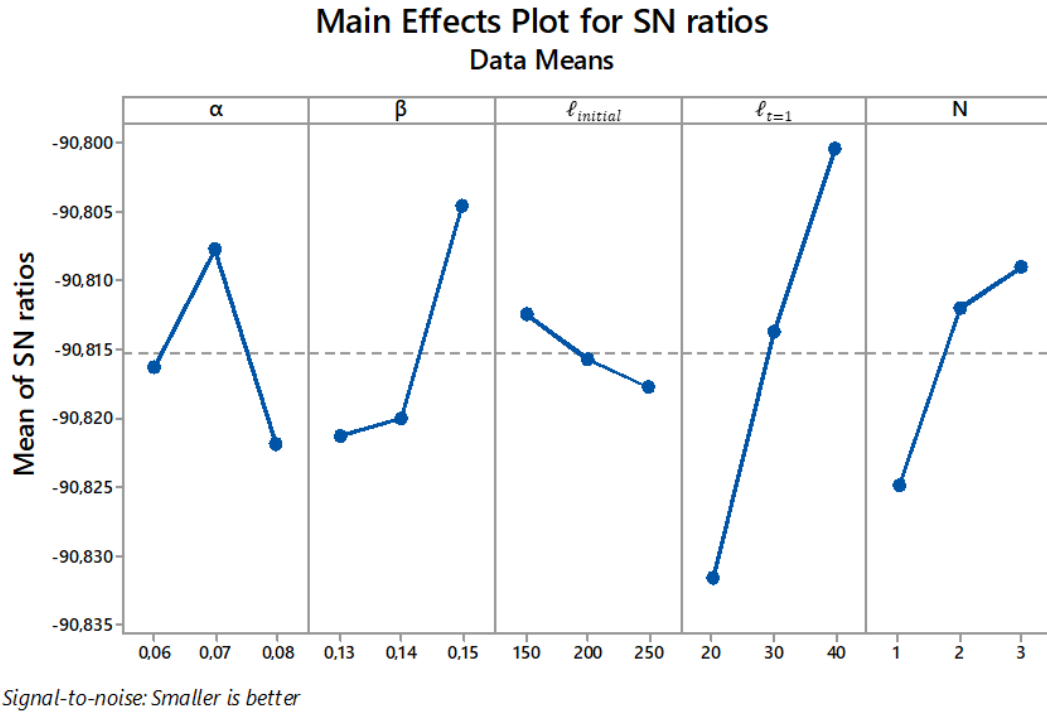


Figure 6.1 The main effects plot for S/N ratios

The main effects plot for S/N ratios is illustrated in Figure 6.1. The best parameter levels for the CP-based LNS are found to be $\alpha = 0.07$, $\beta = 0.15$, $\ell_{\text{initial}} = 150$, $\ell_{t=1} = 40$, and $N = 3$, which maximize the S/N ratio.

6.3 Computational Analysis

This section presents the effect of lot streaming on the FJSSP first, followed by comparing the CP model's performance with the CP-based LNS. The computational study is conducted on a personal computer with Intel(R)Core(TM) i5-760X CPU 4.00 GHz with 32 GB RAM. The CP model and the CP-based LNS are coded in C# on the Microsoft Visual Studio 2015 platform. For the CP model, IBM ILOG CP Optimizer 12.10 is used.

6.3.1 The Effect of Lot Streaming on the FJSSP

Lot streaming is effective in reducing time-based performance measures in scheduling problems. This section analyzes the effect of lot streaming on the FJSSP in terms of the makespan objective. The MFJS9:S125 instance is solved both with and without lot streaming. The Gantt chart for the resulting schedule is illustrated in Figure 6.2. The first Gantt chart, Figure 6.2(a), shows the solution of the FJSSP without lot streaming, while the second one, Figure 6.2(b), shows the solution of the FJSSP extended with lot streaming. According to the figures, we have achieved a 14.40% improvement of objective value when the problem is extended with lot streaming. As can be seen from the figures, lot streaming enhances machine utilization by permitting the operations of sublots to be scheduled in parallel. For example, *job 10* has been divided into three sublots, and each of them has four operations to be executed on machines. In Figure 6.2(a), it is observed that the operations of *job 10* are executed in the following route M4-M5-M5-M7 when the lot streaming is not considered. In Figure 6.2(b), the manufacturing routes are given as M4-M6-M5-M8 for *sublot 1*; M4-M5-M5-M8 for *sublot 2*; and M4-M6-M5-M7 for *sublot 3*. From these routes, it is observed that the second operations of these sublots are handled by two eligible

machines, which are M5 and M6, in parallel; and the fourth operations of the sublots are handled by two eligible machines, which are M7 and M8. As a result, the operations of the sublots belonging to the same job are scheduled in parallel on different eligible machines. This shows that the flexibility in the manufacturing system enhances the effects of lot streaming and machine utilization.

