METHODS FOR HYBRID FLOW SHOP SCHEDULING AND A CASE STUDY IN AN AEROSPACE COMPANY

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

YİĞİTALP ÖZMEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

SEPTEMBER 2019

Approval of the thesis:

METHODS FOR HYBRID FLOW SHOP SCHEDULING AND A CASE STUDY IN AN AEROSPACE COMPANY

submitted by **YİĞİTALP ÖZMEN** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Yasemin Serin
Head of Department, **Industrial Engineering** _____

Assoc. Prof. Dr. Sedef Meral
Supervisor, **Industrial Engineering, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Canan Şerbetçioğlu
Industrial Engineering, METU _____

Assoc. Prof. Dr. Sedef Meral
Industrial Engineering, METU _____

Prof. Dr. Meral Azizoğlu
Industrial Engineering, METU _____

Assoc. Prof. Dr. Ferda Can Çetinkaya
Industrial Engineering, Çankaya University _____

Assist. Prof. Dr. Mustafa Kemal Tural
Industrial Engineering, METU _____

Date: 16.09.2019

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Yiğitalp Özmen

Signature:

ABSTRACT

METHODS FOR HYBRID FLOW SHOP SCHEDULING AND A CASE STUDY IN AN AEROSPACE COMPANY

Özmen, Yiğitalp
Master of Science, Industrial Engineering
Supervisor: Assoc. Prof. Dr. Sedef Meral

September 2019, 157 pages

In this study, we address the scheduling problem in Hybrid Flow Shop (HFS) with makespan objective. Since this problem is known to be NP-hard and an HFS is a common environment in real-life manufacturing systems, several approximate solution approaches have been proposed in the literature. Hence, we resort to some of these such as MILP model, dispatching rules, Palmer, CDS, NEH, and Bottleneck Heuristic. Due to the complexity of HFS scheduling problem, MILP model provides only a near optimal solution by using CPLEX for the real problem which we are inspired by the scheduling problem in the manufacturing of fuselage panels at an aerospace company as a case study whose current hybrid job shop is converted to an HFS by discrete event simulation to improve the output quality, lessen materials handling and shorten the manufacturing lead time. The job sequences of these approaches are simulated to compute makespan values of HFS scheduling problem. Moreover, we propose a Constraint Programming (CP) model for solving HFS scheduling problem to optimality for the real problem and test problems. We also propose a Hybrid Algorithm (HA) and a Galactic Swarm Optimization (GSO) in order not to be stuck in local optima for most of the test problems and to solve the real problem for optimality within an acceptable computational time. While HA and GSO

seem to be promising for solving most of the test problems to optimality, the CP model outperforms the other approaches in the literature by solving all of them to optimality.

Keywords: Aerospace Industry, Constraint Programming, Galactic Swarm Optimization, Hybrid Flow Shop, Scheduling

ÖZ

MELEZ AKIŞ TİPİ ATÖLYE ÇİZELGELEMESİ İÇİN YÖNTEMLER VE BİR HAVACILIK FİRMASINDA VAKA ANALİZİ

Özmen, Yiğitalp
Yüksek Lisans, Endüstri Mühendisliği
Tez Danışmanı: Doç. Dr. Sedef Meral

Eylül 2019, 157 sayfa

Bu çalışmada, Melez Akış-tipi Atölye (MAA) çizelgeleme problemini en büyük tamamlanma zamanı amacı ile ele aldık. Bu problemin NP-zor ve bir MAA'nın gerçek yaşam imalat sistemlerinde yaygın bir ortam olmasından dolayı literatürde birçok yaklaşık çözüm yaklaşımları önerilmiştir. Dolayısıyla, KTDP modeli, dağıtım kuralları, Palmer, CDS, NEH ve Darboğaz Sezgiseli gibi bunlardan bazılarına başvurduk. MAA çizelgeleme probleminin karmaşık olmasından dolayı KTDP CPLEX kullanarak gerçek problem için yalnızca optimale yakın bir sonuç sağlamıştır. Mevcut durumdaki melez sipariş-tipi atölyesi çıktı kalitesini arttırmak, malzeme elleçlemesini azaltmak ve üretim temin süresini kısaltmak için ayrık olaylı benzetim aracılığıyla MAA'ya dönüştürülen bu vakayı bir havacılık firmasında üretilmekte olan orta gövde panellerinin çizelgeleme probleminden ilham aldık. MAA çizelgeleme probleminin en büyük tamamlanma zamanı değerlerinin hesaplanabilmesi için bu çözüm yöntemlerinin iş sıralamaları simüle edilmiştir. Bundan başka, gerçek problem ve test problemlerini optimal olarak çözebilen bir kesin yöntem olan Kısıt Programlama (KP) modeli önerilmiştir. Ayrıca, test problemlerinin büyük bir bölümünün yerel optimalde sıkışmamasını sağlayacak ve gerçek problemi makul bir çözüm süresi içerisinde optimale yakın bir sonuç verecek şekilde çözebilen, bir Melez Algoritma (MA) ve bir Galaktik Sürü Optimizasyonu (GSO) önerilmiştir. MA ve

GSO, test problemlerinin büyük bir bölümünü optimal olarak çözebildiği için umut verici görünmekte iken, KP modeli, bunların tamamını optimal olarak çözdüğü için literatürdeki diğer çözüm yöntemlerinden daha iyi bir performans göstermiştir.

Anahtar Kelimeler: Havacılık Sanayii, Kısıt Programlama, Galaktik Sürü Optimizasyonu, Melez Akış-tipi Atölye, Çizelgeleme

To my beloved mother

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor Assoc. Prof. Dr. Sedef Meral for her kind guidance, encouragement and advice throughout the study. I would like to express my sincere appreciation for her motivation and friendly support in completing the research.

I also appreciate the support of the engineers of Sheet Metal Manufacturing Planning, my colleague, Mr. Semih Karatokuş, and my managers, Mr. Mehmet Şahan and Mr. Arif Köksal for their patience.

I express my gratitude to my father İzzet Özmen for his endless love, moral support, encouragement, and mentorship throughout my whole life.

Finally, I would like to thank my wife Gamze Özmen who is always right beside me whenever I need. This study would not have been completed without her support, motivation and everlasting love.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGEMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xiii

LIST OF FIGURES xvi

LIST OF ABBREVIATIONS xix

CHAPTERS

1. INTRODUCTION 1

2. LITERATURE REVIEW 5

2.1. Framework and Notation 5

2.2. Review of the HFS Scheduling Studies 12

2.3. Summary of the Survey 19

3. PROBLEM DESCRIPTION 23

4. HFS SCHEDULING: MILP MODEL AND HEURISTICS 27

4.1. Mixed-Integer Linear Programming (MILP) Model 27

4.2. Dispatching Rules 31

4.3. Renowned Heuristic Algorithms 33

5. PROPOSED SOLUTION METHODS 47

5.1. Constraint Programming (CP) Model 48

5.2. Hybrid Algorithm (HA) 56

5.2.1. Global Lower Bound (GLB) 56

5.2.2. First Available Machine (FAM) and Earliest Completion Time (ECT) Strategies 61
5.2.3. Pseudo Code and Flowchart of the Hybrid Algorithm (HA) 63
5.3. Galactic Swarm Optimization (GSO) 67
5.3.1. GSO in HFS Scheduling 67
5.3.2. Pseudo Code of the GSO Metaheuristic 71
6. CASE STUDY IN THE AEROSPACE COMPANY 75
6.1. Products: Panels of Fuselage 77
6.2. Production Processes 80
6.2.1. Data Analysis 94
6.2.2. DES Model Design for an HFS Configuration 95
6.3. Computational Study 112
7. COMPUTATIONAL STUDY 117
8. CONCLUSION AND FURTHER RESEARCH ISSUES 129
REFERENCES 133
APPENDICES
A. Some Plots for the Collection, Analysis and Interpretation of the Data 143
B. MoM_k Calculations for the Case Study 149
C. Processing Times of the Jobs 150
D. LEKIN Outputs and Excel Spreadsheets of Dispatching Rules 151
E. Outputs of the Renowned Heuristic Algorithms 155

LIST OF TABLES

TABLES

Table 2.1. Mathematical descriptions of objective function measures 10
Table 2.2. HFS scheduling problems: some examples for $\alpha|\beta|\gamma$ triplet representation 11
Table 2.3. Classification of the literature survey on HFS scheduling 20
Table 3.1. Performance characteristics depending on flow and job shops 23
Table 4.1. Example: Palmer's heuristic for ($F3||C_{max}$) problem 35
Table 4.2. Example: Completion times with Palmer's heuristic for ($F3||C_{max}$) problem 35
Table 4.3. CDS algorithm for ($Fm||C_{max}$) problem 36
Table 4.4. Example: 1^{st} iteration of the CDS algorithm for ($F3||C_{max}$) problem 36
Table 4.5. Example: 2^{nd} iteration of the CDS algorithm for ($F3||C_{max}$) problem 37
Table 4.6. Example: 1^{st} step of the NEH algorithm for ($F3||C_{max}$) problem 39
Table 4.7. Example: 2^{nd} step of the NEH algorithm for ($F3||C_{max}$) problem 39
Table 4.8. Example: 3^{rd} step of the NEH algorithm for ($F3||C_{max}$) problem 39
Table 4.9. Example: 4^{th} step of the NEH algorithm for ($F3||C_{max}$) problem 40
Table 4.10. Example: 5^{th} step of the NEH algorithm for ($F3||C_{max}$) problem 40
Table 4.11. Example: 6^{th} step of the NEH algorithm for ($F3||C_{max}$) problem 41
Table 4.12. Example ($HF3, P2, 1, P2||C_{max}$) problem for BH 43
Table 4.13. Example: Flow ratio table for each stage 43
Table 4.14. Example: Release times of jobs for the bottleneck stage 44
Table 4.15. Example: Due dates of jobs for the bottleneck stage 44
Table 4.16. Example: Table for scheduling the bottleneck stage 44
Table 4.17. Example: Scheduling the stage before the bottleneck stage 45
Table 4.18. Example: Final schedule with the BH 45
Table 5.1. Example: 1^{st} iteration of CP 50

Table 5.2. Example: 2^{nd} iteration of CP 50
Table 5.3. Example: 3^{rd} iteration of CP 50
Table 5.4. Example: 4^{th}, 5^{th} and 6^{th} iterations of CP 51
Table 5.5. Example: 7^{th} iteration of CP 52
Table 5.6. The example problem (*HF3, P2*$||C_{max}$) 55
Table 5.7. Example: (*HF3*, *P2*$||C_{max}$) problem for GLB computation 58
Table 5.8. Example: (*HF3*, *P2*$||C_{max}$) problem job-based lower bound computation 58
Table 5.9. Example: (*HF3*, *P2*$||C_{max}$) problem sum of the processing times of the jobs 58
Table 5.10. Example: (*HF3*, *P2*$||C_{max}$) problem calculation of LB(1) 59
Table 5.11. Example: (*HF3*, *P2*$||C_{max}$) problem calculation of LB(2) 59
Table 5.12. Example: (*HF3*, *P2*$||C_{max}$) problem calculation of LB(3) 60
Table 5.13. Example: The application of FAM strategy for the first stage 61
Table 5.14. Example: The application of ECT job sequencing rule at stage 2 62
Table 5.15. Example: The application of SPV rule 70
Table 6.1. Panels of the center fuselage of Airbus A320 78
Table 6.2. Operations routing for any panel manufacturing 78
Table 6.3. MoM_k calculation for the cooling stage 96
Table 6.4. MoM_k values in the HFS 98
Table 6.5. The results of the scheduling methods for the case study 113
Table 6.6. The completion times with the MILP model 113
Table 6.7. The start and end times of jobs with the CP model 114
Table 7.1. Results of test problems 119
Table 7.2. LB vs. GLB in the test problems 128
Table 0.1. MoM_k calculation table 149
Table 0.2. The processing times of the jobs at each stage (hours) 150
Table 0.3. The job sequence with the STPT rule 153
Table 0.4. The job sequence with the LTPT rule 154
Table 0.5. The job sequence with the Palmer's heuristic 155
Table 0.6. The job sequence with the CDS algorithm 156

Table 0.7. The job sequence with the NEH algorithm..... 157

LIST OF FIGURES

FIGURES

Figure 1.1. The representation of the HFS configuration 2
Figure 3.1. Complexity hierarchy based on the shop configuration 24
Figure 3.2. Complexity hierarchy based on the shop configuration for the makespan minimization 24
Figure 5.1. pulse vs. noOverlap in the CP model 53
Figure 5.2. The role of pulse cumulative function in the CP model for the HFS scheduling problem 54
Figure 5.3. Gantt Chart of the example problem solution generated by CP Optimizer 55
Figure 5.4. Gannt chart of sequence *J*1-*J*3-*J*4-*J*2 60
Figure 5.5. Gannt chart of sequence *J*3-*J*1-*J*4-*J*2 60
Figure 5.6. The flowchart of the HA 66
Figure 6.1. Production stages 0 and 1 in Building B200 82
Figure 6.2. Loaded pallet to be transported with the related work orders 83
Figure 6.3. Alkali clean vacuum manipulator with the fixed crane 84
Figure 6.4. Robotic cutting machine 86
Figure 6.5. Metallic manufacturing stages in Building B10 89
Figure 6.6. Surface operations in Building B20 91
Figure 6.7. Surface operations in Building B220 92
Figure 6.8. Painting process operations in Building B40 93
Figure 6.9. The source module of the DES model 99
Figure 6.10. Initial sequence table of panel (skin) type inserted to source module .. 99
Figure 6.11. The resource module structure of heat treatment stage 100
Figure 6.12. Heat treatment stage backlog 100
Figure 6.13. Heat treatment stage processing times (hrs) table of panels 101

Figure 6.14. Heat treatment stage with 2 parallel identical ovens 101
Figure 6.15. Heat treatment stage with the processing times table of panels 102
Figure 6.16. Heat treatment stage with failure distribution 102
Figure 6.17. Heat treatment stage with failure frequency and duration..... 103
Figure 6.18. Failure settings of a resource in Tecnomatix Plant Simulation 14 104
Figure 6.19. The DES model in Tecnomatix Plant Simulation 14 105
Figure 6.20. 3D version of the DES model in Tecnomatix Plant Simulation 14 106
Figure 6.21. Event controller as one of the simulation settings 107
Figure 6.22. Summary report generated at the end of a simulation run..... 107
Figure 6.23. The frame of shift calendar..... 108
Figure 6.24. The frame of bottleneck analyzer..... 109
Figure 6.25. The frame of bottleneck analyzer with its outcomes displayed on the complete DES model 109
Figure 6.26. The drain module 111
Figure 0.1. The histogram of panel 2's stretching stage processing time data from 2140 raw records 143
Figure 0.2. The best distribution with its mean=1.8920 hrs for panel 2's stretching stage processing time data..... 144
Figure 0.3. Breakdown duration plots of heat treatment stage before cleansing process (2010-2017 data)..... 144
Figure 0.4. Breakdown duration plots of heat treatment stage after cleansing process (2010-2017 data)..... 145
Figure 0.5. The histogram of heat treatment stage's breakdown duration from 73 cleansed records (2010-2017 data) 145
Figure 0.6. The best distribution with its mean=4.3306 hours for heat treatment stage's breakdown duration (2010-2017 data)..... 146
Figure 0.7. Breakdown frequency plots of heat treatment stage before cleansing process (2010-2017 data) 146
Figure 0.8. Breakdown frequency plots of heat treatment stage after cleansing process (2010-2017 data)..... 147

Figure 0.9. The histogram of heat treatment stage's breakdown frequency from 73 cleansed records (2010-2017 data) 147
Figure 0.10. The best distribution with its mean=36.6027 days for heat treatment stage's breakdown frequency (2010-2017 data)..... 148
Figure 0.11. SPT rule performance table 151
Figure 0.12. LPT rule performance table 152

LIST OF ABBREVIATIONS

ABBREVIATIONS

ACO: Ant Colony Optimization

ACS: Ant Colony System

AIS: Artificial Immune System

B&B: Branch and Bound

BH: Bottleneck Heuristic

CDS: Campbell, Dudek, and Smith

CP: Constraint Programming

CT: Cycle Time

DES: Discrete-Event Simulation

ECT: Earliest Completion Time

FAM: First Available Machine

FT: Finish Time

GA: Genetic Algorithm

GLB: Global Lower Bound

GSO: Galactic Swarm Optimization

HA: Hybrid Algorithm

HFS: Hybrid Flow Shop

HJS: Hybrid Job Shop

IH: Insertion Heuristic

LB: Lower Bound

LPT: Longest Processing Time

LS: Local Search

LTPT: Longest Total Processing Time

MILP: Mixed-Integer Linear Programming

MoM: Minimum number of Machines

NEH: Nawaz, Enscore, and Ham

NP: Non-deterministic Polynomial time

OPL: Optimization Programming Language

PP: Parallel Processor

PSO: Particle Swarm Optimization

QIA: Quantum-inspired Immune Algorithm

SIRO: Service in Random Order

SPT: Shortest Processing Time

SPV: Smallest Position Value

STPT: Shortest Total Processing Time

TS: Tabu Search

WIP: Work-In-Process

CHAPTER 1

INTRODUCTION

Due to the global and competitive structure of aerospace and defense industry, it is essential to realize both effective and efficient solutions for manufacturing processes in order to move one step forward to gain an advantageous position in this race. In this study, we are motivated by same planning and scheduling problems one of the companies that has a worldwide reputation in this market with a wide range of products offer. Among these products, one of the biggest shares belongs to the Airbus A320 fuselage's Section-18 and 19 panel parts. In order to manufacture these panels, a remarkably long operations sequence is followed as the manufacturing routing. Since the operations in this routing are carried out in several different manufacturing facilities at the factory campus, some serious problems emerge very often namely, abnormally long lead times, complex materials handling methods, and quality degradation. Moreover, these problems disrupt the timely delivery of the panels and thus the assembly schedule of them. Not only do they result in panel shortages, they also give rise to over-production, and Work-In-Process (WIP) accumulation. On the other hand, extra materials handling and WIP accumulation result in quality-related problems due to the frictions among parts, requiring rework operations which may sometimes result with scrapped parts.

Obviously, the type of layout observed in the manufacturing of the panel parts/subassemblies is a process type layout; and the manufacturing process is typically a Hybrid Job Shop (HJS). In this study, we propose an alternative manufacturing process that is basically a flow shop but with parallel machines at some or all stages of the shop. This type of flow shop is the Hybrid Flow Shop (HFS) which is an extension of the traditional flow shop configuration. The major difference is that

in the traditional flow shop, each stage has only a single machine whereas in the HFS configuration, at least one of the stages has more than one machine working in parallel. Figure 1.1 is a pictorial representation of an HFS with *k* stages and a number of variable machines (*m*) at each stage. But this added feature in the HFS brings about extra complexity and thus difficulty in terms of scheduling compared to the traditional flow shop. Hence, HFS scheduling problems have attracted an ever-increasing attention from researchers since the 1970s. In 1988, HFS scheduling problem is shown to be NP-complete, and, later, in 1996, NP-hard.