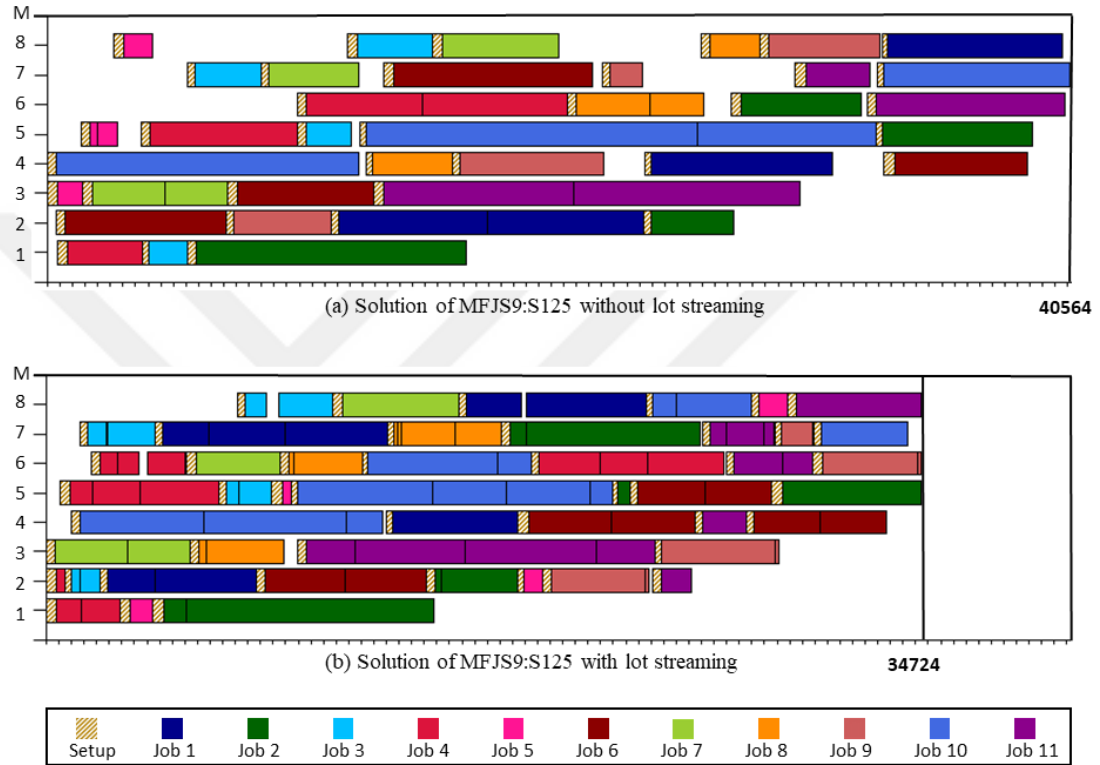


Figure 6.2 The Gantt chart for the resulting schedule of MFJS9:S125 without/with lot streaming

The maximum possible number of operations that can be scheduled in parallel is determined by the maximum number of sublots and eligible machines for that operation. Similarly, the maximum number of setup activities that can be scheduled in parallel is limited by the number of resources available for that activity. For example, the setup activity of *job 10* requires two units of resources, while the available number of resources is given as three for that activity. Therefore, the setup activities of *job 10* can only be overlapped with the other setup activities having a resource requirement of one. Moreover, setup activities are prone to be scheduled in different time frames so that they do not overlap with each other, and the setup activities and processing

operations scheduled on different machines are handled in parallel to satisfy the setup resource restrictions. Therefore, lot streaming uses manufacturing resources by overlapping processing operations, setup, and transport activities.

Furthermore, we have examined the effect of lot streaming with different scenarios determined for each extended benchmark instance. We have found that the improvements brought by lot streaming are reduced when the SDSTs increase. For medium-sized instances, it can be observed that as the SDSTs increase, the effect of lot streaming decreases. The largest improvements are achieved in scenarios S50 and S75 with average makespan reduction of 34.76% and 34.61%, respectively, while these reductions are 30.29% and 29.91% in scenarios S100 and S125, respectively. For large-sized instances, it is found that the effect of lot streaming is relatively small compared to medium-sized instances. These average makespan reductions are 20.53% and 16.16% in scenarios S50 and S75, respectively, compared to 16.66% and 17.27% in scenarios S100 and S125. Since the SDSTs and transport times increase, the effect of the lot streaming is reduced for scenarios S75, S100, and S125 for large-sized instances. The effect of lot streaming is greatly reduced for these instances because of the limited amount of setup and transport resources.

6.3.2 The Performance of the CP-based LNS

The performance of an algorithm is evaluated by its ability to converge to quality solutions in reasonable computation time. The convergence behavior of an algorithm presents the improvement of the solutions over time as the search proceeds. The convergence behavior of the proposed CP-based LNS is illustrated in Figure 6.3 for the MFJS2 data set, including four scenarios. The convergence curves are drawn to show the performance of the CP-based LNS within the first 20 minutes after obtaining the initial solution. We have observed that the performance of the CP model is quite effective in the first 150 seconds. The improvements of the objective value starting from the first solution to the best solution within 150 s. are 58.48%, 71.43%, 59.34%, and 66.49% for problems MFJS2:S50, MFJS2:S75, MFJS2:S100, and MFJS2:S125,

respectively. For this reason, we have set a time limit of 150 s. to the CP model to derive an initial solution and initialize the algorithm with a good starting point. Hence, the curves for the CP-based LNS are drawn after obtaining the initial solution to clearly show the improvements made by the CP-based LNS in the first 20 minutes.

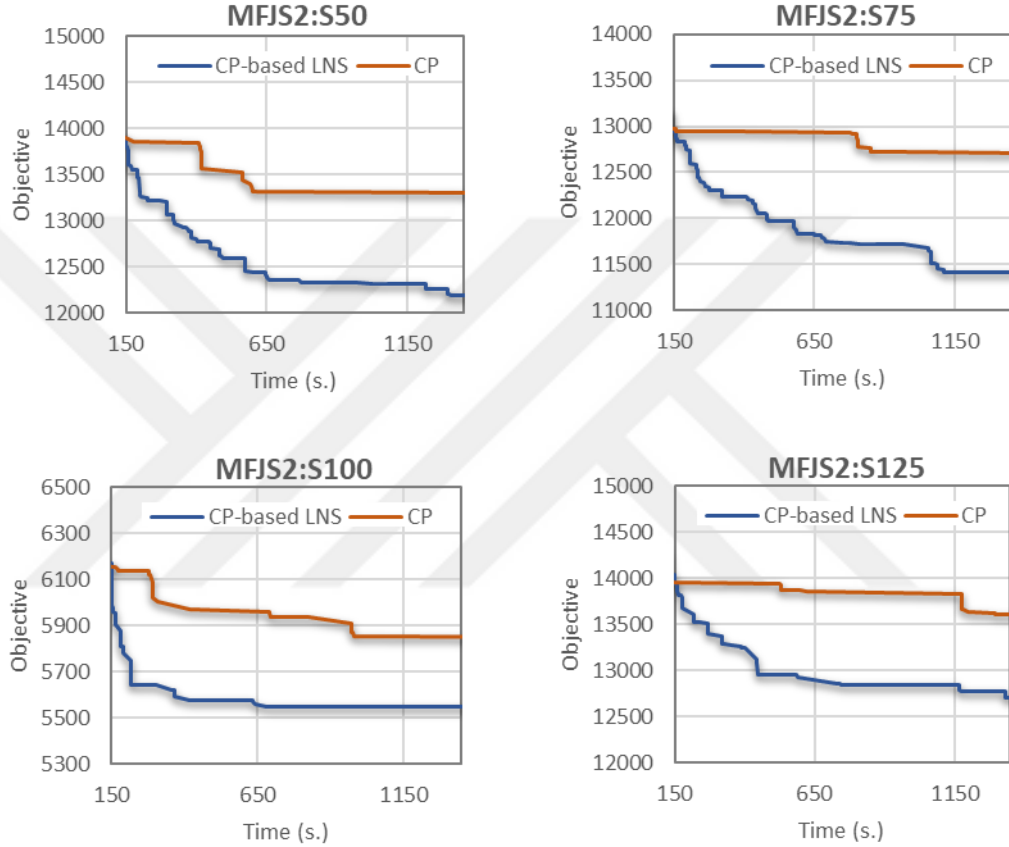


Figure 6.3 Convergence curves for the data set MFJS2 with four scenarios

As shown in Figure 6.3, it is difficult to find an improved solution using the CP model alone. Over time, the CP model gets stuck on a solution, and it takes a long time to instantiate an improved solution. However, the proposed CP-based LNS is capable of finding improved solutions in the same time frame. The reason behind this is the convergence behavior of the CP-based LNS. The CP-based LNS effectively converges solutions in less computation time than the CP model because the proposed approach

can cope with the complexity of the problem by solving subproblems with reduced solution space iteratively for the problem.

6.3.3 The Comparison of the Solution Approaches

The computational experiments are conducted for small-, medium-, and large-sized instances to compare the performance of the CP model and the CP-based LNS. The two solution approaches are executed within 3600 s. to provide a fair comparison. For the CP-based LNS, ten runs are performed per instance. The best and average objective values obtained within the time limit are reported. Moreover, the initial time for instantiating the best solutions found by the two solution approaches is also reported in the computational results. The CP-based LNS is compared with the CP model in terms of objective value and computation time. The improvement of the objective value found by the CP-based LNS compared to that of the CP model is calculated using Equation (6.5), where $CP_{basedLNS_{obj}}$ represents the objective value of the CP-based LNS, CP_{obj} represents the objective value of the CP model with the best branching strategy determined in Section 6.2.1.

$$Imp \% = \frac{CP_{obj} - CP_{basedLNS_{obj}}}{CP_{obj}} \times 100 \quad (6.5)$$

The computational results for small-sized instances are shown in Table 6.10. The solution of the CP model without lot streaming (no-LS) guarantees optimality for all small-sized instances. Lot streaming improves the objective value by 32.41% on average. The CP model without the proposed branching strategy finds feasible solutions for all small-sized instances except SFJS6:S75, SFJS6:S125, and SFJS9:S75. With the proposed branching strategy, the CP model can find feasible solutions for all instances and improve 7.55% of the objective value on average. Additionally, the proposed branching strategy has an advantage over the CP model with no branching strategy in terms of computation time for all instances except SFJS6:S50, as shown in Table 6.10. The average of the best improvements brought by the CP-based LNS is

rather low (0.81%) compared to those for medium-sized (5.29%) and large-sized (4.08%) instances because the CP model has already converged to the best solutions for most of the small-sized instances within 3600 s. time limit.



Table 6.10 Computational results for the small-sized instances

Instance	Problem Complexity		no-LS		with LS								Improvements			
			CP		CP (no-BS)		CP		CP-based LNS							
			Obj	Time	Obj	Time	Obj	Time	Best Obj	Avg Obj	Best Time	Avg Time	Imp(1) %	Imp(2) %	Imp(3) %	Imp(4) %
SFJS2:S50	112	348	1036	0.18	949	3.05	949	0.40	949	949.00	0.33	0.51	8.40	0.00	0.00	0.00
SFJS2:S75	92	286	2615	0.09	1938	26.50	1938	0.15	1905	1905.00	161.91	171.15	25.89	0.00	1.70	1.70
SFJS2:S100	92	286	2336	0.03	1842	3.14	1842	0.15	1842	1842.00	0.18	0.24	21.15	0.00	0.00	0.00
SFJS2:S125	72	224	3261	0.03	2449	3.09	2414	2.30	2414	2414.00	2.47	3.01	25.97	1.43	0.00	0.00
SFJS6:S50	231	800	7144	0.24	4400	132.55	4259	180.70	4225	4235.11	703.01	799.19	40.38	3.20	0.80	0.56
SFJS6:S75	261	898	5724	0.10	x	x	3935	934.05	3857	3886.00	287.42	1006.36	31.25	x	1.98	1.25
SFJS6:S100	231	800	12332	0.10	8711	1784.93	6721	329.13	6695	6711.83	1507.32	838.76	45.50	22.84	0.39	0.14
SFJS6:S125	197	678	8873	0.10	x	x	5280	61.94	5280	5280.00	57.65	66.72	40.49	x	0.00	0.00
SFJS9:S50	242	866	7922	0.09	6415	3229.64	5271	60.06	5271	5271.00	75.21	92.14	33.46	17.83	0.00	0.00
SFJS9:S75	242	866	4347	0.14	x	x	3147	844.09	3070	3104.20	201.49	1209.63	27.61	x	2.45	1.36
SFJS9:S100	174	622	4888	0.25	4223	275.53	4057	91.90	3988	4042.33	2490.43	1063.73	17.00	3.93	1.70	0.36
SFJS9:S125	208	744	5509	0.29	4969	3523.92	4770	1529.94	4600	4701.00	2139.96	968.78	13.41	4.00	3.56	1.45
Average								336.23			635.62	518.35	32.41	7.55	0.81	0.56