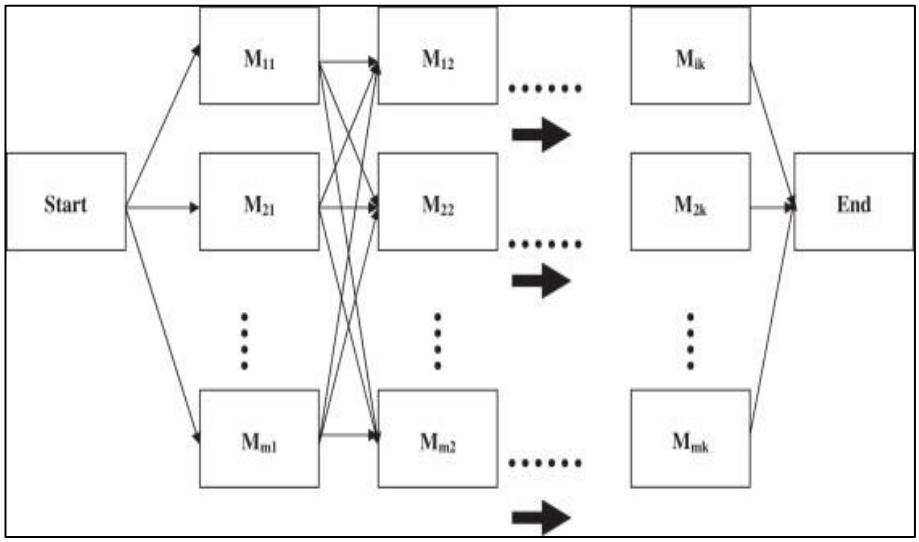


Figure 1.1. The representation of the HFS configuration

In addressing the HFS scheduling problem, MILP for small-to-medium size problems, dispatching rules, and several heuristic algorithms for the large realistic problems are applied in the studies in the literature. In this study, we propose a Constraint Programming (CP) model, a Hybrid Algorithm (HA) as a heuristic method, and a Galactic Swarm Optimization (GSO) as a metaheuristic approach. Both in HA and GSO, we use First Available Machine (FAM) for assignment of machines and Earliest Completion Time (ECT) for job sequences, especially chosen for an HFS scheduling problem. The proposed methods are applied to the test problems provided by Carlier

and Neron (2001) in order to have a comparative analysis of all methods including those in the literature for the HFS scheduling problem.

While HA heuristic and GSO metaheuristic solve the case problem and test problems to near optimality within an acceptable computational time, CP model solves all problems to optimality in a shorter computational time than the others, and hence outperforms the others. Based on the results of the CP model, the proposed HA heuristic and GSO metaheuristic seem to be promising in terms of both solution quality and time for the case problem and test problems. Thus, these solution methods seem to be applicable in other industries as well where the manufacturing process is in the HFS configuration.

In the case study with the aerospace company under consideration, the current HJS process is converted to an HFS process. To start with, the Minimum number of Machines (MoM) required for each stage in the HFS configuration is determined by means of simulation. Hence, a Discrete-Event Simulation (DES) model is developed for the manufacturing of the panel parts in the new HFS. Then scheduling problem is addressed for the HFS using the approaches proposed in the study.

Chapter 2 discusses the relevant studies in the literature about the HFS scheduling problem. In Chapter 3 includes the problem description, while Chapter 4 presents the MILP Model and the renowned heuristics for the HFS scheduling problem with its assumptions. Chapter 5 describes the proposed solution methods including the CP model, HA heuristic, and GSO metaheuristic with their structures and pseudo codes. The case study in the aerospace company under consideration and computational study are presented in Chapter 6 including the current production system at the company. This chapter also covers the conversion of the HJS to HFS configuration at the company by means of the discrete-event simulation modelling. Chapter 7 covers the application of proposed solution methods to the test problems as a computational study. Moreover, they are compared to the other solution methods developed earlier

in the literature in terms of both overall and average performances. Chapter 8 includes the conclusion and further research issues emphasizing the important findings derived from the results.

CHAPTER 2

LITERATURE REVIEW

2.1. Framework and Notation

In all scheduling problems, it is assumed that there is a finite number of jobs and machines in the particular manufacturing environment as a system. Therefore, in order to comprehend the variety of the systems in the literature, it is essential to understand the general framework and notation used for describing the scheduling problems similar to the descriptions of Pinedo (2016), and Ruiz and Vazquez-Rodriguez (2010). In general, the number of machines is denoted by m while the number of jobs is denoted by n. Usually, the index i refers to a machine while the index j refers to a job. If job j is processed on machine k then the pair (j, k) is used in order to represent the relationship between job j and machine k. Concordantly, the descriptions in the following are the system's components in which only the processing time is indispensable for each type of system:

Processing time (p_{jk}): It represents the processing time of job j on machine k.

Scheduling problems are classified by Graham et al. (1979) via a triplet $\alpha|\beta|\gamma$, where α field represents the machine environment referred to as the shop configuration, β field describes a set of assumptions, constraints, job characteristics and restrictions related to the problem, and γ field indicates the performance measures of concern. The possible shop configurations represented in the α field are:

Single machine (*1*): The case of a single machine in shop configuration.

Identical parallel machines (*Pm*): *m* identical parallel machines in a single stage in which job *j* is processed on any one of them.

Uniform parallel machines (*Qm*): *m* parallel machines with different speeds in a single stage where the speed of the machine *k* is denoted by v_k.

Unrelated parallel machines (*Rm*): *m* different machines with different speeds in a single stage where the speed of the process for job *j* executed on machine *k* is v_{jk}.

Flow shop (*Fm*): *m* serial machines on which each job has to be processed in the same sequence. After a job is processed on machine *k*, it joins machine $k + 1$'s queue which is also called as buffer or backlog. If the queue of the machine is assumed to behave according to First-In First-Out (FIFO) discipline, then a flow shop is referred to as a permutation flow shop and it is denoted by *prmu* in β field.

Flexible flow shop (*HFc*): A hybrid version of the traditional flow shop and parallel machine environments. Instead of *m* serial machines, flexible flow shop consists of *c* serial stages in which at least one of them has more than one identical, uniform or unrelated machines in parallel. In the literature, flexible flow shops have been widely known as Hybrid Flow Shop (HFS) or Multi-Processor Flow Shop (MPFS). Each job has the same routing.

The possible entries in the β field that represents the features of the system are as follows:

Due date (d_j): Job *j*'s planned completion date which is agreed upon by both the customer and the supplier.

Release date (r_j): Job *j*'s ready time which is also job *j*'s arrival time at the system.

Preemption (*prmp*): Interrupting a job being processed on a machine without waiting its finish time in order to load a different job.

Precedence constraints (*prec*): Constraints taken into account when there are predecessor or successor relationships among the jobs.

Sequence dependent setup times (s_{sd}): If setup times depend on the sequence of jobs then s_{sd} is taken into account in completion time calculations. If setup times are not sequence dependent, then they are represented as s_{nsd} and included in the processing times.

Job families (*fmls*): n jobs belong to F different job families. There is a setup time denoted by s_{gh} between the job families g and h while there are no setup times among the jobs in the same family.

Batch processing (*batch*): A machine processes a number of the same or different jobs in series.

Unavailability (*unavail*): Usually machines are not always available because of breakdowns (*brkdwn*), shift changes or scheduled maintenances.

Machine eligibility restrictions (M_j): In parallel machines (Pm) environment, if all m machines are not capable of processing job j, then the capable ones that process job j are denoted by M_j.

Blocking (*block*): In a flow shop environment, if the buffer between two successive stages, then blocking may occur, not allowing the upstream machine to release a completed job, therefore, preventing it from processing the next job.

No-wait (*nwt*): In a flow shop environment, a job is not allowed to wait between two consecutive machines/stages. Therefore, the starting time of a job at the first machine/stage is delayed in order to make this job go on without waiting for any next machines/stages. Moreover, it is understood that under *nwt* constraint, the shop operates according to First-In First-Out (FIFO) discipline.

Recirculation (*rcrc*): A job visits a machine, a stage or a work center more than once.

Size ($size_{jk}$): In an HFS environment, at each production stage k, one operation of job j (op_{jk}) is simultaneously processed on $size_{jk}$ parallel machines without preemption during the required processing time of job j.

The γ field represents the objective function measure which usually tries to minimize a function of the completion times of jobs. Therefore, the common γ entries related to the objective function are the followings:

Maximum completion time (C_{max}): Makespan that is the completion time of the last job. Minimizing the makespan is equivalent to maximizing the utilization of machine/s.

Maximum flow time (F_{max}): Flow time of a job is completion time minus release time. The biggest value among the flow times of all jobs represents the maximum flow time.

Maximum lateness (L_{max}): Lateness value of a job can be obtained by subtracting due date from completion time. The biggest lateness value among all jobs represents the maximum lateness. If lateness has a positive value for a job, then it means that the job is completed late and it is tardy. On the other hand, if it has a negative value for a job, then the job is completed early. If lateness value is equal to 0, then the job is completed on time.

Maximum tardiness (T_{max}): Jobs' non-negative lateness values. The biggest tardiness value among all jobs is the maximum tardiness.

Maximum earliness (E_{max}): Earliness value of a job is due date minus completion time. It is either positive or equal to 0 in order to ensure that the job is completed either early or on time. The biggest earliness value among all represents the maximum earliness.

Total/average completion time ($\overline{C}$): Either summation or average of all completion times of jobs.

Total/average weighted completion time ($\overline{C}^w$): Either summation or average of all weighted completion times of jobs.

Total/average flow time ($\overline{F}$): Either summation or average of all flow times of jobs.

Total/average weighted flow time ($\overline{F}^w$): Either summation or average of all weighted flow times of jobs.

Total/average tardiness ($\overline{T}$): Either summation or average of all tardiness values of jobs.

Total/average weighted tardiness ($\overline{T}^w$): Either summation or average of all weighted tardiness values of jobs.

Total/average earliness ($\overline{E}$): Either the summation or average of earliness values of jobs.

Total/average weighted earliness ($\overline{E}^w$): Either summation or average of all weighted earliness values of jobs.

Number of tardy jobs (U): Number of jobs completed later than their due dates.

Total weighted number of tardy jobs (U^w): Summation of all weighted tardy jobs. Usually, these weights are represented by total holding or inventory costs.

The objective function measures based on the review of Ruiz and Vazquez-Rodriguez (2010) are commonly used in scheduling problems (Table 2.1).

Table 2.1. Mathematical descriptions of objective function measures

Notation	Description	Meaning
C_{max}	$\max_j C_j$	maximum completion time
F_{max}	$\max_j (C_j - r_j)$	maximum flow time
L_{max}	$\max_j L_j$	maximum lateness
T_{max}	$\max_j T_j$	maximum tardiness
E_{max}	$\max_j E_j$	maximum earliness
$\overline{C}$	$\sum C_j/n$	total/average completion time
$\overline{C}^w$	$\sum w_j C_j/n$	total/average weighted completion time
$\overline{F}$	$\sum F_j/n$	total/average flow time
$\overline{F}^w$	$\sum w_j F_j/n$	total/average weighted flow time
$\overline{T}$	$\sum T_j/n$	total/average tardiness
$\overline{T}^w$	$\sum w_j T_j/n$	total/average weighted tardiness
U	$\sum U_j$	number of tardy jobs
U^w	$\sum w_j U_j$	total weighted number of tardy jobs
$\overline{E}$	$\sum E_j/n$	total/average earliness
$\overline{E}^w$	$\sum w_j E_j/n$	total/average weighted earliness

Different HFS scheduling problems can be represented by the $\alpha|\beta|\gamma$ triplet as in Table 2.2.

Table 2.2. HFS scheduling problems: some examples for $\alpha|\beta|\gamma$ triplet representation

Triplet notation	**α description**	**β description**	**γ description**
(*HF2, 1, P2*$\|\|C_{max}$)	2-stage HFS with a single machine in the first stage and two parallel identical machines in the second stage	-	Minimize C_{max}
(*HFc, Pm*$\|$*nwt*$\|C_{max}$)	*c*-stage HFS with *m* parallel identical machines in each stage	No-wait constraint	Minimize C_{max}
(*HF2, Pm*$\|$*no-idle*$\|\bar{T}$,*U*)	2-stage HFS with *m* parallel identical machines in each stage	No idle time on machines	Minimize $\sum T_j$ and $\sum U_j$
(*HFc, Rm*$\|$*batch, fmls,* s_{gh}, *unavail,* r_j, M_j, *skip*$\|\alpha\bar{C}^w+\beta\bar{T}^w$)	*c*-stage HFS with *m* parallel unrelated machines in each stage	Batch, job families, sequence dependent setup times, machine unavailabilities, release times and machine eligibility restrictions. In addition, some jobs may skip some stages	Minimize $\sum w_j C_j$ and $\sum w_j T_j$
(*HFc, Qm*$\|$*rcrc, time window, fmls,* $M_j$$|C_{max}$)	*c*-stage HFS with *m* parallel uniform machines in each stage	Recirculation, time window, job families and machine eligibility restrictions	Minimize C_{max}
(*HFc, Rm*$\|$*buffer, block*$\|C_{max}$)	*c*-stage HFS with *m* parallel unrelated machines in each stage	Blocking may occur due to the limited buffer	Minimize C_{max}

2.2. Review of the HFS Scheduling Studies

In this section we review the studies about the HFS scheduling problems.

As the first algorithm, Branch and Bound (B&B) algorithm is proposed for an HFS problem (*HF2, 1, P2*||C_{max}) by Rao (1970).

B&B is applied again to an HFS problem (*HF2, P2, 1*||C_{max}) by Arthanary and Ramaswamy (1971). Later, Gupta (1988) proves that this problem is strongly NP-hard and he proposes a heuristic algorithm.

B&B is applied again by Salvador (1973) to a more generalized HFS problem (*HFc, Pm*|*nwt*|C_{max}) for small instances. Later, B&B algorithm is applied again to a similar HFS problem (*HFc, Pm*||C_{max}) by Brah and Hunsucker (1991).

For the first time, dispatching rules are studied by Paul (1979) for an HFS problem (*HF2, Pm*|*no-idle*|$\bar{T}$, *U*).

Dispatching rules are applied by Narasimhan and Panwalkar (1984) for an HFS problem (*HF2, 1, R2*||*idleness, waiting*).

Wittrock (1985) studies an HFS problem (*HF3, Pm*|*skip*|C_{max}), with different methods, including LP, Longest Processing Time (LPT) as a dispatching rule, and a dynamic balancing algorithm as a heuristic approach.

Kochhar and Morris (1987) study an HFS problem (*HFc, Pm*|s_{nsd}, *block, skip, brkdwn*|$\bar{F}$), by applying dispatching rules and heuristic algorithms such as myopic method and Local Search (LS) approach.

Due to the exhaustive computational time of B&B algorithms, Sriskandarajah and Sethi (1989) propose dispatching rule-based heuristic algorithm for an HFS problem (*HF2, Pm*$||C_{max}$) in order to observe its worst and average cost performances in a short period of time.

A real-world problem, a paper industry, is studied by Sherali et al. (1990) as an HFS problem (*HF2, P10, P12*||*allocation, sequence*) via a mathematical model.

Two HFS problems (*HFc, Pm*|*prmu*|$\overline{F}$) and (*HF2, Pm*||$\overline{F}$) are studied by Rajendran and Chaudhuri (1992). They apply B&B algorithm to the first problem and heuristic algorithm to the second problem. For the purpose of comparison, they also apply Shortest Processing Time (SPT) as a dispatching rule to the second problem in order to determine which method is superior. In terms of both solution's quality and computational time performance, their heuristic algorithm has better results than SPT rule.

A real-world problem, packaging industry's scheduling problem is considered as a (*HFc, Rm*|s_{sd}|$\overline{T}^w$) problem by Adler et al. (1993) and a five-step heuristic algorithm is developed in order to minimize the total priority-based tardiness (also known as the total weighted tardiness).

Lee and Vairaktarakis (1994) apply the first error bound analysis to an HFS problem (*HF2, Pm*$||C_{max}$).

Chen (1995) uses a worst-case performance ratio for two HFS problems, (*HF2, 1, Pm*$||C_{max}$) and (*HF2, Pm, 1*$||C_{max}$), by classifying some of the heuristics proposed earlier for makespan minimization in the literature.

Hoogeveen et al. (1996) show that an HFS problem (*HF2, Pm*|*prmp*|C_{max}) is NP-hard in the strong sense.

Gupta et al. (1997) work on an HFS problem (*HF2, Pm, 1||*C_{max}) using B&B and heuristic algorithms (both constructive and improvement) from the literature with a new LB calculated for experimental test problems.

Nowicki and Smutnicki (1998) apply Tabu Search (TS) to an HFS problem (*HFc, Pm||*C_{max}) by considering tabu restrictions and search diversification while creating the tabu list and developing neighborhood search strategy.

Brah and Loo (1999) study an HFS problem (*HFc, Pm||*C_{max}, $\overline{F}$) by applying regression analysis to determine the performance of Campbell, Dudek, and Smith (1970) heuristic (CDS) algorithm, Nawaz, Enscore and Ham (1983) heuristic (NEH) algorithm, Hundal and Rajgopal (1988) modified Palmer heuristic algorithm, Yang, Pegden, and Enscore (1984) combined heuristic algorithm and Ho (1995) heuristic algorithm. Moreover, with regression analysis, they find out that job characteristics, number of jobs, number of stages and parallel identical machines, have significant effects on the quality of the results obtained.

A real-world problem, concrete blocks production as a building industry, is studied by Grabowski and Pempera (2000) as an HFS problem (*HFc, Pm|nwt|*C_{max}). They apply TS metaheuristic algorithm to the problem. Their algorithm seems to be promising in terms of balancing the trade-off between the solution time and the quality of the solutions obtained.

An HFS problem (*HFc, Pm||*C_{max}) is studied by Neron et al. (2001). They observe that their branching schemes are effective for small to medium size instances, but not for larger instances.

An HFS problem (*HFc, Pm||*$\overline{E}^w+\overline{T}^w+\overline{C}^w+\overline{d}_j^w$) is studied by Gupta et al. (2002) with different problem characteristics, controllable processing times, varying between a minimum and a maximum value depending on the use of a continuously divisible

resource, and assignable weighted due dates, which are not a priori given but can be fixed by a decision-maker in return for a due date assignment cost. This cost is one of the objective measures constituting a cost function in terms of the penalty cost. For this problem, they develop constructive heuristic algorithms as dispatching rules based on insertion techniques and improvement heuristic algorithms as LS methods based on shifting neighborhood procedures.

Kurz and Askin (2003) study an HFS problem (*HFc, Pm*$|s_{sd}|C_{max}$) by exploring cyclic, multiple insertion, and Johnson's Rule-based heuristics and comparing the performance of these heuristics through evaluating them on a set of test problems whose data are generated as an experimental design.

Similar to the genetic algorithm (GA) as an evolutionary algorithm, Artificial Immune System (AIS), a smart problem-solving technique, is proposed by Engin and Doyen (2004) for an HFS problem (*HFc, Pm*$||C_{max}$). Their experimental results show that AIS is an effective and efficient method that can be used for real-life industrial problems. Moreover, their AIS heuristic is hybridized with some other heuristic algorithms.