Time: the time required to instantiate the best objective value within 3600 s.; NV: number of model variables; NC: number of constraints; no-LS: no lot streaming; no-BS: no branching strategy; BS: branching strategy; Imp(1)%: percent improvement in objective value with lot streaming extension; Imp(2)%: percent improvement in objective value with branching strategy; Imp(3)%: percent improvement in the best objective value achieved by CP-based LNS compared to CP Model (with BS); Imp(4)%: percent improvement in the average objective value achieved by CP-based LNS compared to CP Model (with BS); best results in **bold**

Since the problem is strongly NP-hard, it is especially difficult for medium- and large-sized instances to find a feasible schedule. For these instances, the CP model could not find any feasible solution within the 3600 s. time limit. Owing to the proposed branching strategy, the CP model can find feasible solutions. Note that for all instances, the first feasible solutions are instantiated in less than a minute. However, the CP model with the proposed branching strategy could not guarantee the optimality for these instances and has reached the time limit of 3600 s.

The computational results for medium-sized instances are shown in Table 6.11. Lot streaming improves the objective value by 32.39% on average. The CP-based LNS provides an average of 5.29% improvement in the objective value considering the best solution results. The average solution time of the CP model is 1928.03, while the average solution times of the CP-based LNS is 2045.40. In addition, the average scenario improvements are further analyzed. It is observed that the CP-based LNS provides better results with smaller setup time scenarios S50 and S75 compared to solutions obtained by the CP model. The CP-based LNS improves the solutions for these scenarios by 6.90% and 6.21% on average, considering the best solution values. On the other hand, the improvements are lower for larger setup time scenarios S100 and S125, being 3.73% and 4.31%, respectively. These scenarios cover the most difficult instances to be solved in medium-sized instances. Furthermore, the average solution times of the CP-based LNS are 2336.76, 2256.42, 2187.45, and 1740.30, respectively for the scenarios S50, S75, S100, and S125.

Table 6.11 Computational results for the medium-sized instances

Instance	Problem Complexity		no-LS		with-LS					Improvements		
			CP Model		CP Model		CP-based LNS					
	NV	NC	Obj	Time	Obj	Time	Best Obj	Avg Obj	Avg Time			
MFJS1:S50	426	1563	19139	0.33	11063	2849.31	10086	10501.60	1924.50	47.30	8.83	5.07
MFJS1:S75	426	1563	17417	0.25	12318	685.30	11567	11706.29	1800.78	33.59	6.10	4.97
MFJS1:S100	360	1335	14113	0.65	10174	620.60	9875	10002.43	1662.43	30.03	2.94	1.69
MFJS1:S125	462	1697	19432	4.35	14044	1464.66	13208	13461.29	1272.36	32.03	5.95	4.15
MFJS2:S50	473	1871	18500	2.97	13295	1450.42	12021	12335.14	2320.92	35.02	9.58	7.22
MFJS2:S75	471	1855	19438	2.99	12322	2833.30	11371	11735.57	2537.61	41.50	7.72	4.76
MFJS2:S100	431	1689	7623	1.99	5846	1883.46	5548	5628.50	1139.20	27.22	5.10	3.72
MFJS2:S125	469	1841	18493	1.10	13383	3413.11	12649	12778.86	2450.89	31.60	5.48	4.51
MFJS3:S50	571	2273	18553	0.39	11895	1044.27	10930	11070.00	2300.39	41.09	8.11	6.94
MFJS3:S75	457	1819	18457	0.30	11070	191.79	10409	10682.11	1757.04	43.60	5.97	3.50
MFJS3:S100	571	2273	19507	2.09	14256	2080.16	13877	14053.75	2208.52	28.86	2.66	1.42
MFJS3:S125	497	1985	13915	0.42	9957	1472.22	9605	9759.88	2332.34	30.97	3.54	1.98
MFJS4:S50	592	2368	20686	2.30	15067	2205.19	14601	14797.75	1892.80	29.42	3.09	1.79
MFJS4:S75	552	2202	20290	4.28	14161	2460.53	13406	13592.14	1510.25	33.93	5.33	4.02
MFJS4:S100	630	2520	21543	7.80	16298	1176.16	15469	15812.50	2070.73	28.19	5.09	2.98
MFJS4:S125	552	2202	25042	0.52	16114	3480.43	15703	15849.00	2286.40	37.29	2.55	1.64
MFJS5:S50	587	2317	14723	0.54	12235	890.03	11637	11858.29	2439.24	20.96	4.89	3.08
MFJS5:S75	657	2573	13763	5.04	11640	3534.89	10948	11296.50	2613.17	20.45	5.95	2.95
MFJS5:S100	663	2619	17699	1.31	11452	2733.77	11126	11309.88	2156.55	37.14	2.85	1.24
MFJS5:S125	589	2331	15749	7.35	13514	2090.96	12972	13272.75	2231.85	17.63	4.01	1.79
Average						1928.03			2045.40	32.39	5.29	3.47
Time: the time required to instantiate the best objective value within 3600 s; NV: number of model variables; NC: number of constraints; no-LS: no lot streaming; BS: branching strategy; Imp(1)%: percent improvement in objective value with lot streaming extension; Imp(2)%: percent improvement in the best objective value achieved by CP-based LNS; Imp(3)%: percent improvement in the average objective value achieved by CP-based LNS; best results in bold												