Oguz and Ercan (2005) propose a GA with its four different versions for an HFS problem (*HFc, Pm*$|size_{jk}|C_{max}$). They check the deviation of these GAs from the LB value inspired by the previous studies in the literature in order to find the GA with the best genetic operators among all. They also add that the best GA outperforms the TS algorithm of Oguz et al. (2004).

A real-world problem, similar to fabric manufacturing as a textile industry, the production process of ceramic tiles is considered as an HFS scheduling problem (*HFc, Rm*$|s_{sd}$, $M_j|C_{max}$) by Ruiz and Maroto (2006) through developing a GA algorithm with four new crossover operators. After designing extensive experimental datasets, they calibrate their algorithm and compare it to nine other metaheuristic algorithms introduced earlier in the literature.

For an HFS problem (*HF3, Rm|prec, s_{sd}, block|C_{max}*), Chen et al. (2007) propose a TS metaheuristic algorithm, a mathematical model and a LB.

Ruiz et al. (2008) develop a mathematical model for a realistic HFS problem (*HFc, Rm|skip, s_{sd}, time lag, r_m, M_j, prec|C_{max}*) and also test the model for medium size instances. However, for real medium and especially large instances, they develop six heuristic algorithms, five of which are based on dispatching rules and one of which is inspired from the earlier studies. For benchmarking purposes, they develop a Classification Tree as a Machine Learning (ML) technique by using advanced statistical tools.

Naderi et al. (2009) propose a metaheuristic algorithm, namely Hybrid Simulated Annealing (HSA), for a realistic HFS problem (*HFc, Pm|s_{sd}, transportation time|*$\overline{F}$, $\overline{T}$), by comparing the performance of HSA to well-known dispatching rules and the adaptations of some metaheuristic algorithms introduced earlier in the literature of HFS without transportation times. According to their experimental assessment, HSA outperforms the solution methods in the literature that they include in their comparisons.

Dugardin et al. (2010) develop three metaheuristics as Strength Pareto Evolutionary Algorithm (SPEA), Non-dominating Sorting GA (NSGA) and Lorenz NSGA (L-NSGA) and an exact algorithm for an HFS problem (*HFc, Pm|rcrc|U, capacity utilization*). They examine the performances of these three metaheuristic algorithms by using a DES model. Their exact method is a full enumeration technique that is used for small instances only to obtain optimal solutions, as expected, in order to determine the solution qualities of these three metaheuristic algorithms. The computational results of the DES model show that SPEA and NSGA are outperformed by L-NSGA whose solutions are very close to the optimal solutions yielded by the full enumeration technique.

An Efficient GA (EGA) with Neighborhood Based Mutation (NBM) is proposed by Engin et al. (2011) for an HFS problem (*HFc, Pm*$||C_{max}$) through comparing it to the GA without NBM and a parallel greedy heuristic algorithm. It is observed that EGA performs better than the GA and parallel greedy heuristic algorithm in terms of solution quality for the test problems.

Liao et al. (2012) develop a metaheuristic algorithm, Particle Swarm Optimization (PSO) with a Bottleneck Heuristic (BH) to completely manipulate the bottleneck stage and also with a SA heuristic to avoid stucking in local optima (PSO-SA-BH), for an HFS problem (*HFc, Pm*$||C_{max}$) through comparing it to PSO and PSO-SA in order to find the best PSO variant. As a result, PSO-SA-BH is chosen as the best one for the purpose of further comparison. Then, PSO-SA-BH is compared to Quantum-inspired Immune Algorithm (QIA), Ant Colony Optimization (ACO), AIS and B&B algorithms. According to the experimental results, PSO-SA-BH performs better than QIA, ACO and AIS in terms of both effectiveness and efficiency, and better than B&B algorithm in terms of efficiency, and the same in terms of effectiveness.

Luo et al. (2013) develop a metaheuristic algorithm, Multi-Objective ACO (MOACO), for an HFS problem (*HFc, Qm*$||C_{max}$*, electric power cost*) with the presence of Time-Of-Use (TOU) electricity prices as an energy consumption approach in the context of green manufacturing. In the light of computational experiments' results, even though MOACO is worse than SPEA and NSGA in terms of efficiency, it outperforms them in terms of effectiveness.

Marichelvam et al. (2014) develop metaheuristic algorithms, Cuckoo Search (CS) and Improved CS (ICS) for an HFS problem (*HFc, Pm*$||C_{max}$) by comparing CS and ICS to GA, SA, ACO, PSO and an existing constructive heuristic algorithm which is also used to generate initial solutions for ICS in order to obtain optimal or near optimal solutions quickly. Computational results show that not only does ICS provide optimal

results with minimum CPU time, but also it is superior to the other metaheuristic algorithms in terms of both effectiveness and efficiency.

Li and Pan (2015) propose a novel hybrid algorithm, combining TS and Artificial Bee Colony (ABC), (TABC) to solve an HFS problem (*HFc, Rm|buffer, block|*C_{max}). According to the experimental results, TABC performs better than the five existing heuristic algorithms in the literature for most of the instances in terms of both effectiveness and efficiency.

A realistic HFS problem (*HFc, Rm|batch, fmls,* s_{gh}*, unavail,* r_j*,* M_j*, skip|*$\alpha\overline{C}^w+\beta\overline{T}^w$) is studied by Shahvari and Logendran (2016) through a mathematical model solved via CPLEX for small instances to obtain optimal/upper and LBs, and developing several TS based metaheuristic algorithms due to NP-hardness of the medium and especially large instances in the strong sense. According to the comparative numerical results, TS with Path-Relinking (TS/PR) based batch scheduling is promising and performs well most of the time for small to large instances in terms of effectiveness and especially efficiency.

According to the study of Chamnanlor et al. (2017), a metaheuristic algorithm, GA hybridized with ACO (GACO), for an HFS problem (*HFc, Qm|rcrc, time window, fmls,* M_j*|*C_{max}) and a mathematical model are presented. GACO is compared to GA and ACO in terms of computational results showing that GACO has the best results compared to the other two.

Li et al. (2018) develop Energy-Aware Multi-objective Optimization Algorithm (EA-MOA) for an HFS problem (*HFc, Pm|*s_{sd}*|*C_{max}*, total energy consumption*). Comparing EA-MOA to several efficient heuristic algorithms in the literature, the experimental results show that EA-MOA's robustness and efficiency are promising.

2.3. Summary of the Survey

It is obvious that shop configurations vary in terms of the number of stages, the number of machines at the stages and the similarity/dissimilarity of the machines at any stage. On the other hand, shop characteristics and the objective function measures cannot be classified so easily as the shop configurations. Therefore, each HFS scheduling problem is considered according to its shop characteristics and the objective function measures. Solution methods can be classified in four main groups, namely, exact, heuristic, hybrid, and other methods, like in the classification of Ribas et al. (2010)'s review paper. Exact methods solve HFS scheduling problems to optimality, while the others do not guarantee optimality all the time owing to the parameters of the problems such as the size of the problem instance. B&B algorithm and mathematical modelling are good examples for the exact method. On the other hand, commonly used dispatching rules are good examples for constructive heuristic algorithms, while the frequently used metaheuristic GA is a good example for an improvement heuristic algorithm. Furthermore, any combined versions of these methods such as Column Generation (CG) with GA or with SA, and Dynamic Programming (DP) with Lagrangian Relaxation (LR) are good examples for the hybrid algorithms. Finally, good examples for the other methods are DES models or expert systems. In order to simplify the comprehension, any combinations among exact, heuristic, hybrid and the other solution methods are considered as hybrid algorithms. To summarize the literature survey in the previous section, Table 2.3 is developed based on this classification.

Table 2.3. Classification of the literature survey on HFS scheduling

Authors & Year	Objective Function Measures	Number of Stages (c>3)	Machines Type at a Stage	Solution Methods			
				Exact	Heuristic	Hybrid	Other
Rao (1970)	C_{max}	2	P	X			
Arthanary and Ramaswamy (1971)	C_{max}	2	P	X			
Salvador (1973)	C_{max}	c	P	X			
Paul (1979)	$\overline{T}$, U	2	P		X		
Narasimhan and Panwalkar (1984)	*idleness, waiting*	2	R		X		
Wittrock (1985)	C_{max}	3	P	X	X		
Kochhar and Morris (1987)	$\overline{F}$	c	P		X		
Gupta (1988)	C_{max}	2	P		X		
Sriskandarajah and Sethi (1989)	C_{max}	2	P		X		
Sherali et al. (1990)	*allocation, sequence*	2	P	X			
Brah and Hunsucker (1991)	C_{max}	c	P	X			
Rajendran and Chaudhuri (1992)	$\overline{F}$	2	P		X		
		c		X			

Continued on next page

Table 2.3. – *Continued from previous page*

Authors & Year	Objective Function Measures	Number of Stages (c>3)	Machines Type at a Stage	Solution Methods			
				Exact	Heuristic	Hybrid	Other
Adler et al. (1993)	$\bar{T}^w$	c	R		X		
Lee and Vairaktarakis (1994)	C_{max}	2	P		X		
Chen (1995)	C_{max}	2	P		X		
Hoogeveen et al. (1996)	C_{max}	2	P		X		
Gupta et al. (1997)	C_{max}	2	P	X	X		
Nowicki and Smutnicki (1998)	C_{max}	c	P			X	
Brah and Loo (1999)	C_{max}, $\bar{F}$	c	P		X	X	
Grabowski and Pempera (2000)	C_{max}	c	P		X		
Neron et al. (2001)	C_{max}	c	P	X			
Gupta et al. (2002)	$\bar{E}^w$, $\bar{T}^w$, $\bar{C}^w$ and $\bar{d}_j^w$	c	P		X		
Kurz and Askin (2003)	C_{max}	c	P		X		
Engin and Doyen (2004)	C_{max}	c	P		X		
Oguz and Ercan (2005)	C_{max}	c	P		X		
Ruiz and Maroto (2006)	C_{max}	c	R		X		

Continued on next page

Table 2.3. – *Continued from previous page*

Authors & Year	Objective Function Measures	Number of Stages (c>3)	Machines Type at a Stage	Solution Methods			
				Exact	Heuristic	Hybrid	Other
Chen et al. (2007)	C_{max}	3	R	X	X		
Ruiz et al. (2008)	C_{max}	c	R	X	X		X
Naderi et al. (2009)	$\overline{F}$, $\overline{T}$	c	P		X		
Dugardin et al. (2010)	*U, capacity utilization*	c	P	X	X	X	X
Engin et al. (2011)	C_{max}	c	P		X		
Liao et al. (2012)	C_{max}	c	P			X	
Luo et al. (2013)	C_{max}, *electric power cost*	c	Q		X		
Marichelvam et al. (2014)	C_{max}	c	P		X		
Li and Pan (2015)	C_{max}	c	R			X	
Shahvari and Logendran (2016)	$\overline{C}^w$, $\overline{T}^w$	c	R	X		X	
Chamnanlor et al. (2017)	C_{max}	c	Q	X		X	
Li et al. (2018)	C_{max}, *total energy consumption*	c	P		X		

CHAPTER 3

PROBLEM DESCRIPTION

In this study, we are motivated by the production planning and scheduling issues in the job shop of center fuselage panels' manufacturing at the aerospace company under consideration. However, since an HFS configuration has more characteristics (Table 3.1) (Askin et al., 1993) better than an HJS environment, in this study, HFS scheduling problem is considered.

Table 3.1. Performance characteristics depending on flow and job shops

Characteristic	Flow Shop	Job Shop
Lead time	Low	High
WIP	Low	High
Skill level	Choice	High
Product flexibility	Low	High
Demand flexibility	Medium	High
Machine utilization	High	Medium-low
Worker utilization	High	High
Unit production cost	Low	High

COMPLEXITY

As it is seen on Table 3.1, two of the major contributing factors why HFS scheduling problem is considered in this study are lower lead time and WIP accumulation. Other than these, lower unit production cost is another beneficial feature of an HFS configuration as a layout type of the manufacturing process. Moreover, higher machine and worker utilizations can be positively considered in terms of efficiency in the manufacturing process. Because the machines in an HFS configuration are orderly

located in the manufacturing site, capacity increase can easily be adjusted. In this way, machine/worker utilizations, skill level, and demand flexibility can also be adjusted with ease. Due to the fact that a certain type of products is considered for addressing an HFS scheduling problem in this study, lower product flexibility and moderate demand flexibility cause no negative effect on the manufacturing process.

Due to the fact that HFS scheduling is a complex problem which is proven to be NP-hard in the strong sense, it is also difficult to solve this problem. In order to understand the complexity level of HFS scheduling problem and its place among the other shop configurations, Figures 3.1 and 3.2 are presented (Pinedo, 2016).

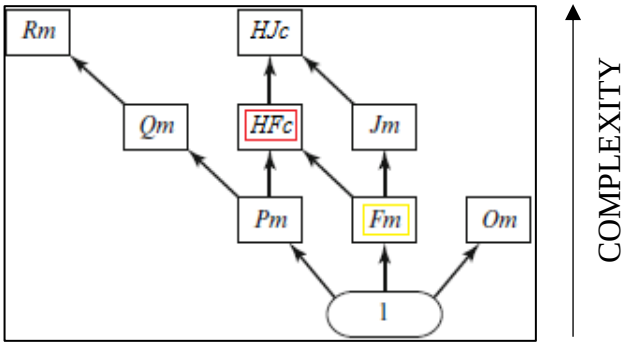


Figure 3.1. Complexity hierarchy based on the shop configuration

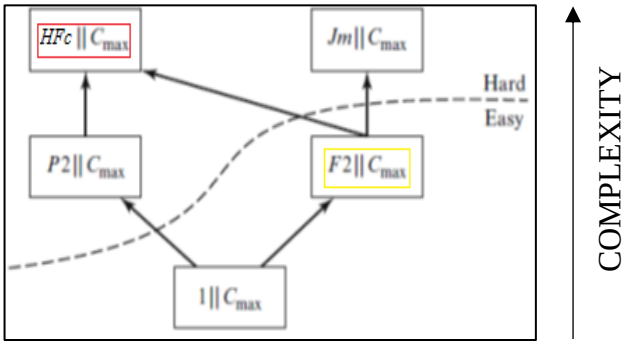


Figure 3.2. Complexity hierarchy based on the shop configuration for the makespan minimization

As it is seen on Figure 3.1, the only shop configurations, more complex than the HFS, are the unrelated parallel machines and the HJS as expected. Furthermore, on Figure 3.2, for the makespan minimization, scheduling the HFS and the job shop configurations is harder. Especially, the HFS scheduling problem is harder than 2-stage flow shop scheduling problem for the makespan minimization.

As an extended version of a traditional flow shop, an HFS has a more complex structure, due to the number of machines at a stage. Therefore, according to the study by Brah (1988), there are too many paths, shown on Equation (3.1), in order to obtain a schedule for an HFS scheduling problem ($HFc||C_{max}$).

$$\prod_{k=1}^{c} \binom{n\text{-}1}{m_k\text{-}1} \frac{n!}{m_k!} \tag{3.1}$$

On the other hand, a permutation flow shop has ($n!$) and a non-permutation flow shop has $(n!)^m$ paths for a schedule. Hence, compared to permuation and non-permutation flow shops, HFS scheduling problem is obviously harder to be solved to optimality.

Because of the motivations stated above, in this study, several renowned solution approaches in the literature are applied to the HFS scheduling problem. After the application of these approaches, it is observed that there is a room for improvement to obtain better solutions for the HFS scheduling problem. For this purpose, three solution approaches, a Constraint Programming (CP) model as an exact method, a Hybrid Algorithm (HA) as a heuristic method, and a Galactic Swarm Optimization (GSO) as a metaheuristic method, are proposed.

In the following chapters, we focus on the renowned methods for the HFS scheduling problem.

CHAPTER 4

HFS SCHEDULING: MILP MODEL AND HEURISTICS

Due to the NP-hardness nature of the HFS scheduling problem, exact solution methods remain insufficient to obtain both effective and efficient solutions for medium and especially large problem instances. Hence, several heuristic methods are proposed in terms of effectiveness and efficiency. As reviewed in Chapter 2, these heuristics are either used as a single heuristic or as a mix of them in a hybrid way. In this chapter, first, we formulate the HFS scheduling problem as a Mixed-Integer Linear Programming (MILP) model and then discuss the dispatching rules as heuristics. Finally, we discuss the renowned heuristics for the HFS scheduling problem.

4.1. Mixed-Integer Linear Programming (MILP) Model

This section presents the MILP model for an HFS scheduling problem based on the model of Ruiz and Vazquez-Rodriguez (2010). Before describing this model, following assumptions are made:

- All jobs and all machines are available at time zero. Therefore, there are no release dates constraints for the jobs.

- Parallel machines are identical at each stage. Therefore, there is no machine eligibility constraint for the jobs.

- A machine at a stage processes only one operation at a time and a job is processed by only one machine at a stage at a time. Therefore, partial processings are not allowed.

- Setup times are not sequence dependent. Therefore, they are included in the processing times of the jobs at each stage.

- Preemption is not allowed. Therefore, before loading the next job, current job on the machine is to be finished.

- The buffer between two consecutive stages has unlimited capacity. Therefore, blocking does not occur.

- Problem data are deterministic and known a priori. Therefore, the routings of the jobs are known, the number of parallel identical machines at a given stage is fixed and the processing times of the jobs are deterministic. There are no specific due date constraints related to the jobs, that is, all the jobs are assumed to have a common due date. Since the importance of the jobs is the same, the weights of the jobs are set to "1".

- A job may skip a given stage. This is represented by equaling this job's processing time to "0" at this stage.

- There are no predecessor and successor relationships among the jobs. Therefore, the sequence of the jobs can be altered as required.

- All machines are always available. Therefore, there are no breakdown, shift change, and scheduled maintenance constraints related to the machines.

- Batch processing of a certain job is not possible due to the high cost of tool requirements.

- There are no no-wait constraints for the jobs. However, there are also no technical constraints to make the jobs stop.

- A job does not visit a given stage more than once. Therefore, there are no recirculation constraints related to the jobs.

- Job sequencing is allowed to change from one stage to another, that is, non-permutation schedules are allowed.