The computational results for large-sized instances are shown in Table 6.12. Lot streaming improves the objective value by 17.65% on average. The effect of the lot streaming is rather low because the large-sized instances belong to the hardest class of problems in the computational experiments. The CP-based LNS provides an average 4.08% improvement of the objective value, considering the best solution values. Additionally, the CP-based LNS has an advantage over the CP model in terms of computation time to reach the best solution values obtained for the large-sized instances. The average solution time required to instantiate the best solution is 2868.54 for the CP model and 2872.12 for the CP-based LNS respectively. The average improvements for scenarios S50 and S125 are 4.66% and 4.35%, respectively. The improvements are lower for scenarios S75 and S100, at 3.57% and 3.74%, respectively. Therefore, it is observed that the problem gets harder as the processing time and SDST get closer to each other for the large-sized instances.

Table 6.12 Computational results for the large-sized instances

Instance	Problem Complexity		no-LS		with LS					Improvements		
			CP Model		CP Model		CP-based LNS					
	NV	NC	Obj	Time	Obj	Time	Best Obj	Avg Obj	Avg Time	Imp(1)%	Imp(2)%	Imp(3)%
MFJS6:S50	681	2687	27929	1.62	18452	2267.29	17332	17606.75	2724.84	37.94	6.07	4.58
MFJS6:S75	565	2221	22307	120.98	18884	915.02	18252	18637.20	2382.47	18.18	3.35	1.31
MFJS6:S100	787	3077	22936	1.43	17039	2217.36	16227	16593.50	2583.07	29.25	4.77	2.61
MFJS6:S125	751	2943	18635	116.41	15977	2870.51	15113	15627.00	2582.69	18.90	5.41	2.19
MFJS7:S50	813	3267	35353	58.89	31066	3592.64	29574	30240.40	2365.65	16.35	4.80	2.66
MFJS7:S75	771	3127	29791	1.75	22839	2913.19	22030	22292.20	1636.76	26.05	3.54	2.39
MFJS7:S100	769	3111	25811	144.66	22619	705.75	21984	22213.60	2499.96	14.83	2.81	1.79
MFJS7:S125	917	3707	37813	1.79	28488	3388.99	27726	28137.50	2877.79	26.68	2.67	1.23
MFJS8:S50	955	3845	24160	130.17	20411	3317.94	19547	19949.67	3234.38	19.09	4.23	2.26
MFJS8:S75	855	3439	29590	376.30	26087	2920.16	25331	25576.00	2859.72	14.39	2.90	1.96
MFJS8:S100	853	3421	31964	4.33	29184	3385.33	28211	28577.75	3095.95	11.74	3.33	2.08
MFJS8:S125	907	3659	28630	43.19	25221	2676.38	23464	24586.00	3228.07	18.04	6.97	2.52
MFJS9:S50	1236	4927	32019	102.36	28344	3406.65	27244	27786.25	3542.45	14.91	3.88	1.97
MFJS9:S75	948	3797	27079	15.04	23784	3420.06	22932	23115.80	3017.03	15.31	3.58	2.81
MFJS9:S100	1144	4583	33264	65.65	30710	3230.90	29101	29722.20	3131.84	12.52	5.24	3.22
MFJS9:S125	1144	4581	40564	111.11	36213	3370.43	34724	35320.40	3470.16	14.40	4.11	2.46
MFJS10:S50	1222	4917	34378	14.82	30782	2844.05	29451	29778.00	3498.61	14.33	4.32	3.26
MFJS10:S75	1258	5009	36669	1880.79	35751	3513.55	34149	34805.40	3428.50	6.87	4.48	2.64
MFJS10:S100	1308	5211	33565	93.74	29300	2857.02	28550	28819.75	2773.54	14.94	2.56	1.64
MFJS10:S125	1210	4821	44632	497.59	41990	3557.50	40908	41223.00	2508.83	8.34	2.58	1.83
Average						2868.54			2872.12	17.65	4.08	2.37
Time: the time required to instantiate the best objective value within 3600 s; NV: number of model variables; NC: number of constraints; no-LS: no lot streaming; BS: branching strategy; Imp(1)%: percent improvement in objective value with lot streaming extension; Imp(2)%: percent improvement in the best objective value achieved by CP-based LNS; Imp(3)%: percent improvement in the average objective value achieved by CP-based LNS; best results in bold												

CHAPTER 7

CONCLUSION

This thesis focuses on the FJSSP-LS with additional resource constraints for setup and transport activities. Consistent sublots, subplot-attached sequence-dependent setups, and transport times are considered for scheduling decisions. The studied problem belongs to the hardest class of scheduling problems and is strongly NP-hard in nature. Therefore, we propose a novel CP model and CP-based LNS for solving this constrained optimization problem. The CP model effectively captures the combinatorial part of the problem by using custom variable/value definers and specific constraints designed to address scheduling problems. Moreover, the proposed CP is enhanced by an efficient branching strategy to find quality feasible solutions in a reasonable computation time. However, as the search progresses, the CP model remains stuck with a solution and cannot fully explore the problem's search space, especially in the most difficult instances. Consequently, we propose the CP-based LNS, which can improve the quality of the solutions obtained. This method explores the search space using a CP model to get solutions in large neighborhoods by applying small perturbations. Moreover, we propose adaptive mechanisms for the parameters of the CP-based LNS to enhance its performance.