According to the practical situations, these assumptions may slightly change for the different variants of the HFS scheduling problems. The MILP model based on these assumptions is presented below:

The MILP model

Sets

J: number of jobs: $j=\{1, \ldots, n\}$
K: number of stages: $k=\{1, \ldots, c\}$
L: number of parallel identical machines at stage k: $l=\{1, \ldots, m_k\}$

Parameters

p_{jk}: processing time of job j at stage k

$M=\sum_{j=1}^{n}\sum_{k=1}^{c} m_k p_{jk}$ a big number

Decision Variables

c_{jk}: completion time of job j at stage k

$y_{jkl} = \begin{cases} 1, \text{ if job } j \text{ is scheduled at the } l^{th} \text{ machine of stage } k \\ 0, \text{ otherwise} \end{cases}$

$x_{jrk} = \begin{cases} 1, \text{ if job } j \text{ precedes job } r \text{ at stage } k \\ 0, \text{ otherwise} \end{cases}$

C_{max}: maximum completion time of the jobs at the last stage

Objective Function

minimize C_{max} (4.1)

Constraints

subject to

$$\sum_{l=1}^{m_k} y_{jkl} = 1 \qquad \forall j \in J, \forall k \in K \qquad (4.2)$$

$$c_{j,k-1} + p_{jk} \leq c_{jk} \qquad \forall j \in J, \forall k \in K: k > 1 \qquad (4.3)$$

$$p_{j1} \leq c_{j1} \qquad \forall j \in J \qquad (4.4)$$

$$c_{jk} + p_{rk} - M(3 - x_{jrk} - y_{jkl} - y_{rkl}) \leq c_{rk} \qquad \forall j \in J, \forall r \in J: j < r, \forall l \in L \qquad (4.5)$$

$$c_{rk} + p_{jk} - M(2 + x_{jrk} - y_{jkl} - y_{rkl}) \leq c_{jk} \qquad \forall j \in J, \forall r \in J: j < r, \forall l \in L \qquad (4.6)$$

$$c_{jc} \leq C_{max} \qquad \forall j \in J \qquad (4.7)$$

$$y_{jkl} \in \{0, 1\} \qquad \forall j \in J, \forall k \in K, \forall l \in L \qquad (4.8)$$

$$x_{jrk} \in \{0, 1\} \qquad \forall j \in J, \forall r \in J, \forall k \in K \qquad (4.9)$$

$$c_{jk} \geq 0 \qquad \forall j \in J, \forall k \in K \qquad (4.10)$$

$$C_{max} \geq 0 \qquad (4.11)$$

The objective function (4.1) tries to minimize the maximum completion time at the end of the last stage also known as the makespan. Constraint (4.2) ensures that each

job is scheduled on exactly one machine of a given stage. Constraint (4.3) calculates the completion time of a job at the end of a stage by adding the processing time of this job in this stage to the completion time of this job at the end of the previous stage. In constraint (4.4), the completion time of a job at the end of the first stage is at least equal to the processing time of a job in the first stage. Constraints (4.5) and (4.6) prevent any two jobs from overlapping when they are scheduled to the same machine of a given stage. Constraint (4.7) determines the makespan value by checking the completion times of the jobs at the end of the last stage. Sign constraints (4.8) and (4.9) shows the domains of the binary decision variables, whereas sign constraints (4.10) and (4.11) shows the domains of the continuous decision variables.

Being one of the exact methods, MILP may not provide the optimal solution for each problem instance, especially for real-life size instances. Hence, the application of other methods like dispatching rules and heuristics gains more importance in manufacturing industry where HFS environment is very common.

4.2. Dispatching Rules

Dispatching rules have been studied for HFS since 1979. They are also known as construction heuristic algorithms generating initial solutions to be improved later via improvement heuristic algorithms. They are very simple to implement and also fast for making quick decisions in scheduling. They usually yield relatively good solutions in a reasonable time. Moreover, they provide optimal solutions for some special cases. Furthermore, they are classified as static and dynamic rules where dynamic rules are time dependent. In this study we use the following dispatching rules.

Service in Random Order (SIRO): A simple static dispatching rule frequently used in practice does not try to optimize any measure (Pinedo, 2016). For example, SIRO can be hybridized with First Available Machine (FAM) and Earliest Completion Time (ECT) strategies for job sequencing at a stage.

Shortest Processing Time (SPT): This simple static dispatching rule tries to minimize the average time (waiting time+processing time) that a job spends in the system, especially for a single machine, based on scheduling jobs according to non-decreasing order of their p_{jk} values (Pinedo, 2016). For example, SPT schedules the jobs according to non-decreasing order of jobs' processing times at each stage.

Shortest Total Processing Time (STPT): This static dispatching rule tries to minimize the average time based on scheduling jobs according to non-decreasing order of their total processing times thru the shop shown on Equation (4.12) (Alharkan, 2005).

$$\sum_{k=1}^{c} p_{jk} \qquad \forall j \in J \qquad (4.12)$$

For example, STPT can be hybridized with FAM and ECT strategies for job sequencing at a stage.

Longest Processing Time (LPT): A static dispatching rule which tries to minimize the makespan, especially for a single machine, based on scheduling jobs according to non-increasing order of their p_{jk} values (Pinedo, 2016). For example, LPT schedules the jobs according to the non-increasing order of jobs' processing times at each stage.

Longest Total Processing Time (LTPT): This static dispatching rule tries to minimize the makespan based on scheduling jobs according to non-increasing order of their total processing times thru the shop (Alharkan, 2005).
For example, LTPT can be hybridized with FAM and ECT strategies for job sequencing at a stage.

4.3. Renowned Heuristic Algorithms

Some heuristic algorithms in the literature are observed to be effective and efficient in terms of generating good solutions particularly for flow shop configurations. Moreover, they are also applied in HFS configurations for comparison purposes with other solution methods in the literature. In this study, in order to apply these algorithms (except Bottleneck Heuristic (BH)) with FAM and ECT strategies, the HFS configuration is reduced down to a traditional flow shop configuration through distributing the processing time of a job at each stage equally based on the number of parallel identical machines at that stage as in Equation (4.13).

$$\bar{p}_{jk}=\frac{p_{jk}}{m_k} \qquad \forall j\in J,\ \forall k\in K \qquad (4.13)$$

In Equation (4.13), $\bar{p}_{jk}$ is equal to the average processing time of job j at stage k in a traditional flow shop, as reduced from the HFS configuration.

In this study following we use the renowned heuristic algorithms for generating relatively good solutions in an acceptable amount of time:

Johnson's Algorithm: The most popular heuristic algorithm which yields the optimal solution for the flow shop problem ($F2||C_{max}$) according to the study of Johnson (1954). This algorithm also solves ($F3||C_{max}$) problem to optimality, if one of the conditions in Expression (4.14) is satisfied (Alharkan, 2005).

$$\text{Either, } \min(p_{j1})\geq\max(p_{j2}) \text{ or } \min(p_{j3})\geq\max(p_{j2}) \qquad \forall j\in J \qquad (4.14)$$

It is understood that there is no bottleneck condition for the second machine, i.e., it is dominated either by the first and/or the third machine. If at least either of these conditions is satisfied, the processing times of the jobs on the first and the second

machines are summed for creating the first dummy machine. Then, for creating the second dummy machine, the processing times of the jobs on the second and the third machines are summed. By this way, ($F3||C_{max}$) problem is converted to ($F2||C_{max}$) problem to be solved optimally. Similar to the three-machine flow shop adaptation, Johnson's algorithm is also applied to ($Fm||C_{max}$), if one of the conditions in Expression (4.15) is satisfied (Puaar, 2017).

$$\text{Either, } \min(p_{j1}) \geq \max(p_{j2}, p_{j3}, \ldots, p_{j,m-1}) \text{ or } \min(p_{jm}) \geq (p_{j2}, p_{j3}, \ldots, p_{j,m-1}) \quad \forall j \in J \quad (4.15)$$

The processing times of the jobs from machine 1 to machine m-1 and the processing times of the jobs from machine 2 to machine m are summed in order to create two dummy machines for converting the original problem to ($F2||C_{max}$) configuration to be solved to optimality by the Johnson's algorithm with the iterations which are the same as the iterations in the algorithm for the two-machine flow shop problem, if one of the conditions above is satisfied. However, even if none of the conditions above is satisfied, Johnson's algorithm is still applied as a constructive heuristic algorithm yielding not an optimal but a relatively good initial solution.

Palmer's Heuristic: This is another popular and easy to implement algorithm as a constructive heuristic for ($Fm||C_{max}$) problem in order to generate a relatively good initial solution according to the study of Palmer (1965). Palmer's heuristic consists of two steps as described below:

- **Step 1:** Calculate slope A_j for j^{th} job for ($Fm||C_{max}$) problem (Equation (4.16)).

$$A_j = -\sum_{k=1}^{m} [m-(2\times k-1)]\times p_{jk} \qquad \forall j \in J \qquad (4.16)$$

- **Step 2:** Schedule the jobs based on sequencing them in a non-increasing order according to A_j values.

In order to understand how Palmer's heuristic functions, the following example is used for illustration:

Table 4.1. Example: Palmer's heuristic for ($F3||C_{max}$) problem

j \ k	**1**	**2**	**3**	A_j
	$-(3-(2\times1-1))$	$-(3-(2\times2-1))$	$-(3-(2\times3-1))$	
1	3	8 (p_{12})	10	14
2	12	9	12	0
3	8	6	13	10
4	12	10	16	8

Table 4.2. Example: Completion times with Palmer's heuristic for ($F3||C_{max}$) problem

j \ k	**1**	**2**	**3**
1	3	11	21
3	11	17	34
4	23	33	50
2	35	44	62

Since the example problem is solved to optimality (C_{max}=62) with the Johnson's algorithm, it is seen that Palmer's algorithm has also given the optimum C_{max}.

Campbell, Dudek, and Smith (CDS) Algorithm: Actually, this is $m-1$ times application of the Johnson's algorithm to the subproblems of ($Fm||C_{max}$) problem in order to find which subproblem/s provide the best C_{max} value according to the study of Campbell et al. (1970). Therefore, it is understood that in order to solve ($Fm||C_{max}$) problem by the CDS algorithm, the number of iterations is equal to $m-1$ (Table 4.3).

Table 4.3. CDS algorithm for ($Fm||C_{max}$) problem

Iteration	Left column: Sum of processing times	Right column: Sum of processing times
1	p_{j1}	p_{jm}
2	$p_{j1}+ p_{j2}$	$p_{j,m-1}+p_{jm}$
3	$p_{j1}+ p_{j2}+ p_{j3}$	$p_{j,m-2}+ p_{j,m-1}+p_{jm}$
…	…	…
***m*-1**	$p_{j1}+ p_{j2}+ p_{j3}+\ldots+p_{j,m-1}$	$p_{j2}+\ldots+ p_{j,m-2}+ p_{j,m-1}+p_{jm}$

At each iteration, there are two dummy machines where ($Fm||C_{max}$) problem is converted to a ($F2||C_{max}$) subproblem to be solved to optimality via the Johnson's algorithm. Totally $m-1$ many ($F2||C_{max}$) subproblems are solved. As a result, the subproblem with the best C_{max} value is chosen in order to derive the best schedule for ($Fm||C_{max}$) problem. The following example illustrates the CDS algorithm:

- **Iteration 1:** Create two dummy machines with the first and the last machines for the application of the Johnson's algorithm (Table 4.4):

Table 4.4. Example: 1st iteration of the CDS algorithm for ($F3||C_{max}$) problem

j \ k	p_{j1}	p_{j3}
1	3	10
2	12	12
3	8	13
4	12	16

Sequences			
1	3	2	4

1	3	4	2

Completion times			
j \ k	**1**	**2**	**3**
1	3	11	21
3	11	17	34
2	23	32	46
4	35	45	62

j \ k	**1**	**2**	**3**
1	3	11	21
3	11	17	34
4	23	33	50
2	35	44	62

- **Iteration 2:** Create two dummy machines with $p_{j1}+p_{j2}$ and $p_{j2}+p_{j3}$ for the application of the Johnson's algorithm (Table 4.5):

Table 4.5. Example: 2^{nd} iteration of the CDS algorithm for ($F3||C_{max}$) problem

j \ *k*	$p_{j1}+p_{j2}$	$p_{j2}+p_{j3}$
1	11	18
2	21	21
3	14	19
4	22	26

Sequences			
1	3	2	4

1	3	4	2

Completion times			
j \ *k*	**1**	**2**	**3**
1	3	11	21
3	11	17	34
2	23	32	46
4	35	45	62

j \ *k*	**1**	**2**	**3**
1	3	11	21
3	11	17	34
4	23	33	50
2	35	44	62

Coincidentally, the optimal solution is found at the first iteration due to the fact that the Johnson's algorithm condition is satisfied. If none of the conditions is satisfied, one (alternative best sequences may be obtained) of the iterations provides the best sequence with the best C_{max} value.

The CDS algorithm is a good constructive heuristic, since it checks the variants of two dummy machines' structures. When the Johnson's algorithm condition is not satisfied, it is expected that the CDS algorithm yields better results, due to the fact that the CDS algorithm provides several sequences.

Nawaz, Enscore, and Ham (NEH) Algorithm: As an iterative insertion heuristic, the NEH algorithm calculates C_{max} value for each insertion in each iteration (Nawaz et al., 1983). NEH algorithm consists of the following steps:

- **Step 1:** Calculate total work content for each job (Equation (4.17)).

$$T_j = \sum_{k=1}^{m} p_{jk} \qquad \forall j \in J \qquad (4.17)$$

- **Step 2:** Sequence the jobs in non-increasing order of T_j values.

- **Step 3:** Select the first two jobs and calculate C_{max} values for partial schedules based on the positions of these jobs in the schedule (Sequence 1: 1^{st}-2^{nd}, and sequence 2: 2^{nd}-1^{st}). Choose the partial schedule with the best C_{max} value.

- **Step 4:** Pick the next job from the list. Insert this job into all possible positions of the partial schedule. Calculate C_{max} values for the new partial schedules based on the position of this job in the schedule. Suppose sequence 2 is the best partial schedule. Then sequence 1': 3^{rd}-2^{nd}-1^{st}, sequence 2': 2^{nd}-3^{rd}-1^{st}, and sequence 3': 2^{nd}-1^{st}-3^{rd}.

- **Step 5:** Choose the partial schedule in Step 4 with the best C_{max} value.

- **Step 6:** Follow this procedure from Step 4, until there is no job unsequenced.

  When all jobs are sequenced, stop the algorithm.

The following example illustrates the NEH algorithm:

- **Step 1:** See Table 4.6.

- **Step 2:** See Table 4.7.

Table 4.6. Example: 1st step of the NEH algorithm for ($F3||C_{max}$) problem

j \ k	**1**	**2**	**3**	T_j
1	3	8	10	21
2	12	9	12	33
3	8	6	13	27
4	12	10	16	38

Table 4.7. Example: 2nd step of the NEH algorithm for ($F3||C_{max}$) problem

j \ k	**1**	**2**	**3**	T_j
4	12	10	16	38
2	12	9	12	33
3	8	6	13	27
1	3	8	10	21

- **Step 3:** See Table 4.8.

Table 4.8. Example: 3rd step of the NEH algorithm for ($F3||C_{max}$) problem

Completion times			
j \ k	**1**	**2**	**3**
4	12	22	38
2	24	33	50

Completion times			
j \ k	**1**	**2**	**3**
2	12	21	33
4	24	34	50

In this step, there are two partial schedules which are alternative to each other. From now on, remainder steps of NEH algorithm are followed based on these two partial schedules.

- **Step 4:** See Table 4.9.

- **Step 5:** See Table 4.10.

Table 4.9. Example: 4^{th} step of the NEH algorithm for ($F3||C_{max}$) problem

Completion times			
j \ ***k***	**1**	**2**	**3**
3	8	14	27
4	20	30	46
2	32	41	58

Completion times			
j \ ***k***	**1**	**2**	**3**
3	8	14	27
2	20	29	41
4	32	42	58

j \ ***k***	**1**	**2**	**3**
4	12	22	38
3	20	28	51
2	32	41	63

j \ ***k***	**1**	**2**	**3**
2	12	21	33
3	20	27	46
4	32	42	62

j \ ***k***	**1**	**2**	**3**
4	12	22	38
2	24	33	50
3	32	39	63

j \ ***k***	**1**	**2**	**3**
2	12	21	33
4	24	34	50
3	32	40	63

Table 4.10. Example: 5^{th} step of the NEH algorithm for ($F3||C_{max}$) problem

Completion times			
j \ ***k***	**1**	**2**	**3**
3	8	14	27
4	20	30	46
2	32	41	58

Completion times			
j \ ***k***	**1**	**2**	**3**
3	8	14	27
2	20	29	41
4	32	42	58

- **Step 6:** See Table 4.11.

As it is seen in Table 4.11, the NEH algorithm provides alternative optimal solutions for this particular problem.

Although the NEH algorithm is more time consuming than the CDS algorithm for the same example problem, it is expected that the NEH algorithm provides better solution than the CDS algorithm when Johnson's algorithm condition is

not satisfied. It is obvious that, due to its iterative insertion method, the NEH algorithm produces relatively better solutions.

Table 4.11. Example: 6^{th} step of the NEH algorithm for ($F3||C_{max}$) problem

Completion times			
j \ ***k***	**1**	**2**	**3**
1	3	11	21
3	11	17	34
4	23	33	50
2	35	44	62

Completion times			
j \ ***k***	**1**	**2**	**3**
1	3	11	21
3	11	17	34
2	23	32	46
4	35	45	62

j \ ***k***	**1**	**2**	**3**
3	8	14	27
1	11	22	37
4	23	33	53
2	35	44	65

j \ ***k***	**1**	**2**	**3**
3	8	14	27
1	11	22	37
2	23	32	49
4	35	45	65

j \ ***k***	**1**	**2**	**3**
3	8	14	27
4	20	30	46
1	23	38	56
2	35	47	68

j \ ***k***	**1**	**2**	**3**
3	8	14	27
2	20	29	41
1	23	37	51
4	35	47	67

j \ ***k***	**1**	**2**	**3**
3	8	14	27
4	20	30	46
2	32	41	58
1	35	49	68

j \ ***k***	**1**	**2**	**3**
3	8	14	27
2	20	29	41
4	32	42	58
1	35	50	68

Bottleneck Heuristic (BH): According to the study of Paternina Arboleda et al. (2008), it is also called Theory of Constraints (TOC)-based heuristic for an HFS scheduling problem. The steps of this heuristic are described below:

Step 1: Identify the bottleneck stage:

- For each stage k, the flow ratio is computed (Equation (4.18)).

$$FR_k = \sum_{j=1}^{n} \frac{p_{jk}}{m_k} \qquad \forall k \in K \qquad (4.18)$$

- Stage with the maximum FR_k value is chosen as the bottleneck stage. Let b denote the bottleneck stage.

- Release time of job j for stage b is calculated (Equation (4.19)).

$$R_j = \sum_{k=1}^{b-1} p_{jk} \qquad \forall j \in J \qquad (4.19)$$

- Due date of job j for stage b is calculated (Equation (4.20)).

$$D_j = \sum_{k=1}^{c} FR_k - \sum_{k=b+1}^{c} p_{jk} \qquad \forall j \in J \qquad (4.20)$$

Step 2: Sequence the bottleneck stage:

- Schedule the jobs in non-decreasing order of R_j. If there is a tie, rank the jobs in non-decreasing order of D_j. If there is a tie again, rank the jobs in non-decreasing order of processing times.

- Schedule the jobs on the machines of the bottleneck stage according to the preceding ranking.

Step 3: Sequence the non-bottleneck stages:

- Stages before the bottleneck stage *b*: Schedule the jobs in non-decreasing order of D_j. If there is a tie, rank the jobs in non-decreasing order of R_j. If there is a tie again, rank the jobs in non-decreasing order of processing times.

- Stages after the bottleneck stage *b*: Schedule the jobs according to FAM and ECT strategies.

In order to comprehend BH, following problem is used as an example (Table 4.12):

Table 4.12. Example (*HF3, P2, 1, P2*$||C_{max}$) problem for BH

j \ ***c***	**1**	**2**	**3**
1	3	8	10
2	12	9	12
3	8	6	13
4	12	10	16
PP	2	1	2

Step 1: See Tables 4.13, 4.14, and 4.15.