The proposed CP-based LNS converges to the quality solutions in a reasonable computation time compared to the novel CP model. The computational study is performed for small-, medium-, and large-sized instances extended from the well-known FJSSP benchmarks. The effect of the lot streaming is analyzed, and the performance of the two proposed solution methodologies is compared in terms of objective value and computation time. It is found that the CP-based LNS is effective in guiding the search progress to promising regions and outperforms the novel CP model with 4.68% improvement for medium- and large-sized instances on average. Future research should consider different subplot types, i.e. variable sublots, in a flexible job shop manufacturing environment. Furthermore, the setup and transport costs and intermediate storage can also be included in the FJSSP-LS to reflect real-world flexible job shop manufacturing systems.

REFERENCES

- Abdelmaguid, T. F. (2015). A neighborhood search function for flexible job shop scheduling with separable sequence-dependent setup times. *Applied Mathematics and Computation*, 260, 188-203.
- Abdollahzadeh Sangroudi, H., & Ranjbar-Bourani, M. (2021). Solving a flexible job shop lot sizing problem with shared operations using a self-adaptive COA. *International Journal of Production Research*, 59(2), 483-515.
- Ahmadizar, F., & Shahmaleki, P. (2014). Group-shop scheduling with sequence-dependent set-up and transportation times. *Applied Mathematical Modelling*, 38(21-22), 5080-5091.
- Åstrand, M., Johansson, M., & Zanarini, A. (2020). Underground mine scheduling of mobile machines using Constraint Programming and Large Neighborhood Search. *Computers & Operations Research*, 123, 105036.
- Bagheri, A., & Zandieh, M. (2011). Bi-criteria flexible job-shop scheduling with sequence-dependent setup times—Variable neighborhood search approach. *Journal of Manufacturing Systems*, 30(1), 8-15.
- Beck, J. C., Feng, T. K., & Watson, J. P. (2011). Combining constraint programming and local search for job-shop scheduling. *INFORMS Journal on Computing*, 23(1), 1-14.
- Blum, C., Puchinger, J., Raidl, G. R., & Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6), 4135-4151.
- Božek, A., & Werner, F. (2018). Flexible job shop scheduling with lot streaming and subplot size optimisation. *International Journal of Production Research*, 56(19), 6391-6411.

- Bożek, A., & Wysocki, M. (2016). Off-line and dynamic production scheduling-A comparative case study. *Management and Production Engineering Review*, 7, 21-32.
- Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, 45(4), 369-375.
- Buscher, U., & Shen, L. (2009). An integrated tabu search algorithm for the lot streaming problem in job shops. *European Journal of Operational Research*, 199(2), 385-399.
- Carchrae, T., & Beck, J. C. (2009). Principles for the design of large neighborhood search. *Journal of Mathematical Modelling and Algorithms*, 8(3), 245-270.
- Chan, F., Wong, T., & Chan, L. (2006). Flexible job-shop scheduling problem under resource constraints. *International Journal of Production Research*, 44(11), 2071-2089.
- Chan, F., Wong, T., & Chan, L. (2008). Lot streaming for product assembly in job shop environment. *Robotics and Computer-Integrated Manufacturing*, 24(3), 321-331.
- Chan, F. T., Wong, T., & Chan, L. (2009a). The application of genetic algorithms to lot streaming in a job-shop scheduling problem. *International Journal of Production Research*, 47(12), 3387-3412.
- Chan, F. T., Wong, T. C., & Chan, L. (2009b). An evolutionary algorithm for assembly job shop with part sharing. *Computers & Industrial Engineering*, 57(3), 641-651.
- Chang, J. H., & Chiu, H. N. (2005). A comprehensive review of lot streaming. *International Journal of Production Research*, 43(8), 1515-1536.

- Chaudhry, I. A., & Khan, A. A. (2016). A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23(3), 551-591.
- Cheng, M., Mukherjee, N., & Sarin, S. (2013). A review of lot streaming. *International Journal of Production Research*, 51(23-24), 7023-7046.
- Cormier, D., O'Grady, P., & Sanii, E. (1998). A constraint-based genetic algorithm for concurrent engineering. *International Journal of Production Research*, 36(6), 1679-1697.
- Dai, M., Tang, D., Giret, A., & Salido, M. A. (2019). Multi-objective optimization for energy-efficient flexible job shop scheduling problem with transportation constraints. *Robotics and Computer-Integrated Manufacturing*, 59, 143-157.
- Daneshamooz, F., Fattahi, P., & Hosseini, S. M. H. (2021). Scheduling in a flexible job shop followed by some parallel assembly stations considering lot streaming. *Engineering Optimization*, 1-20.
- Dauzere-Peres, S., & Lasserre, J.-B. (1993). An iterative procedure for lot streaming in job-shop scheduling. *Computers & Industrial Engineering*, 25(1-4), 231-234.
- Dauzere-Peres, S., & Lasserre, J.-B. (1997). Lot streaming in job-shop scheduling. *Operations Research*, 45(4), 584-595.
- Defersha, F. M., & Chen, M. (2010). A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups. *The International Journal of Advanced Manufacturing Technology*, 49(1), 263-279.
- Defersha, F. M., & Chen, M. (2012). Jobshop lot streaming with routing flexibility, sequence-dependent setups, machine release dates and lag time. *International Journal of Production Research*, 50(8), 2331-2352.