Table 4.13. Example: Flow ratio table for each stage

c	**1**	**2**	**3**
$\boldsymbol{FR_k}$	17.5	33	25.5

- According to Table 4.13, stage 2 is identified as the bottleneck stage.

Step 2: See Table 4.16.

Table 4.14. Example: Release times of jobs for the bottleneck stage

j	R_j
1	3
2	12
3	8
4	12

Table 4.15. Example: Due dates of jobs for the bottleneck stage

j	D_j
1	66
2	64
3	63
4	60

Table 4.16. Example: Table for scheduling the bottleneck stage

Stage 2					**PP 1**	
j	**PP 1**	R_j	D_j	p_{j2}	**start**	**finish**
1	[1]	3	66	8	3	11
2	[4]	12	64	9	27	36
3	[2]	8	63	6	11	17
4	[3]	12	60	10	17	27

Step 3: See Table 4.17.

- Since stage 3 is scheduled according to FAM and ECT strategies, the job sequence for this stage is *J*1-*J*3-*J*4-*J*2. Therefore, the final schedule is shown in Table 4.18.

  This example problem is solved by the proposed CP model (in Chapter 5 below) to optimality, and the optimal makespan value is found to be 48.

Hence, BH provides the optimal solution for this example problem as seen in the table.

Table 4.17. Example: Scheduling the stage before the bottleneck stage

Stage 1						**PP 1**		**PP 2**	
j	**PP 1**	**PP 2**	R_j	D_j	p_{j1}	**start**	**finish**	**start**	**finish**
1	[1]		0	3	3	0	3		
2		[4]	0	27	12			8	20
3		[2]	0	11	8			0	8
4	[3]		0	17	12	3	15		

Table 4.18. Example: Final schedule with the BH

Completion times			
j \ ***c***	**1**	**2**	**3**
1	3	11	21
3	8	17	30
4	15	27	43
2	20	36	48
PP	2	1	2

In the following chapter, we discuss the solution approaches that we propose in this study:

- A Constraint Programming (CP) model as an exact method
- A Hybrid Algorithm (HA) as a heuristic method
- A Galactic Swarm Optimization (GSO) as a metaheuristic method

CHAPTER 5

PROPOSED SOLUTION METHODS

We propose a Constraint Programming (CP) model, a Hybrid Algorithm (HA), and a Galactic Swarm Optimization (GSO) metaheuristic algorithm in order to provide stabilized efficiency and especially effectiveness for HFS scheduling problems.

Different from the MILP model in terms of definitions, but the same in terms of the assumptions, in the CP model, instead of binary and continuous decision variables, the processing time of a job at a stage is modelled as an interval length decision variable having the size of the processing time of the job at the stage. Moreover, the relationships between these operations are modeled with precedence constraints. The assignments of these operations to parallel identical machines at each stage are modeled with the cumulative function of the CP model as a resource constraint. This CP model is inspired by the model based on the study of Laborie et al. (2011).

Furthermore, we propose a Hybrid Algorithm (HA) that consists of three phases. In the first phase, the HA calculates a Global Lower Bound (GLB) value as in the study of Santos et al. (1995) in order to measure the quality of the solution. In the second phase, in order to obtain the random order of the jobs determined initially at the beginning of the first stage, two dispatching rules, FAM and ECT, particularly powerful for HFS scheduling are used at the beginning of each stage. The reason why these dispatching rules FAM as a machine allocation strategy and ECT as a job sequencing strategy at each stage are chosen is that FAM maximizes machine utilization and ECT minimizes job idleness simultaneously in order to shorten the makespan. By this way, the schedule which is generated by the algorithm approaches to a non-delay schedule for the purpose of obtaining near optimal or optimal makespan

value if possible. The makespan value is then compared to the GLB. If the makespan is observed to be equal to the GLB, then the optimal schedule is obtained. Most of the time, this case is observed for small to some medium instances within an acceptable computational time. The larger the problem size, the more difficult the problem becomes due to the NP-hardness property of the HFS scheduling problem. On the other hand, if the makespan value is observed to be close to the GLB, then a near-optimal schedule is obtained with a makespan value within a relatively acceptable gap from the GLB value. This case is commonly observed for the problems of medium to large sizes. In medium size instances, the gap from the GLB is relatively smaller than the gap in large size instances. Therefore, it is obvious that the larger the problem size is, the bigger the gap is. In the third and last phase of the HA, the sequence of this generated schedule with the best available makespan value for the test problem is recorded for comparison purposes.

We propose another solution method for the HFS scheduling problem which is a new metaheuristic approach inspired by the motion of stars and superclusters inside galaxies, based on the study of Muthiah-Nakarajan and Noel (2016). This metaheuristic is called Galactic Swarm Optimization (GSO) which balances exploration and exploitation phases for a proper global search. In this chapter, GSO is explained in detail with all of its phases. Similar to HA, GSO also uses FAM and ECT strategies to form its cost function in order to compute the makespan value for a given HFS scheduling problem. Similar to the HA, the performance of GSO varies according to the problem instance size in terms of effectiveness and efficiency.

In the following sections, we elaborate more on these three methods proposed.

5.1. Constraint Programming (CP) Model

The CP model (Laborie et al., 2011) is a new method to find solutions for scheduling and other combinatorial optimization problems. In order to deal with the complexity

of real-world problems for especially large-scale scheduling problems, the CP model becomes a powerful and invaluable tool. Rather than using an imperative programming language, the CP model uses a declarative programming language which simplifies the scheduling of jobs to machines with resource constraints. The automatic search algorithm of the CP model is complete, and it uses Tree Search (Depth First) and Constraint Propagation. This automatic search algorithm starts with first reducing the set of possible values in the domain of decision variables according to constraint propagations. When any further reduction is not possible in the domain of decision variables, the CP model backtracks according to depth search and starts the whole procedure in order to find a feasible or a better solution for the objective function value.

The following example is used as an illustration for understanding the CP model with its depth search and constraint propagation strategies based on Google OR-Tools (2018):

How can 4 queens be placed on a 4x4 chessboard so that no two of them attack each other? (In chess, a queen can attack horizontally, vertically, and diagonally.)

Placing the first queen in the upper left corner reduces the domain of the objective functions with the application of constraint propagation. Then, the second queen is placed and thus the domain of the objective functions is reduced again with constraint propagation. After placing the third queen, it is seen that this solution is infeasible due to constraint propagation according to the location of the third queen shown in Table 5.1.

Due to the fact that the solution above is infeasible, the location of the second queen is changed. Constraint propagation is repeated according to the new location of the second queen. After placing the third queen, it is seen that this solution is infeasible

due to constraint propagation according to the new location of the third queen shown in Table 5.2.

Table 5.1. Example: 1st iteration of CP

Q	**X**	**X**	**X**
X	**X**		
X		**X**	
X			**X**

Q	**X**	**X**	**X**
X	**X**	**Q**	**X**
X	**X**	**X**	**X**
X		**X**	**X**

Q	**X**	**X**	**X**
X	**X**	**Q**	**X**
X	**X**	**X**	**X**
X	**Q**	**X**	**X**

Table 5.2. Example: 2nd iteration of CP

Q	**X**	**X**	**X**
X	**X**		
X		**X**	
X			**X**

Q	**X**	**X**	**X**
X	**X**	**X**	**Q**
X		**X**	**X**
X	**X**		**X**

Q	**X**	**X**	**X**
X	**X**	**X**	**Q**
X	**Q**	**X**	**X**
X	**X**	**X**	**X**

Since an infeasible solution is encountered again, the location of the second queen is changed once more. Constraint propagation is applied again according to the new location of the second queen. After placing the third queen, it is seen that this solution is infeasible due to constraint propagation according to the new location of the third queen shown in Table 5.3.

Table 5.3. Example: 3rd iteration of CP

Q	**X**	**X**	**X**
X	**X**		
X		**X**	
X			**X**

Q	**X**	**X**	**X**
X	**X**	**X**	
X	**Q**	**X**	**X**
X	**X**	**X**	**X**

Q	**X**	**X**	**X**
X	**X**	**X**	**Q**
X	**Q**	**X**	**X**
X	**X**	**X**	**X**

Because the third iteration of the CP approach yields an infeasible solution, in the fourth, fifth and sixth iterations, the second queen is placed at new available locations.

However, a feasible solution is not reached in any of the iterations. By this iterative manner, depth search is completed for the first location of the first queen as shown in Table 5.4.

Table 5.4. Example: 4^{th}, 5^{th} and 6^{th} iterations of CP

4^{th}	Q	X	X	X		Q	X	X	X		Q	X	X	X
	X	X				X	X	X	X		X	X	X	X
	X		X			X	X	X	Q		X	X	X	Q
	X			X		X		X	X		X	Q	X	X
5^{th}	Q	X	X	X		Q	X	X	X		Q	X	X	X
	X	X				X	X		X		X	X	Q	X
	X		X			X	X	X			X	X	X	X
	X			X		X	Q	X	X		X	Q	X	X
6^{th}	Q	X	X	X		Q	X	X	X		Q	X	X	X
	X	X				X	X	X			X	X	X	Q
	X		X			X	X	X	X		X	X	X	X
	X			X		X	X	Q	X		X	X	Q	X

In the seventh iteration, the first queen is placed to the intersection of the first row and the second column as its new location. According to this new location of the first queen, the second queen is placed at the first available location. By placing each queen at the available location, constraint propagation is applied according to these placements. Then, the third queen is placed at the available location. After the last constraint propagation is completed according to the location of the third queen, the fourth queen is placed at the final position as shown in Table 5.5.

By synchronously using depth search and constraint propagation, the CP approach drastically decreases the memory usage and the solution time. Therefore, the CP approach seems to be promising, when MILP model is inefficient to optimally solve especially real-life size instances.

Table 5.5. Example: 7^{th} iteration of CP

X	Q	X	X
X	X	X	
	X		X
	X		

X	Q	X	X
X	X	X	Q
	X	X	X
	X		X

X	Q	X	X
X	X	X	Q
Q	X	X	X
X	X		X

X	Q	X	X
X	X	X	Q
Q	X	X	X
X	X	Q	X

For our HFS scheduling problem, following expressions are used in the CP model to represent the corresponding expressions described in the MILP model:

The CP model

Sets

J: number of jobs: $j=\{1, \ldots, n\}$

K: number of stages: $k=\{1, \ldots, c\}$

L: number of parallel identical machines at each stage: $l=\{1, \ldots, m_k\}$

Parameters

p_{jk}: processing time of job j at stage k

Decision Variables

op_{jk}: the interval length of job j's operation time at stage k

Objective Function

$$\text{minimize } \max_{j \in J}(endOf(op_{jk})) \tag{5.1}$$

Constraints

subject to

$$endBeforeStart(op_{j,k-1}, op_{jk}) \qquad \forall j \in J, \forall k \in K: k>1 \qquad (5.2)$$

$$\sum_{j=1}^{n} pulse(op_{jk},1) \leq m_k \qquad \forall k \in K \qquad (5.3)$$

In this CP model, decision variable is described as an interval. The functions *endOf* and *endBeforeStart*, represent the precedence relationships of the interval length decision variables. Finally, *pulse* is the critical function representing the number of parallel identical machines as a resource constraint which is a cumulative function of the CP model. If this problem were a traditional flow shop scheduling problem, instead of constraint (5.3) of the CP model, constraint (5.4) would be used in the CP model.

$$noOverlap(op_{jk}) \qquad \forall j \in J, \forall k \in K \qquad (5.4)$$

In the CP model, the difference between the constraint (5.3) model and the constraint (5.4) is illustrated in Figure 5.1.

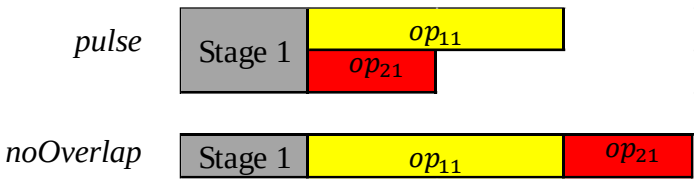


Figure 5.1. *pulse* vs. *noOverlap* in the CP model

Pulse makes operations of the jobs overlap at a given stage on the parallel identical machines. On the other hand, the function *noOverlap* sequences the operations of jobs for a given stage without overlapping. If the problem were a traditional flow shop, *noOverlap* does exactly what is expected from it. However, our problem is an HFS

scheduling problem and thus we actually want the operations of jobs to overlap at a given stage until the parallel identical machines at this stage are filled up with these operations. Figure 5.2 shows that how exactly *pulse* works for the HFS scheduling problem with two jobs (j=1, 2) at stage 1 having two parallel identical machines (PP 1 and PP 2).

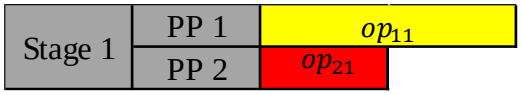


Figure 5.2. The role of *pulse* cumulative function in the CP model for the HFS scheduling problem

Due to the fact that op_{jk} is the interval length decision variable, it takes only positive integer values which cover (4.4), (4.10), and (4.11) domain constraints in the MILP model. Moreover, in order to describe this interval length decision variable, the processing times of the jobs whose units of measure are hours are converted to minutes. Since the allocations of jobs to machines are performed with a depth search and constraint propagation in the CP model, (4.8) and (4.9) sign constraints in the MILP model are satisfied after the optimal solution of the CP model is obtained. Objective function (5.1) in the CP model corresponds to objective function (4.1) in the MILP model, covers the constraint (4.7) and tries to minimize the makespan. Constraint (5.2) in the CP model is similar to constraint (4.3) in MILP model and covers it. Constraint (5.3) in the CP model covers constraints (4.2), (4.5), and (4.6) in the MILP model. Constraint (5.3) allows the jobs to be scheduled one by one up to the number of parallel identical machines at a given stage. Therefore, the number of jobs processed in parallel at a given stage never exceeds the number of parallel identical machines at that stage.

In order to understand the CP model, the example HFS scheduling problem in the study of Santos et al. (1995) is optimally solved with this CP model. The example

problem is represented in Table 5.6 and a Gantt Chart generated by the CP Optimizer of IBM ILOG CPLEX 12.6 is shown in Figure 5.3.

Table 5.6. The example problem (*HF3, P2||C_{max}*)

j \ c	**1**	**2**	**3**
1	3	5	9
2	7	1	4
3	2	7	4
4	8	2	2
5	6	3	7
PP	2	2	2

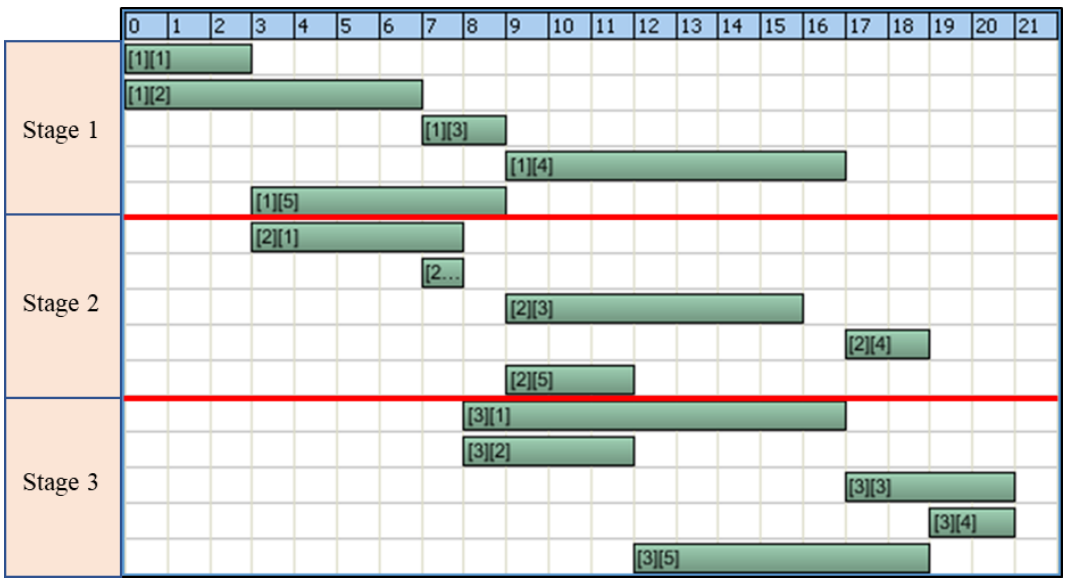


Figure 5.3. Gantt Chart of the example problem solution generated by CP Optimizer

At the first stage, jobs 1 and 2 are scheduled to the two parallel identical machines, concurrently. Interval [1][1] (op_{11}) is represented with [stage][job] structure to obtain stage-based Gantt Chart from the CP Optimizer of IBM ILOG CPLEX 12.6. After job 1 is completed on the first machine of the first stage, on this machine, job 5 is scheduled. Job 3 and 4 are scheduled consecutively, after job 2 is completed on the

second machine of the first stage. At the second stage, jobs 1 and 2 are scheduled to parallel identical machines. After job 1 is completed on the first machine and job 2 is completed on the second machine at the second stage, jobs 3 and 5 are scheduled to these machines, concurrently. After the completion of job 4 at the first stage, it can be scheduled to either one of the machines at the second stage, however, according to FAM strategy, it is scheduled on the second machine at the second stage. At the last stage, again, jobs 1 and 2 are scheduled to the parallel identical machines, concurrently. After job 1 is completed on the first machine at the last stage, job 3 is scheduled. Jobs 5 and then 4 are scheduled to the second machine at the last stage, after job 2 is completed. By this way, the schedule having the shortest length (21 units) is obtained for this example problem, which is (*HF3, P2*||C_{max}) of Santos et al. (1995).

5.2. Hybrid Algorithm (HA)

We propose an HA inspired by the GLB of Santos et al. (1995). With the inclusion of the GLB, the solution quality of HA heuristic is always under control, even though the job sequence is randomly generated. In the following sections, we explain the concept of the HA in detail.

5.2.1. Global Lower Bound (GLB)

The development of the GLB value is executed by a stage-based approach. For each stage, the lower bound value is denoted by LB(k), k=1, …, c. Moreover, there is also a job-based lower bound which is denoted by LB(0). Then the GLB value is equal to the maximum value of a set consisting of LB(0) and LB(k) values for all k=1, …, c. LB(0) and LB(k) are explained in Equations (5.5) and (5.6).

$$\mathrm{LB}(0)=\max_{j\in J}\left(\sum_{k=1}^{c} p_{jk}\right) \tag{5.5}$$

$$\mathrm{LB}(k)=\frac{1}{m_k}\times\left(\sum_{j=1}^{m_k} LSA_{jk}+\sum_{j=1}^{n} p_{jk}+\sum_{j=1}^{m_k} RSA_{jk}\right) \qquad \forall k\in K \qquad (5.6)$$

LSA is the left-hand side total processing times for jobs from stage 1 to $k-1$ sequenced in non-decreasing order represented by Equation (5.6.1). Moreover, *RSA* is the right-hand side total processing times for jobs from stage $k+1$ to c sequenced in non-decreasing order represented by Equation (5.6.2).

$$LSA_{jk}=\sum_{k'=1}^{k-1} p_{jk'} \text{ else } 0 \text{ if } k=1 \qquad \forall j\in J,\ \forall k\in K:\ 1<k\leq c \qquad (5.6.1)$$

$$RSA_{jk}=\sum_{k'=k+1}^{c} p_{jk'} \text{ else } 0 \text{ if } k=c \qquad \forall j\in J,\ \forall k\in K:\ 1\leq k<c \qquad (5.6.2)$$

Equation (5.5) represents the calculation of the job-based lower bound value which is equal to the maximum of all job-based lower bound values. Equation (5.6) shows all of the stage-based lower bound values' calculations. If the processing times of the jobs are integer, then $\mathrm{LB}(k)$ value is simply rounded up. Finally, the GLB is the maximum value of a set consisting of $\mathrm{LB}(0)$ and $\mathrm{LB}(k)$ values as in Equation (5.7).

$$\mathrm{GLB}=\max\left[\mathrm{LB}(0),\ \max_{k\in K}(\mathrm{LB}(k))\right] \qquad (5.7)$$

For better comprehension, the following example (Table 5.7) is used to illustrate the computation of the GLB:

- **Step 1:** LB(0) computation (Table 5.8).

- **Step 2:** Sum of the processing times of the jobs for each stage (Table 5.9).

Table 5.7. Example: (*HF3, P2||C_{max}*) problem for GLB computation

j \ *c*	**1**	**2**	**3**
1	3	8	10
2	12	9	12
3	8	6	13
4	12	10	16
PP	2	2	2

Table 5.8. Example: (*HF3, P2||C_{max}*) problem job-based lower bound computation

j \ *c*	**1**	**2**	**3**	**LB(0)**
1	3	8	10	21
2	12	9	12	33
3	8	6	13	27
4	12	10	16	38
PP	2	2	2	

Table 5.9. Example: (*HF3, P2||C_{max}*) problem sum of the processing times of the jobs

j \ *c*	**1**	**2**	**3**
1	3	8	10
2	12	9	12
3	8	6	13
4	12	10	16
Total	35	33	51
PP	2	2	2

- **Step 3:** Lower bound calculations for each stage (Tables 5.10, 5.11, and 5.12).

LB(1)=(1/2)×[0+35+(18+19)]=36

Table 5.10. Example: (*HF3, P2||C_{max}*) problem calculation of LB(1)

j \ *c*	**1**	**2+3**
1	3	18
2	12	21
3	8	19
4	12	26
Total	35	
PP	2	

j \ *c*	**1**	**2+3**
1	3	18
3	8	19
2	12	21
4	12	26
Total	35	
PP	2	

LB(2)=(1/2)×[(3+8)+33+(10+12)]=33

Table 5.11. Example: (*HF3, P2||C_{max}*) problem calculation of LB(2)

j \ *c*	**1**	**2**	**3**
1	3	8	10
2	12	9	12
3	8	6	13
4	12	10	16
Total		33	
PP		2	

j \ *c*	**1**	**2**	**3**
1	3	8	10
3	8	6	13
2	12	9	12
4	12	10	16
Total		33	
PP		2	

j \ *c*	**1**	**2**	**3**
1	3	8	10
2	12	9	12
3	8	6	13
4	12	10	16
Total		33	
PP		2	

LB(3)=(1/2)×[(11+14)+51+0]=38

- **Step 4:** Global lower bound calculation.

GLB=max(38, max(36, 33, 38))=38

The schedules in Figures 5.4 and 5.5 are generated by the proposed HA in order to show the lowest gap value possible.

These schedules makespan values' gap away from the GLB is calculated as in Equation (5.8).

Table 5.12. Example: (*HF3*, *P2*||C_{max}) problem calculation of LB(3)

j \ *c*	1+2	3
1	11	10
2	21	12
3	14	13
4	22	16
Total		51
PP		2

j \ *c*	1+2	3
1	11	10
3	14	13
2	21	12
4	22	16
Total		51
PP		2

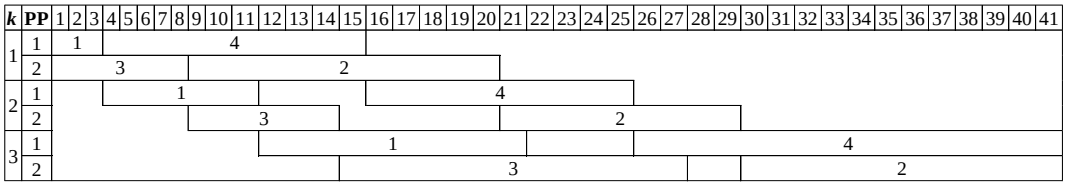


Figure 5.4. Gannt chart of sequence *J*1-*J*3-*J*4-*J*2

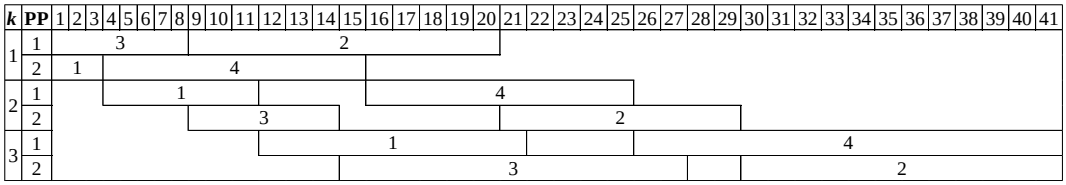


Figure 5.5. Gannt chart of sequence *J*3-*J*1-*J*4-*J*2

$$\text{Gap}=\frac{C_{max}\text{-GLB}}{\text{GLB}} \qquad (5.8)$$

For these schedules, gap value is equal to 7.89 %. Since the optimal makespan value is greater than or equal to the GLB value, there is a probability that this gap is zero.

Therefore, this problem is also solved with the CP model, and the makespan value is found that it is equal to 41 which is thr optimal solution for this problem.

5.2.2. First Available Machine (FAM) and Earliest Completion Time (ECT) Strategies

FAM is a dynamic strategy that is always chasing the machines at a stage to see the machine with the earliest finish time among the parallel identical machines for a given stage. Therefore, FAM simply outperforms other methods for machine allocation to jobs, since FAM is a dynamic strategy depends on the time. By means of FAM strategy, machine utilization increases. Table 5.13 is used to illustrate the FAM dispatching rule.

Table 5.13. Example: The application of FAM strategy for the first stage

Stage 1				***J*3-*J*1-*J*4-*J*2**
PP			**FT**	
1	→	**J3**	8	
2	→	**J1**	3	
2	→	**J4**	15	
1	→	**J3**	8	
1	→	**J2**	20	
2	→	**J4**	15	

For job sequence *J*3-*J*1-*J*4-*J*2, finish times at machine 1 and 2 are respectively equal to p_{31}=8 and p_{11}=3 for the first stage. In order to process job 4 at stage 1, the machine with minimum finish time is selected. Therefore, job 4 is loaded to machine 2 after it completes the processing of J1. New finish time for machine 2 is equal to 3+p_{41}=15. According to the sequence, job 2 is the last job to be processed at stage 1. For this purpose, job 2 is loaded to machine 1 after it completes the processing of job 3, since

machine 1 has the minimum finish time (8≤15) compared to machine 2's finish time which is now equal to 15. At the end, machine 1's finish time is updated. The finish time of machine 1 is now equal $8+p_{21}=20$.

ECT is the job sequencing strategy at the beginning of each stage after FAM dispatching rule is applied at the previous stage. The aim of ECT job sequencing rule is to minimize job waiting time. In this way, with FAM and ECT rules applied together to an HFS problem, the makespan value is tried to be decreased as much as possible. Based on the definition of a non-delay schedule, with this proposed Hybrid Algorithm (HA), the makespan value of any HFS problem approaches to the optimal solution as much as possible. ECT job sequencing rule is illustrated through an example in Table 5.14.

Table 5.14. Example: The application of ECT job sequencing rule at stage 2

FAM	
Stage 1	**Completion times**
3	8
1	3
4	15
2	20
PP	2

ECT
→

FAM	
Stage 2	**Completion times**
1	11
3	14
4	25
2	29
PP	2

For job sequence *J*3-*J*1-*J*4-*J*2, the completion times of the jobs at the end of the first stage are calculated according to FAM dispatching rule. As a result, job 1 has the shortest completion time at the end of the first stage, despite the fact that it is scheduled to the first stage at the 2^{nd} order of the sequence. Using the ECT job sequencing rule, the order of jobs changes and thus updates as *J*1-*J*3-*J*4-*J*2 at the beginning of the second stage. Since, coincidentally, the completion times of the jobs result in a non-decreasing order at the end of the second stage, ECT keeps this same sequence at the

beginning of the third stage. Ties are broken with the machine having the lowest index for FAM dispatching rule and with the job having the lowest index for ECT rule.

The gap value determines how good the generated schedule's makespan value is. The sequence of the jobs at the beginning of the first stage is selected randomly, the fitness of which depends on how far the makespan value is from the GLB. Therefore, in order to ensure that the proposed algorithm is not stuck in the local optima, the job sequence at the first stage is randomly generated for each iteration as a random search method. In the first iteration, for a given HFS problem, the GLB value is calculated. Then, by using randomly generated sequence at the first stage, makespan value is calculated with FAM and ECT strategies for this problem. After that, calculated makespan value is checked whether it is equal to the GLB value or not. If it is equal to the GLB value, then the solution reached is certainly optimal. Otherwise, the gap value is calculated, and the next iteration is initiated. In the next iteration, for a new randomly generated sequence at the first stage, new makespan value is calculated. If this new makespan value is smaller than GLB(1+Gap) value, this new sequence is accepted as a better solution, thus, gap value is updated for the next iteration according to this new makespan value. This loop continues until the stopping criterion is reached. In the HA, the stopping criterion is the number of iterations. This gap strategy is the reason why this proposed HA is powerful and easy to implement for any variant of HFS scheduling problems.

5.2.3. Pseudo Code and Flowchart of the Hybrid Algorithm (HA)

The proposed HA for (*HFc, Pm||Cmax*) problems is presented in the following pseudo code where π represents the sequence of the jobs.

Hybrid Algorithm (HA)	
1:	Load HFS problem data
2:	Calculate LB(0) job-based LB
3:	Calculate *LSA*

Hybrid Algorithm (HA) (cont'd)

Calculate *RSA*
Calculate LB(*k*) stage-based LB
GLB=max(LB(0), LB(*k*))
Initialization: generate π randomly
Scheduling the first stage:
For j=1:m_1
    Allocate the empty machines for the jobs up to m_1
    Find the completion times of jobs up to m_1
    Find the FTs of machines
EndFor
For j=m_1+1:n
    Find the machine with the minimum FT
    Allocate the FAM for the remaining jobs
    Find the completion times of remaining jobs
    Update the FTs of machines
EndFor
Scheduling the other stages:
For k=2:c
    Apply the ECT based on the completion times of jobs at the previous stage
    Based on the ECT, reorder the jobs which is a new sequence at the current stage
    For j=1:m_k
        Allocate the empty machines for the jobs up to m_k
        Find the completion times of jobs up to m_k
        Find the FTs of machines
    EndFor
    For j=m_k+1:n
        Find the machine with the minimum FT
        Allocate the FAM for the remaining jobs
        Find the completion times of remaining jobs
        Update the FTs of machines
    EndFor
EndFor
Calculate C_{max}
If C_{max}=GLB then C_{max} is optimal and **STOP**
Else Gap=(C_{max}−GLB)/GLB
    Set the counter and the number of iterations
    do
        Initialization: generate π' randomly
        Scheduling the first stage:
        For j=1:m_1
            Allocate the empty machines for the jobs up to m_1
            Find the completion times of jobs up to m_1
            Find the FTs of machines
        EndFor
        For j=m_1+1:n
            Find the machine with the minimum FT
            Allocate the FAM for the remaining jobs
            Find the completion times of remaining jobs
            Update the FTs of machines
        EndFor
        Scheduling the other stages:
        For k=2:c

Hybrid Algorithm (HA) (cont'd)

```
56:             Apply the ECT based on the completion times of jobs at the previous stage
57:             Based on the ECT, reorder the jobs which is a new sequence at the current stage
58:             For j=1:m_k
59:                 Allocate the empty machines for the jobs up to m_k
60:                 Find the completion times of jobs up to m_k
61:                 Find the FTs of machines
62:             EndFor
63:             For j=m_k+1:n
64:                 Find the machine with the minimum FT
65:                 Allocate the FAM for the remaining jobs
66:                 Find the completion times of remaining jobs
67:                 Update the FTs of machines
68:             EndFor
69:         EndFor
70:         Calculate C_max'
71:         If C_max'<GLB(1+Gap)
72:             C_max= C_max'
73:             Gap=(C_max−GLB)/GLB
74:         EndIf
75:     While counter<iteration
76: EndIf
77: Return C_max
78: STOP
```

The flowchart of the proposed HA heuristic is as well presented in Figure 5.6.

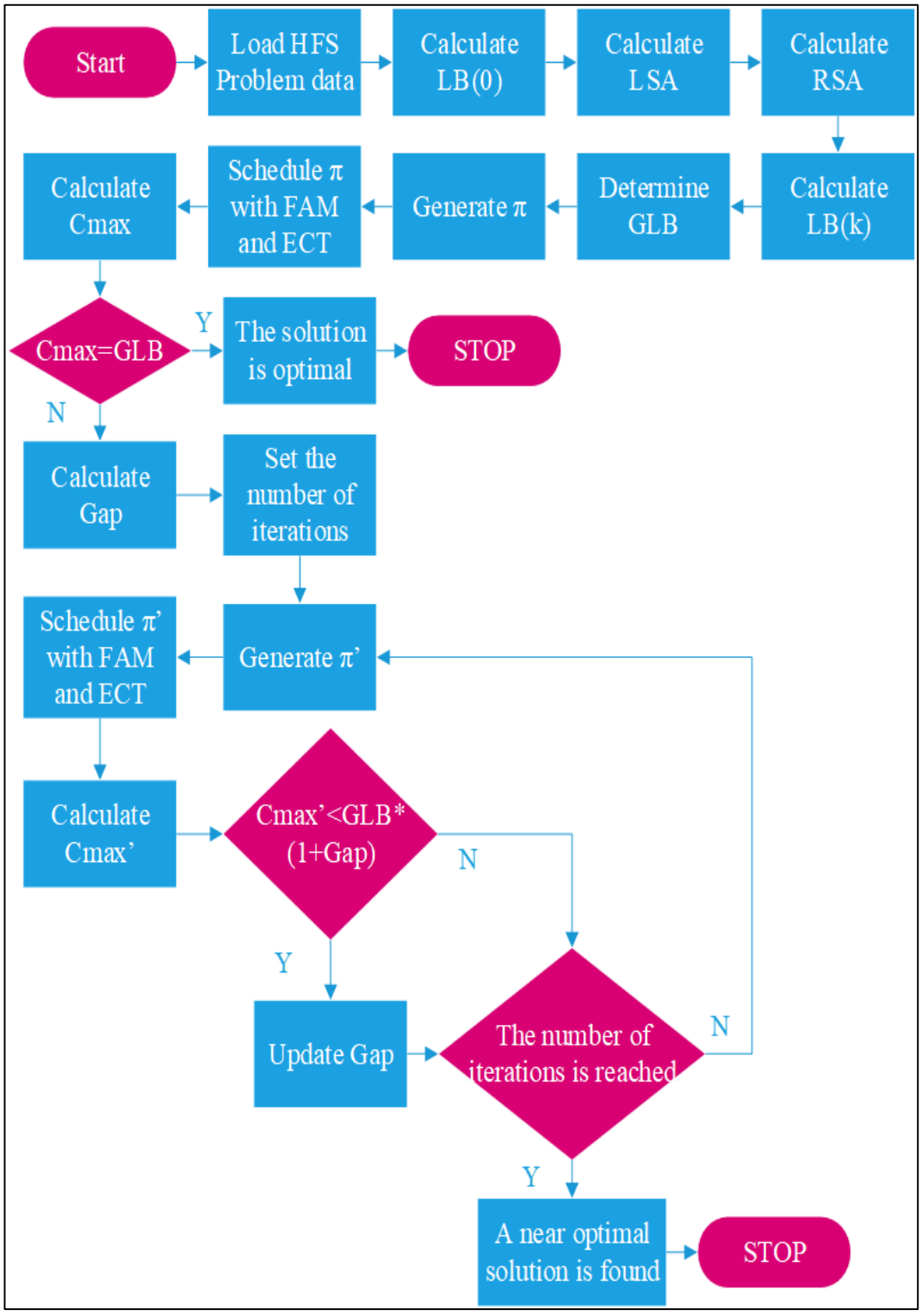


Figure 5.6. The flowchart of the HA

5.3. Galactic Swarm Optimization (GSO)

GSO is a new evolutionary based metaheuristic algorithm inspired by the motion of stars and super-clusters inside galaxies in order to find the global optima of a given optimization problem based on the study by Muthiah-Nakarajan and Noel (2016). This metaheuristic consists of two levels, adjusting the balance between exploration (diversification) and exploitation (intensification) phases for a given optimization problem. In many metaheuristic algorithms, in order to explore better solutions or in other words not to be stuck in local optima while exploiting, a lot of parameter optimization processes are required. For instance, in GA, the number of iterations, population size, crossover and mutation rates, and the selection strategy should be considered carefully. On the other hand, the structure of GSO has already been designed to manage the trade-off between diversification and intensification. Moreover, GSO is such a flexible metaheuristic that, in its two phases, different types of metaheuristic algorithms can be used such as GA, TS, ACO and SA.