- Defersha, F. M., & Movahed, S. B. (2018). Linear programming assisted (not embedded) genetic algorithm for flexible jobshop scheduling with lot streaming. *Computers & Industrial Engineering*, 117, 319-335.
- Defersha, F. M., & Rooyani, D. (2020). An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time. *Computers & Industrial Engineering*, 147, 106605.
- Deris, S., Omatu, S., Ohta, H., & Saad, P. (1999). Incorporating constraint propagation in genetic algorithm for university timetable planning. *Engineering applications of artificial intelligence*, 12(3), 241-253.
- Du, Y., Li, J. Q., Luo, C., & Meng, L. L. (2021). A hybrid estimation of distribution algorithm for distributed flexible job shop scheduling with crane transportations. *Swarm and Evolutionary Computation*, 62, 100861.
- Fanjul-Peyro, L. (2020). Models and an exact method for the Unrelated Parallel Machine scheduling problem with setups and resources. *Expert Systems with Applications: X*, 5, 100022.
- Fattahi, P., Mehrabad, M. S., & Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 18(3), 331-342.
- Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2), 117-129.
- Glass, C., Gupta, J., & Potts, C. (1994). Lot streaming in three-stage production processes. *European Journal of Operational Research*, 75(2), 378-394.

- Güçdemir, H., & Selim, H. (2018). Integrating simulation modelling and multi criteria decision making for customer focused scheduling in job shops. *Simulation Modelling Practice and Theory*, 88, 17-31.
- Hojabari, H., Gendreau, M., Potvin, J. Y., & Rousseau, L. M. (2018). Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. *Computers & Operations Research*, 92, 87-97.
- IBM. (2019a). *CP optimizer user's manual*. Retrieved April 30, 2021, from <https://www.ibm.com/docs/en/icos/12.10.0?topic=optimizer-cp-users-manual>.
- IBM. (2019b). *Scheduling search phases*. Retrieved April 30, 2021, from <https://www.ibm.com/docs/en/icos/12.10.0?topic=phases-scheduling-search>.
- IBM. (2021a). *IBM ILOG CPLEX optimization studio documentation: CumulFunction*. Retrieved June 15, 2021, from <https://www.ibm.com/docs/tr/icos/12.10.0?topic=keywords-cumulfunction>.
- IBM. (2021b). *IBM ILOG CPLEX Optimization studio documentation: CP.ImpactOfLastBranch Method*. Retrieved June 15, 2021, from <https://www.ibm.com/docs/en/icos/12.10.0?topic=cm-impactoflastbranch-method>.
- IBM. (2021c). *IBM ILOG CPLEX optimization studio documentation: ISolution interface*. Retrieved June 15, 2021, from <https://www.ibm.com/docs/en/icos/12.10.0?topic=namespace-isolution-interface>.
- Jussien, N., & Lhomme, O. (2002). Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1), 21-45.

- Kacem, I., Hammadi, S., & Borne, P. (2002). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 32(1), 1-13.
- Kacprzyk, J., & Pedrycz, W. (2015). *Springer handbook of computational intelligence* (1th ed.). Heidelberg: Springer.
- Karimi, S., Ardalan, Z., Naderi, B., & Mohammadi, M. (2017). Scheduling flexible job-shops with transportation times: Mathematical models and a hybrid imperialist competitive algorithm. *Applied Mathematical Modelling*, 41, 667-682.
- Kesen, S. E., & Güngör, Z. (2012). Job scheduling in virtual manufacturing cells with lot-streaming strategy: a new mathematical model formulation and a genetic algorithm approach. *Journal of the Operational Research Society*, 63(5), 683-695.
- Kilby, P., & Urli, T. (2016). Fleet design optimisation from historical data using constraint programming and large neighbourhood search. *Constraints*, 21(1), 2-21.
- Kress, D., Müller, D., & Nossack, J. (2019). A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. *OR Spectrum*, 41(1), 179-217.
- Kreter, S., Schutt, A., Stuckey, P. J., & Zimmermann, J. (2018). Mixed-integer linear programming and constraint programming formulations for solving resource availability cost problems. *European Journal of Operational Research*, 266(2), 472-486.
- Lei, D., & Guo, X. (2013). Scheduling job shop with lot streaming and transportation through a modified artificial bee colony. *International Journal of Production Research*, 51(16), 4930-4941.

- Li, J. Q., Deng, J. W., Li, C. Y., Han, Y. Y., Tian, J., Zhang, B., & Wang, C. G. (2020). An improved Jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times. *Knowledge-Based Systems*, 200, 106032.
- Li, M., & Lei, D. (2021). An imperialist competitive algorithm with feedback for energy-efficient flexible job shop scheduling with transportation and sequence-dependent setup times. *Engineering Applications of Artificial Intelligence*, 103, 104307.
- Liu, C.-H. (2009). Lot streaming for customer order scheduling problem in job shop environments. *International Journal of Computer Integrated Manufacturing*, 22(9), 890-907.
- Liu, C. H., Chen, L. S., & Lin, P. S. (2013). Lot streaming multiple jobs with values exponentially deteriorating over time in a job-shop environment. *International Journal of Production Research*, 51(1), 202-214.
- Meng, T., & Pan, Q. K. (2018, August). An enhanced migrating birds optimization for the flexible job shop scheduling problem with lot streaming. In *International Conference on Intelligent Computing* (769-779). Springer, Cham.
- Mousakhani, M. (2013). Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness. *International Journal of Production Research*, 51(12), 3476-3487.
- Nguyen, S., Thiruvady, D., Zhang, M., & Tan, K. C. (2021). A genetic programming approach for evolving variable selectors in constraint programming. *IEEE Transactions on Evolutionary Computation*, 25(3), 492-507.
- Nouri, H. E., Driss, O. B., & Ghédira, K. (2016). Simultaneous scheduling of machines and transport robots in flexible job shop environment using hybrid metaheuristics

- based on clustered holonic multiagent model. *Computers & Industrial Engineering*, 102, 488-501.
- Novas, J. M. (2019). Production scheduling and lot streaming at flexible job-shops environments using constraint programming. *Computers & Industrial Engineering*, 136, 252-264.
- Özgüven, C., Yavuz, Y., & Özbakır, L. (2012). Mixed integer goal programming models for the flexible job-shop scheduling problems with separable and non-separable sequence dependent setup times. *Applied Mathematical Modelling*, 36(2), 846-858.
- Palpant, M., Artigues, C., & Michelon, P. (2004). LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131(1), 237-257.
- Pesant, G., & Gendreau, M. (1999). A constraint programming framework for local search methods. *Journal of Heuristics*, 5(3), 255-279.
- Prestwich, S. (2008). Generalised graph colouring by a hybrid of local search and constraint programming. *Discrete Applied Mathematics*, 156(2), 148-158.
- Gendreau, M., & Potvin, J. Y. (2010). *Handbook of metaheuristics* (2nd ed.). New York: Springer.
- Rossi, A. (2014). Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships. *International Journal of Production Economics*, 153, 253-267.