5.3.1. GSO in HFS Scheduling

In this study, in the levels of GSO metaheuristic, PSO method by Eberhart and Kennedy (1995) is used in order to let the stars and the super-clusters in the galaxy find better solutions. Like particles in PSO, in GSO, each star and super-cluster has its own position and velocity values. In the first level of GSO, in order to update the velocity of a star s at time t+1 Equation (5.9) is used.

$$v_{xs}^{t+1}=w_1 v_{xs}^{t}+c_1 r_1\left(p_{xs}^{t}-x_s^{t}\right)+c_2 r_2\left(g_{xs}^{t}-x_s^{t}\right) \tag{5.9}$$

In Equation (5.10), L_1 represents the number of iterations in the first level of GSO and k represents the current iteration value. In Equation (5.9), c_1 is the cognitive acceleration coefficient which makes star s to learn from its best personal position where p_{xs}^{t} is equal to the best personal position of star s at time t and x_s^{t} is equal to the

position of star s at time t. Similarly, c_2 is the social acceleration coefficient which makes stars s learn from the global best position where g_{xs}^t is equal to the global best position at time t. In Equation (5.9), r_1 and r_2 are uniformly distributed random numbers between 0 and 1. In Equation (5.9), w_1 is the inertia weight determining the balance between local and global searches. In the first level of GSO, w_1 is linearly decreasing according to the Equation (5.10). After calculating v_{xs}^{t+1}, the position of star s at time t+1 is computed via Equation (5.11).

$$w_1=1-k/(L_1+1) \tag{5.10}$$

$$x_s^{t+1}=x_s^t+v_{xs}^{t+1} \tag{5.11}$$

After completing the first level, or in other words, the exploration phase, the second level of GSO is initiated. Global best positions from the first level are passed to the second level of GSO in order to form a super-cluster. Now, the exploitation phase is starting with the application of PSO once again to the stars in this super-cluster. From now on, the position of star s in super-cluster at time t is represented by y_s^t. Similarly, the velocity of star s in super-cluster at time t is represented by v_{ys}^t. Therefore, Equation (5.12) is the new velocity update equation.

$$v_{ys}^{t+1}=w_2v_{ys}^t+c_3r_3\left(p_{ys}^t-y_s^t\right)+c_4r_4(g_{ys}^t-y_s^t) \tag{5.12}$$

In Equation (5.13), k is the current iteration value whereas L_2 is the number of iterations in the second level of GSO. Similar to c_1 and c_2 in the first level, c_3 and c_4 are cognitive and social acceleration coefficients in the second level of GSO, respectively. r_3 and r_4 are random numbers between 0 and 1. Like w_1 in the first level, w_2 is the inertia weight in the second level of GSO. It also decreases linearly according to the Equation (5.13). After calculatingv_{ys}^{t+1}, in order to compute the position of star s in super-cluster, Equation (5.14) is used.

$$w_2=1-k/(L_2+1) \quad (5.13)$$

$$y_s^{t+1}=y_s^t+v_{ys}^{t+1} \quad (5.14)$$

In this study, initial position and velocity values are generated randomly according to the Equations (5.15) and (5.16) based on the study of Taşgetiren et al. (2007).

$$x_s^0=x_{\min}+(x_{\max}-x_{\min})rand() \quad (5.15)$$

$$v_{xs}^0=v_{\min}+(v_{\max}-v_{\min})rand() \quad (5.16)$$

In Equation (5.15), $x_{\min}=0$, $x_{\max}=4$and in Equation (5.16), $v_{\min}=-4$, $v_{\max}=4$. On the other hand, continuous velocity values are restricted in the range [−4, 4]. v_{ys}^0 is also restricted in the same range. Moreover, and again based on the study of Taşgetiren et al. (2007), c_1, c_2, c_3, and c_4 are equal to 2. Furthermore, w_1 and w_2 values start from "1" and they decrease linearly according to the number of iterations in the first and the second levels of the GSO based on the Equations (5.10) and (5.13), respectively. However, they are never to be decreased below 0.4. The number of iterations in the first level of the GSO, L_1 is equal to the number of jobs n. Similarly, the number of stars in each cluster inside the galaxy is also equal to the number of jobs n. However, in order to increase diversification, the number of clusters in the galaxy is set to n+5. Similarly, the number of iterations in the second level of the GSO, L_2 is also set to n+5.

Since the position values of stars are continuous but not discrete, they turn out to be insufficient to represent the decision variables for a combinatorial optimization problem. Due to the fact that HFS scheduling problem is a combinatorial optimization problem with the positions of the jobs as its decision variables, Smallest Position Value (SPV) rule based on the study of Taşgetiren et al. (2007) is applied to the position values of stars. In order to comprehend how SPV works, Table 5.15 is used for an illustrative example:

Table 5.15. Example: The application of SPV rule

Job	1	2	3	4	5
Position values	2.922	−2.574	−1.426	1.251	2.402
Job Position	5	1	2	3	4
Job Sequence (π)	2	3	4	5	1

According to the updated position values in Table 5.15, job 2 has the smallest position value. Therefore, its job position is equal to 1. Job 2 is followed by job 3 having the second smallest position value. Hence, it takes the job position 2. The third smallest position value belongs to job 4. Thus, job position 3 belongs to it. Job 5 has the fourth smallest position value and, so, it takes the job position 4. Finally, job 1 has the largest position value among all. Therefore, it takes the last job position. By this way, we have a candidate job sequence for stage 1 to be evaluated.

Similar to the HA, the GSO uses FAM and ECT in order to form its cost function. Therefore, the makespan value of each job permutation π alternative is calculated in the same way.

At the end of the second iteration in the GSO, in order to improve the solution quality without hindering the solution time, Insertion Heuristic (IH) and Local Search (LS) methods are applied to the global solution obtained. By this way, it is thought that if there is a chance for improvement, then global solution can be improved. In the IH, a job is inserted to different job positions one by one by swapping the job in that position. If the new job permutation is better than the previous one in terms of makespan value, global job permutation is updated, thus, new job permutation is made to be equal to it. This process continues until each job is inserted to each job position except inserted job's current job position. Therefore, the number of iterations for the IH is determined by the number of jobs *n*. When a better job permutation is obtained, the procedure goes on by trying the next job instead of restarting. Otherwise, solution time is extended, if the procedure is restarted when encountered with a better

permutation. The IH builds up its search method based on providing a better permutation. Otherwise, it keeps the previous permutation as the global solution and, naturally, it tries to improve this solution. After the IH is completed, swap mutation which is randomly changing two jobs' positions at a time, is applied to the latest global solution as an LS method. The number of iterations in the LS is equal to L_2 which is thought to be sufficient to obtain a better permutation, if possible, without hindering the solution time. The GSO is a powerful metaheuristic providing a good global solution. Therefore, the IH and LS methods have a slight effect on that solution. However, solution time is not drastically hindered by these methods, hence, they are still applied to that solution in order to find hopefully a better one.

In the next section of the proposed GSO metaheuristic, pseudo code of the approach is presented in order to clarify the mechanism of this method.

5.3.2. Pseudo Code of the GSO Metaheuristic

In order to clarify the proposed GSO for (*HFc, Pm||Cmax*) problem, the following pseudo code is designed.

Galactic Swarm Optimization (GSO)

Initialization: $x_{\min}=0$, $x_{\max}=4$, $v_{\min}=-4$, $v_{\max}=4$, $c_1=c_2=c_3=c_4=2$
For 1:n+5
  For 1:n
    Randomly generate x_s^0 & $p_{xs}^0 \sim rand(x_{\min}, x_{\max})$ and $v_{xs}^0 \sim rand(v_{\min}, v_{\max})$
    Apply SPV to x_s^0 and p_{xs}^0
    If $C_{max}(x_s^0) < C_{max}(p_{xs}^0)$
      $p_{xs}^0 = x_s^0$
    EndIf
  EndFor
  $g_{xs}^0 = p_{xs}^0(1)$
  For 2:n
    If $C_{max}(p_{xs}^0) < C_{max}(g_{xs}^0)$
      $g_{xs}^0 = p_{xs}^0$
    EndIf
  EndFor
EndFor

Galactic Swarm Optimizaton (GSO) (cont'd)

galaxy=g_{xs}^0(1)
For 2:n+5
  If $C_{max}(g_{xs}^0)$<C_{max}(galaxy)
    galaxy=g_{xs}^0
  EndIf
EndFor
Level 1 (Exploration):
For 1:n+5
  For 0:L_1-1
    Equation (5.10) and check w_1<0.4 condition
      For 1:n
        r_1 & r_2~*rand*()
        Equation (5.9) and restrict v_{xs}^{t+1}~[-4, 4]
        Equation (5.11) and apply SPV x_s^{t+1}
        If $C_{max}(x_s^{t+1})$<$C_{max}(p_{xs}^{t+1})$
          p_{xs}^{t+1}=x_s^{t+1}
            If $C_{max}(p_{xs}^{t+1})$<$C_{max}(g_{xs}^{t+1})$
              g_{xs}^{t+1}=p_{xs}^{t+1}
                If $C_{max}(g_{xs}^{t+1})$<C_{max}(*galaxy*)
                  galaxy=g_{xs}^{t+1}
                EndIf
            EndIf
        EndIf
      EndFor
  EndFor
EndFor
Forming the super-cluster and initialization: $v_{\min}$=-4, $v_{\max}$=4
For 1:n+5
  y_s^0=g_{xs}^{t+1}
  v_{ys}^0~*rand*($v_{\min}$, $v_{\max}$)
  p_{ys}^0=y_s^0
EndFor
Level 2 (Exploitation):
For 0:L_2-1
  Equation (5.13) and check w_2<0.4 condition
    For 1:n+5
      r_3 & r_4~*rand*()
      Equation (5.12) and restrict v_{ys}^{t+1}~[-4, 4]
      Equation (5.14) where *galaxy*=g_{ys}^{t+1} and apply SPV y_s^{t+1}
      If $C_{max}(y_s^{t+1})$<$C_{max}(p_{ys}^{t+1})$
        p_{ys}^{t+1}=y_s^{t+1}
        If $C_{max}(p_{ys}^{t+1})$<C_{max}(*galaxy*)
          galaxy=p_{ys}^{t+1}
        EndIf
      EndIf
    EndFor
EndFor
Return *galaxy*

Galactic Swarm Optimization (GSO) (cont'd)

Initialization of the Insertin Heuristic (IH):
galaxy_copy=*galaxy*
For *i*=1:*n*
  For *j*=1:*n*
    If *i*≠*j*
      galaxy(*j*)=*galaxy_copy*(*i*)
      galaxy(*i*)=*galaxy_copy*(*j*)
      If C_{max}(*galaxy*)<C_{max}(*galaxy_copy*)
        galaxy_copy=*galaxy* (update)
      Else *galaxy*=*galaxy_copy* (reset)
      EndIf
    EndIf
  EndFor
EndFor
Initialization of the Local Search (LS) (Swap mutation):
For 1:L_2
r=*randperm*(*length*(*galaxy_copy*)) (random permutation of jobs' indices)
g=*galaxy_copy* (copy *galaxy_copy*)
g([*r*(1) *r*(2)])=*galaxy_copy*([*r*(2) *r*(1)]) (randomly select two indices to swap jobs on them)
  If C_{max}(*g*)<C_{max}(*galaxy_copy*)
    galaxy_copy=*g*
  EndIf
EndFor
Return *galaxy_copy*
STOP

CHAPTER 6

CASE STUDY IN THE AEROSPACE COMPANY

In this chapter, we attempt to propose a Hybrid Flow Shop (HFS) configuration to the company which is currently operating as a Hybrid Job Shop (HJS). Furthermore, we present alternative methods to schedule the designed HFS so as to meet the demand for the required panels of the A320 fuselage.

For the case study, first of all, an HFS configuration is required to be designed. Since the current manufacturing environment is an HJS with longer lead times and complex material handling systems, an HFS configuration turns out to be necessary for shorter lead times and less complex material handling systems than the previous one. In this chapter, for better comprehension of these problems in the case study, the current manufacturing system operating as an HJS is explained in detail.

In order to convert the current HJS manufacturing environment to an HFS configuration, firstly, all required data such as demand, capacity, production and machine availability information are collected.

Secondly, after data collection is completed, in the data analysis and interpretation, due to the fact that processing times and machine availability data are stochastic, we fit each of them to the distribution with the best goodness value. Before fitting machine availability data to the best distribution, we cleanse them from the outliers falling far away from the conglomerated values according to their plots. Since processing times data are less polluted with the outliers considered as negligible according to their plots, we directly fit each of processing times data to the best available distribution by skipping the data cleansing process.

Thirdly, after data analysis and interpretation are completed, Cycle Time (CT) is calculated according to demand and capacity data. The source module of Discrete-Event Simulation (DES) model creates each type of panel according to this cycle time. From production data, routes of the parts are obtained to be realized in DES model as an HFS configuration. The processing times of the jobs and the breakdowns of the machines are defined in DES model according to the outputs of data analysis and interpretation.

Then, with Minimum number of Machines (MoM) calculation for each stage, the number of parallel identical machines is determined for each stage according to the results from the DES model. By this way, HJS environment is converted to an HFS configuration. In MoM calculations, if demand per year is satisfied for each type of panel, then there is no need for an additional machine for any stage. However, if there is a bottleneck stage and/or demand is not satisfied then, the number of parallel identical machines is increased by one unit for this stage according to the results of DES model runs.

After the conversion of HJS to an HFS configuration by means of the DES model, the job sequences obtained by the solution methods are inserted to the source module. Then DES model is run for a single production of each panel (job) in order to obtain the makespan value of each job sequence inserted to the source module. The solution method for HFS scheduling problem that gives the best makespan value identifies the best solution method for the case study.

In order to understand how an HFS configuration is converted from HJS environment by means of the DES model, the phases of this process are explained explicitly in the following sections.

6.1. Products: Panels of Fuselage

The company is Turkey's technology leader in design, development, modernization, manufacturing, integration, and life cycle support of integrated aerospace systems with a remarkable number of products from fixed and rotary wing air platforms to Unmanned Air Vehicles (UAV)s and satellites.

The company, ranking among the top hundred global companies in aerospace and defense industry, based its business on six strategic areas which are Aerostructure, Aircraft, Helicopter, UAV Systems, Space Systems, and National Combat Aircraft groups with the provision of related integrated logistics support.

Today, the company is conducting A320 Section-18 Panels (ABS) of Fuselage and AIRBUS-PAG SA Section-19 Shells & Barrel (S19) Programs which are two of the projects under the Aerostructure Group.

The main and strategic assembly parts (load items) of these two programs are the panels. Before completion of the assembly process, the panels are called "skins" (detail parts) in the manufacturing area. In ABS project, there are seven types of panels which are upper, lower 1, lower 2, left forward side, right forward side, left rear side, and right rear side skins. On the other hand, in S19 project, there are six types of panels which are upper middle, upper left, upper right, lower left, lower right and lower middle skins. In skin manufacturing, the batch size of each panel is equal to one, due to the difficulties of materials handling in Hybrid Job Shop (HJS) configuration, the size of panels being too big to be maneuvered in an HJS configuration, and in the assembly of the fuselage, the usage of each panel is only one unit. Table 6.1 lists the panels making up the center fuselage of Airbus A320 with the potential annual demand values.

Table 6.1. Panels of the center fuselage of Airbus A320

			Panels of Sections	Project ABS	Description	Annual Demand
Final Product: Airbus A320	**Sub-aseembly Part: Center Fuselage**	**Section-18**	1	S18	UPPER SKIN	960
			2		LOWER SKIN 1	
			3		LOWER SKIN 2	
			4		LEFT FORWARD SIDE SKIN	
			5		RIGHT FORWARD SIDE SKIN	
			6		LEFT REAR SIDE SKIN	
			7		RIGHT REAR SIDE SKIN	
		Section-19	8	S19	UPPER MIDDLE SKIN	720
			9		UPPER LEFT SKIN	
			10		UPPER RIGHT SKIN	
			11		LOWER LEFT SKIN	
			12		LOWER RIGHT SKIN	
			13		LOWER MIDDLE SKIN	

In order to manufacture the panels, several operations (each corresponding to a separate stage of manufacturing) must be performed on the panels taking into account the precedence relationships among these operations. Except the raw material issue, in order to manufacture a single panel, 19 sequential operations shown on Table 6.2 are performed.

Table 6.2. Operations routing for any panel manufacturing

Stage No	Stage Name
0	Raw Material Issue (Cycle Time)
1	First Cut
2	Roll
3	Clean Ops1 (Alkali Clean1 & Vapor Degrease1)
4	Heat Treatment
5	Refrigerator
6	Stretch Press

Continued on next page

Table 6.2. – *Continued from previous page*

Stage No	Stage Name
7	Clean Ops2 (Alkali Clean2 & Vapor Degrease2)
8	Deburr1, Drill Hole & Remove Tab
9	Mechanic Mill
10	Vapor Degrease3
11	Hand Form
12	Deburr2 & Hand Finish
13	Conductivity, Dimensional & Hardness Inspection
14	Pre-Penetrant Etch Ops
15	Non-Destructive Penetrant Inspection
16	Mask & Wet Blast & Surface Inspection Ops
17	Tartaric Sulfuric Acid Anodize Ops
18	Paint Ops
19	Paint Inspection

One of the major problems is that the panels are carried out and in five different buildings in order to manufacture them. If the revisit is counted as well, total number of buildings that panels visit is six. Moreover, complex materials handling is used both during operations and transportation among buildings. Due to the fact that the physical manufacturing area is too large to control, there are lots of quality and coordination problems that cause assembly line to stop occasionally.

Furthermore, due to the lack of quality, some of the operations, especially manual finish tasks require more time than their standard processing times. Therefore, any excessive work on the panels extends the manufacturing lead time which causes unplanned stoppages and bottlenecks not only in the assembly line, but also in the manufacturing process itself. Because the panels are manufactured in an HJS environment, there is always accumulating WIP in front of the machines. These WIP accumulations make a stack of parts where they scrap each other by scratching. Since the panels have bigger sizes, unbalanced WIP accumulations in both manufacturing and assembly areas hinder the movement of the other parts. Also, they affect the quality of other parts, negatively, because of the frictions.

Another disadvantage of the existing HJS configuration in panel manufacturing is that it is almost impossible to purchase and locate new machines in parallel to the existing ones without altering the layout. The HJS configuration currently has almost no free space to make small adjustments such as paralleling the bottleneck machines and installing smart materials handling methods including conveyor belts, Automated Guided Vehicles (AGV) or Automated Storage-Retrieval Systems (AS-RS) rather than local cranes, carts, trolleys and dollies. Paralleling machines is useful for not only handling bottleneck operations, but also offering volume flexibility and streamlining the operations in the form of a flow line in case of increasing demand and product variety.

In this case study, we first address the conversion of the current HJS environment as an HFS configuration and then the scheduling of manufacturing in this converted HFS configuration:

- A Discrete-Event Simulation (DES) model is developed and utilized so as to obtain the minimum number of parallel identical machines at each manufacturing stage in the HFS, taking into account the practical restrictions and the expected demand value.

- For the HFS scheduling problem part of the case study, we utilize several solution methods for scheduling the HFS with the objective of minimizing the makespan, expecting a growing demand for panels in the near future.

6.2. Production Processes

For each panel, the production process starts with issuing related raw material. The specification of raw material, such as the measure of raw material already matching with its required stock size to manufacture the panels and whether heat treatment has already been applied to raw material or not, affects the first cut and heat treatment

processes. Briefly, under both conditions, first cut and heat treatment operations are not required in order to prepare raw material for metallic manufacturing. However, there is a slight difference among the panels 1, 2 and 3. While both operations are not applied to panel 1, only first cut operation is applied to panels 2 and 3. On the other hand, these conditions are not valid for the other panels. But, if stock-sized material has already been available, raw material is sometimes provided without first cut process for S19 panels 9, 10, 11, and 12 under the condition of urgent demand. However, most of the time, this approach is impractical due to the high price of raw material.

After the completion of the issuance of raw material, production goes on with first cut operations. In this operation, firstly, raw material is loaded to the related manual cutting machine from its pallet. After that, with the help of saw and ruler on the machine, raw material is cut in order to bring its dimensional measures to the required stock size. In the final phase of first cut operations, stock-sized raw material is unloaded from the machine and loaded to an empty pallet previously prepared with the help of a forklift and a hooked crane during cutting operation until all of the empty pallets are loaded with materials one by one. These operations form stages 0 and 1 that are executed in Building 200 (B200) where the warehouse of raw material and first cut machines are located. These stages are shown on the partial layout of B200 in Figure 6.1.

After the completion of the first cut operations, the pallets filled with the panels shown in Figure 6.2 are loaded to a truck with the support of a forklift in order to transport stock-sized materials from one location to another, because first cut operations are performed in a different building which is the main warehouse of the raw materials. Even though the first cut operations are under the roof of metallic manufacturing, the other operations belonging to this class are still performed in a different building which is the structural manufacturing and assembly facility.