- Rossi, R., Tarim, S. A., & Bollapragada, R. (2012). Constraint-based local search for inventory control under stochastic demand and lead time. *INFORMS journal on computing*, 24(1), 66-80.
- Rossi, F., Van Beek, P., & Walsh, T. (2006). *Handbook of constraint programming* (1th ed.). The Netherlands: Elsevier.
- Ruiz, R., & Maroto, C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research*, 169(3), 781-800.
- Ruiz, R., & Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3), 1143-1159.
- Saidi-Mehrabad, M., & Fattahi, P. (2007). Flexible job shop scheduling with tabu search algorithms. *The International Journal of Advanced Manufacturing Technology*, 32(5-6), 563-570.
- Shen, L., Dauzère-Pérès, S., & Neufeld, J. S. (2018). Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 265(2), 503-516.
- Sioud, A., Gravel, M., & Gagné, C. (2012). A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, 39(10), 2415-2424.
- Sun, J., Zhang, G., Lu, J., & Zhang, W. (2021). A hybrid many-objective evolutionary algorithm for flexible job-shop scheduling problem with transportation and setup times. *Computers & Operations Research*, 105263.

- Taguchi, G. (1986). *Introduction to quality engineering: designing quality into products and processes* (1th ed.). United States: Wiley-Interscience.
- Tamssaouet, K., Dauzère-Pérès, S., Knopp, S., Bitar, A., & Yugma, C. (2021). Multiobjective Optimization for Complex Flexible Job-Shop Scheduling Problems. *European Journal of Operational Research*.
- Thiruvady, D. R., Meyer, B., & Ernst, A. (2011, July). Car sequencing with constraint-based ACO. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, 163-170.
- Thiruvady, D., Singh, G., Ernst, A. T., & Meyer, B. (2013). Constraint-based ACO for a shared resource constrained scheduling problem. *International Journal of Production Economics*, 141(1), 230-242.
- Topaloglu, S., & Ozkarahan, I. (2011). A constraint programming-based solution approach for medical resident scheduling problems. *Computers & Operations Research*, 38(1), 246-255.
- Valouxis, C., & Housos, E. (2003). Constraint programming approach for school timetabling. *Computers & Operations Research*, 30(10), 1555-1572.
- Wang, S. M., Chen, J. C., & Wang, K. J. (2007). Resource portfolio planning of make-to-stock products using a constraint programming-based genetic algorithm. *Omega*, 35(2), 237-246.
- Wong, T. C., Chan, F. T., & Chan, L. (2009). A resource-constrained assembly job shop scheduling problem with Lot Streaming technique. *Computers & Industrial Engineering*, 57(3), 983-995.

- Wong, T. C., & Ngan, S.-C. (2013). A comparison of hybrid genetic algorithm and hybrid particle swarm optimization to minimize makespan for assembly job shop. *Applied Soft Computing*, 13(3), 1391-1399.
- Xiuli, W. U., Junjian, P. E. N. G., Zirun, X. I. E., Ning, Z. H. A. O., & Shaomin, W. U. (2021). An improved multi-objective optimization algorithm for solving flexible job shop scheduling problem with variable batches. *Journal of Systems Engineering and Electronics*, 32(2), 272-285.
- Yan, J., Liu, Z., Zhang, C., Zhang, T., Zhang, Y., & Yang, C. (2021). Research on flexible job shop scheduling under finite transportation conditions for digital twin workshop. *Robotics and Computer-Integrated Manufacturing*, 72, 102198.
- Yepes-Borrero, J. C., Perea, F., Ruiz, R., & Villa, F. (2021). Bi-objective parallel machine scheduling with additional resources during setups. *European Journal of Operational Research*, 292(2), 443-455.
- Yepes-Borrero, J. C., Villa, F., Perea, F., & Caballero-Villalobos, J. P. (2020). GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. *Expert Systems with Applications*, 141, 112959.
- Yunusoglu, P., & Topaloglu Yildiz, S. (2021). Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 1-18.
- Zhang, G., Hu, Y., Sun, J., & Zhang, W. (2020). An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. *Swarm and Evolutionary Computation*, 54, 100664.

- Zhang, Q., Manier, H., & Manier, M. A. (2012). A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times. *Computers & Operations Research*, 39(7), 1713-1723.
- Zhang, C., Wang, K., Ma, Q., Li, X., & Gao, L. (2021, May). A Discrete Grey Wolf Optimizer for Solving Flexible Job Shop Scheduling Problem with Lot-streaming. In *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (pp. 969-974). IEEE.
- Zhang, Z., Wu, L., Peng, T., & Jia, S. (2019). An improved scheduling approach for minimizing total energy consumption and makespan in a flexible job shop environment. *Sustainability*, 11(1), 179.
- Zhang, Q., & Zhang, C. (2018). An improved ant colony optimization algorithm with strengthened pheromone updating mechanism for constraint satisfaction problem. *Neural Computing and Applications*, 30(10), 3209-3220.