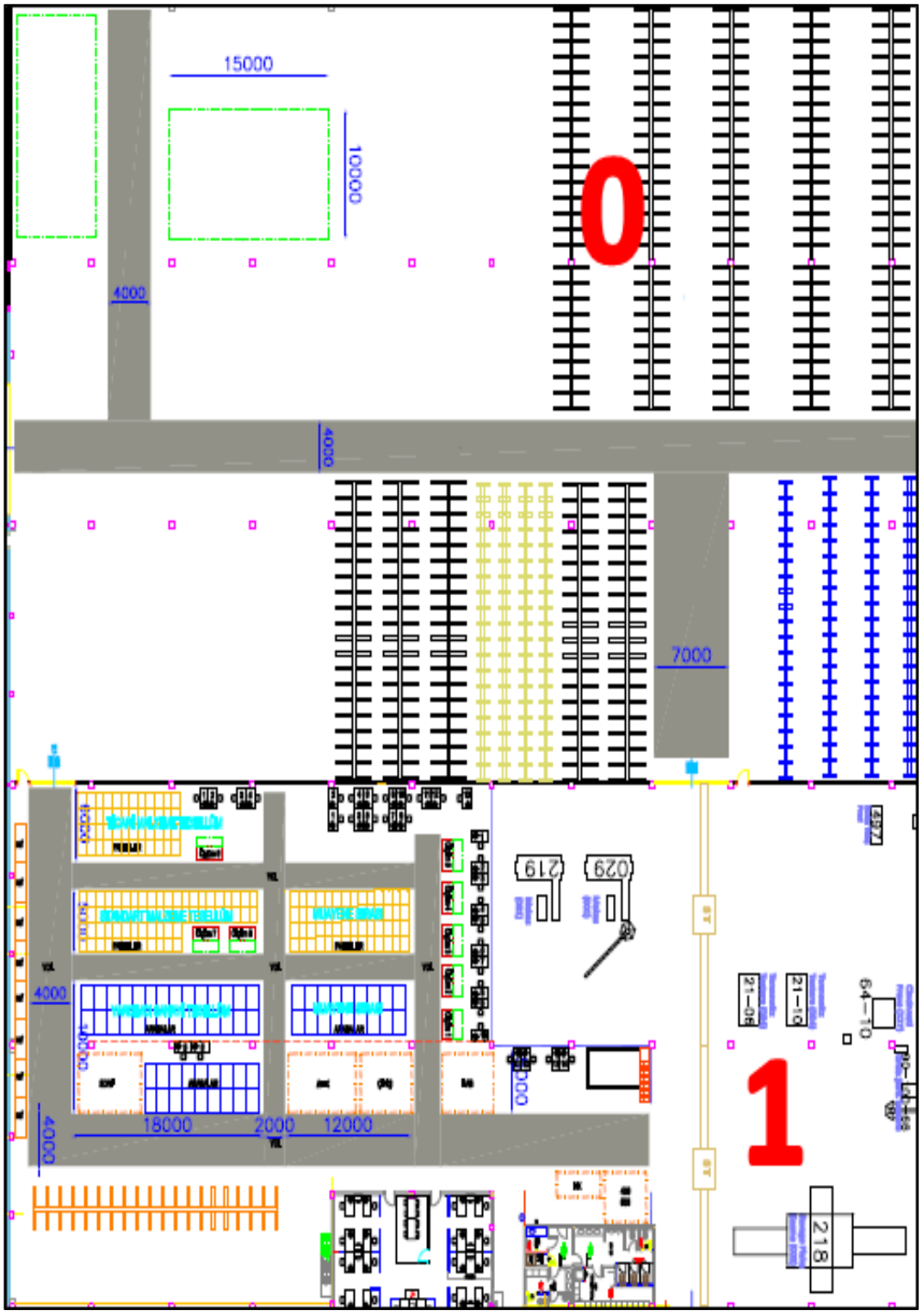


Figure 6.1. Production stages 0 and 1 in Building B200

Therefore, a transportation operation is required, however, if current HJS environment is converted to an HFS configuration, this waste can be eliminated.

Figure 6.2. Loaded pallet to be transported with the related work orders

When transportation operation is completed, the chips on the panels are removed by alkali clean and rinse tanks with the help of vacuum manipulator shown in Figure 6.3. On the other hand, if current HJS environment were converted to HFS configuration, excessive materials handling tools and motions waste would have been removed.

First, with the vacuum manipulator, the panel is lifted from the pallet and loaded to an empty transportation cart previously prepared. Then, by the moving crane, transportation cart is sinked into alkali clean tank and then it is lifted and sinked into rinse tank, in succession. By this way, cleansing operations are completed before the heat treatment process which is the succeeding stage.

Now, the panels are located on transportation carts as a batch whose quantity varies from 4 to 8 according to their thickness value. The panels are first loaded to oven

loading apparatus with the help of a vacuum manipulator and fixed crane, due to the fact that this process is performed while the panels are still flat.

Figure 6.3. Alkali clean vacuum manipulator with the fixed crane

And again, with the help of the moving crane, oven loading apparatus is lifted and loaded to oven in order to make the panels reach their required finish condition in a predetermined period of oven time. After dwelling time is up, oven loading apparatus is removed from oven via the moving crane. Then, the panels are reloaded to transportation carts via vacuum manipulator and fixed crane. At the final stage of this process, transportation carts are transferred to the refrigerator by technicians, manually, in order to complete cooling process of the panels for a predetermined period of cooling time. This transportation requires materials handling manually which can be reduced or even eliminated in an HFS environment.

Before removing transportation carts from the refrigerator manually after the completion of the cooling process, the related stretch form tool is loaded to the stretch form machine with the help of the mega crane. Each panel has its own unique stretch form tool which is designed, manufactured, and dedicated to the panel's particular geometric structure. In an HFS conversion, there needs to be more than one stretch form machine. However, batch size of each panel type must be one only due to the high cost of stretch form tools. By this way, excessive tool cost is eliminated. Furthermore, in order to initiate assembly operations of the fuselage without stopping, each type of panel is to be completed in the same manufacturing line, one after the other.

The panels, removed from the refrigerator, are loaded to the stretch form machine one by one after each panel's stretch process is completed. But, this time, due to the fact that the panels are formed, they are transferred from regular transportation carts to the formed panels' transportation carts when the panels are unloaded from the stretch machine with the help of the mega crane. After that, alkali clean and rinse processes are repeated. However, because the materials handling is slightly different from previous process, loading and unloading processes also differ. Because different types of transportation carts require different types of materials handling systems, non-value-added costs drastically increase. Therefore, an HFS conversion is thought to decrease the non-value-added costs in terms of automated materials handling systems implementation.

With the completion of cleansing process, transportation carts are transferred to drilling and removing tab area in order to prepare the panels for mechanical milling process. In order to connect the panels to related milling fixture on mechanical milling machine, the panels are drilled. With the help of fixed crane and robotic cutting machine shown in Figure 6.4. Although the tabs of the panels are required for the stretching process, they are actually excess materials on the panels which are removed for further processes. Therefore, the tabs of the panels are removed with the help of

the same machine. Then, again, the panels are unloaded from the robotic cutting machine and loaded to the formed panels' transportation carts with the help of the fixed crane.

After transporting the panels to the mechanical milling machine, with the help of the fixed crane, the panels are loaded to the mechanical milling machine. There are two entrance doors each with its own fixed crane at both sides of the machine. A panel is loaded to its milling fixture which is slided into the machine from a door with the help of constant crane. While the loaded panel is processed by the machine, without waiting for the completion time of this panel's milling process, the other panel is prepared outside of the machine (external setup) and loaded to its milling fixture which is slided into the machine from the other door with the help of the opposite fixed crane after unloading the processed panel.

Figure 6.4. Robotic cutting machine

By this way, loading and unloading operations of the panels are following each other without leaving the machine idle. Due to the fact that it takes too much time to process a single panel on this machine, this external setup operation is pretty useful for increasing the daily output. After converting the HJS environment to an HFS configuration, it may not be feasible to parallel this milling machine due to the high cost of it. Similar to the stretch form machine, batch size must be again equal to 1 in order to avoid excessive milling fixture tool cost.

After the completion of mechanical milling process, alkali cleaning and rinsing processes are repeated. Then, for hand forming and deburring operations, the panels are transported to hand finish area. If there is a forming problem on the panel, it is corrected by manual hammering. After hammering operation, chips from previous operations and traces from hammering operation are deburred in order to increase the surface quality of the panels for incoming dimensional inspection which also checks the surface quality and the correction of the panel. Hand finish processes are applied to both sides of a panel with the help of specially designed carts according to the panel's form and geometric structure.

Firstly, inner area of the panel is deburred and dimensionally inspected. After that, outer area of the panel is deburred and dimensionally inspected. By this way, hand finish and dimensional inspection processes are completed consecutively.

Before transporting parts to the facility of chemical processes, parts are loaded to the building-to-building transportation carts for further processes.

The manufacturing stages above are all located in Building 10 (B10) which is the core facility including both manufacturing and assembly operations. The sequence of the stages executed in B10 is shown on its partial layout in Figure 6.5.

As it can be seen in Figure 6.5, the HJS environment causes a lot of non-value-added activities adversely affecting the lead time and the quality of the panels. Therefore, an HFS configuration must be implemented in order to decrease materials handling complexity and thus lowering lead times and improving quality.

After the panels are transported to the chemical processes facility, they are loaded to the white carts with the help of the moving crane and manual labor for further processes. By this way, empty building-to-building carts are sent back for maintaining the loop between metallic production in the structural manufacturing facility and surface preparation in the chemical processes facility.

At this point of the manufacturing process, the panels are in another building which is the third different facility named as Building 20 (B20). Converting HJS the environment to an HFS configuration, wastes resulting from waiting, transportation, and quality can be decreased by eliminating the panel movements among buildings via different types of transportation carts. Especially, quality defects may be reduced, some of which cause a panel to be reworked for unpredictable rework times increasing manufacturing lead time or even stopping a panel which leads to a shortage in the assembly line depending on the severity of error. Some of these defects can be tolerable and corrected by rework process whereas some of them can make a panel totally scrap and permanently useless.

Before non-destructive inspection, which is also called penetrant inspection, pre-penetrant etch operation is applied to the panels in order to increase the surface quality. Since this is a chemical process, rinse operation must also be applied afterwards. These two consecutive operations have their own tanks in which the panels are sinked into one by one with the help of the moving crane in order to meet the required dwelling times.

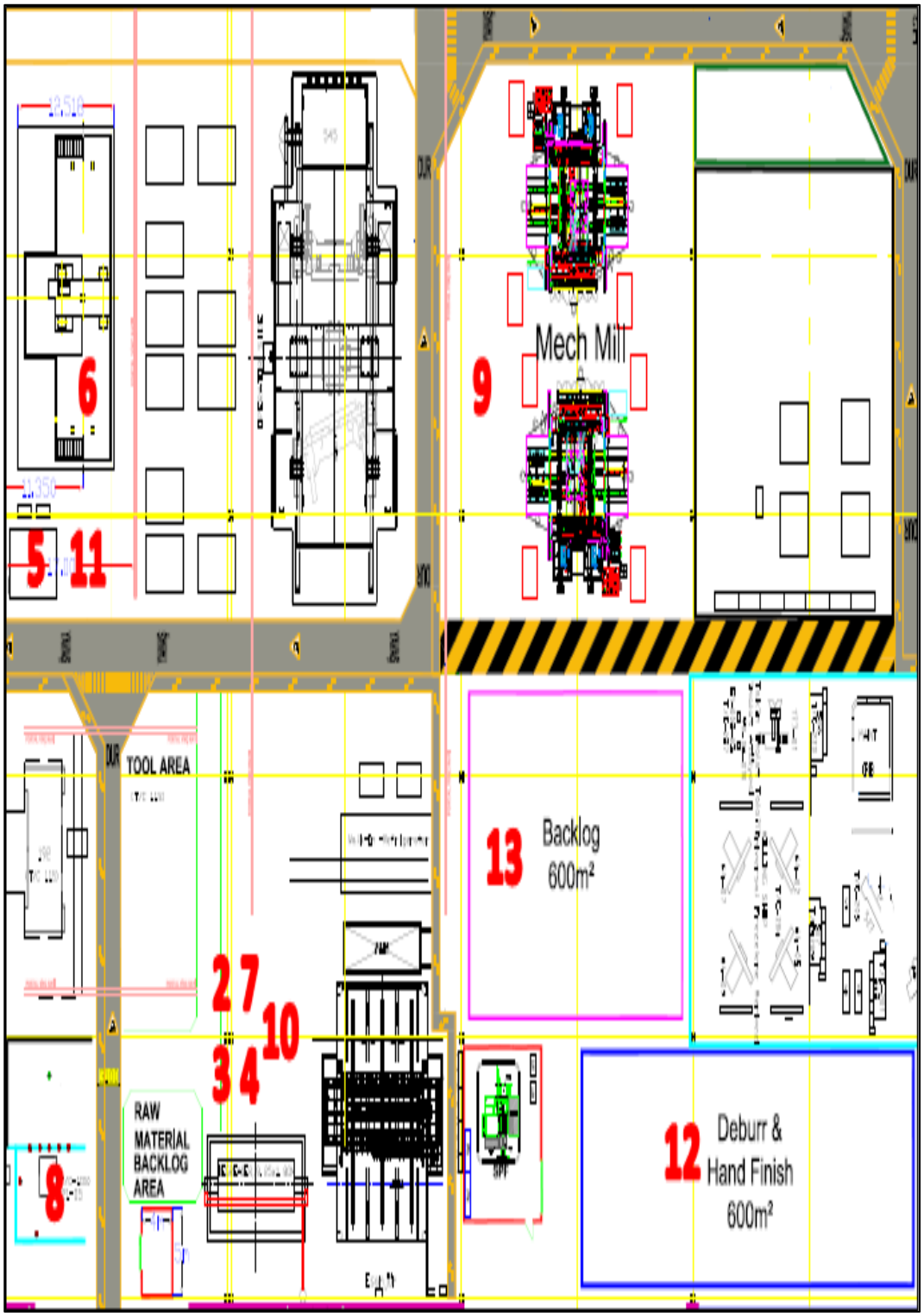


Figure 6.5. Metallic manufacturing stages in Building B10

After pre-penetrant etch and its rinse operation, the panels are sinked into infiltration liquid for penetrant inspection. After completion of penetrant inspection, the panels are transported to wet blast operation area. In this process, the panels are put into a cabinet with a spraying system, one by one after masking process is completed.

After surface quality is satisfied with surface inspection, for other chemical processes, the panels are ready to be transported to another building which is the fourth different facility named as Building 220 (B220) and has much bigger tanks than the tanks of B20. In this facility, tartaric sulfuric acid anodize with its rinse operation, sealant with its rinse operation and drying in oven operation are applied to the panels, successively. Surface operations applied to the panels are shown in Figures 6.6 and 6.7, on the partial layouts of B20 and B220, respectively.

All of the operations related to the painting process are applied to the panels in the painting process facility which is the fifth and the final building utilized in the manufacturing process of the panels. Painting process operations are executed in Building 40 (B40) as shown in Figure 6.8.

After all painting operations are completed, under the condition that the panels are suitable for the assembly line according to the final inspection, they are sent back to B10 where assembly of the panels takes place, including riveting and mating, packaging and final inspection. Assembly of the panels so as to obtain the center fuselage is out of the scope of this study.

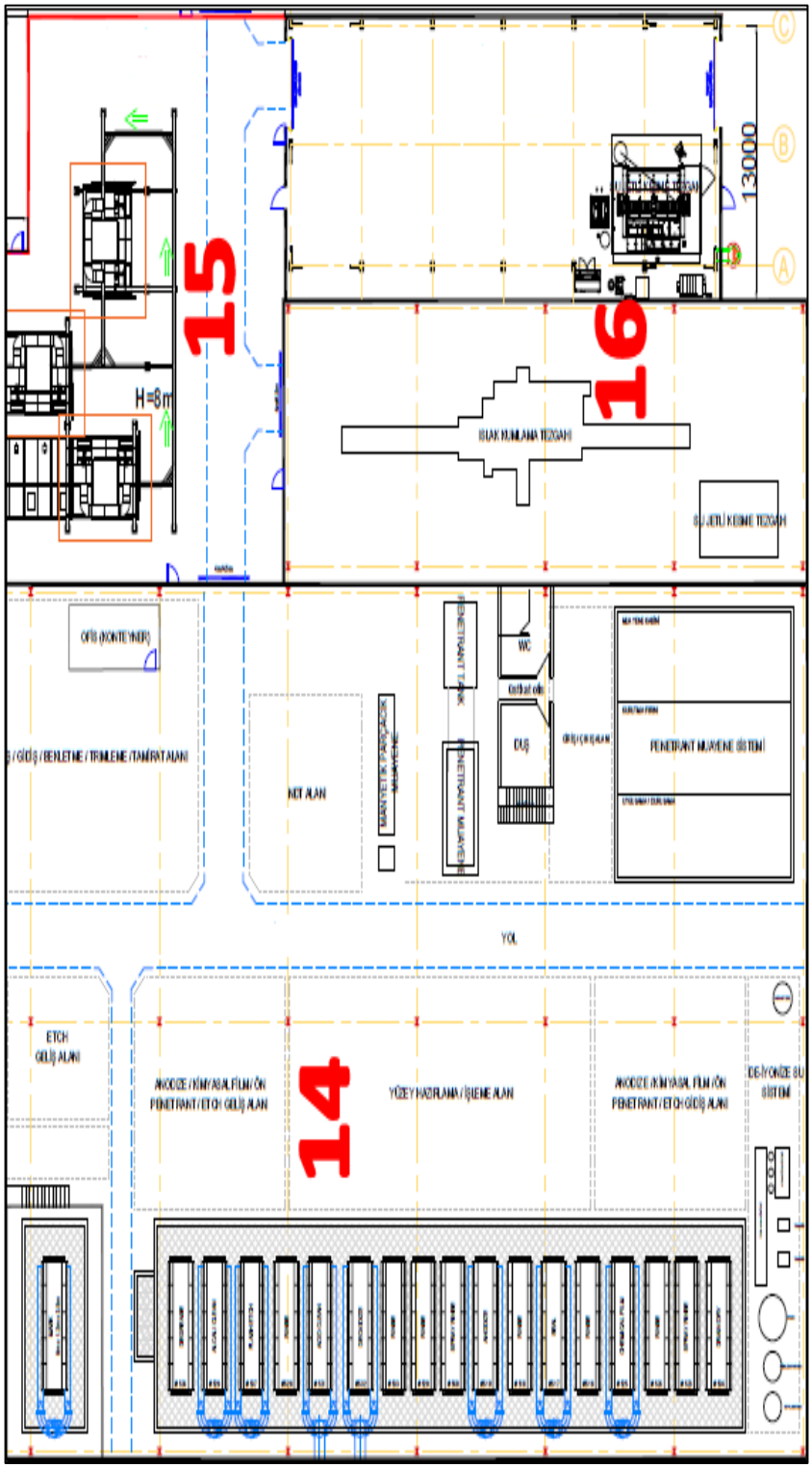


Figure 6.6. Surface operations in Building B20

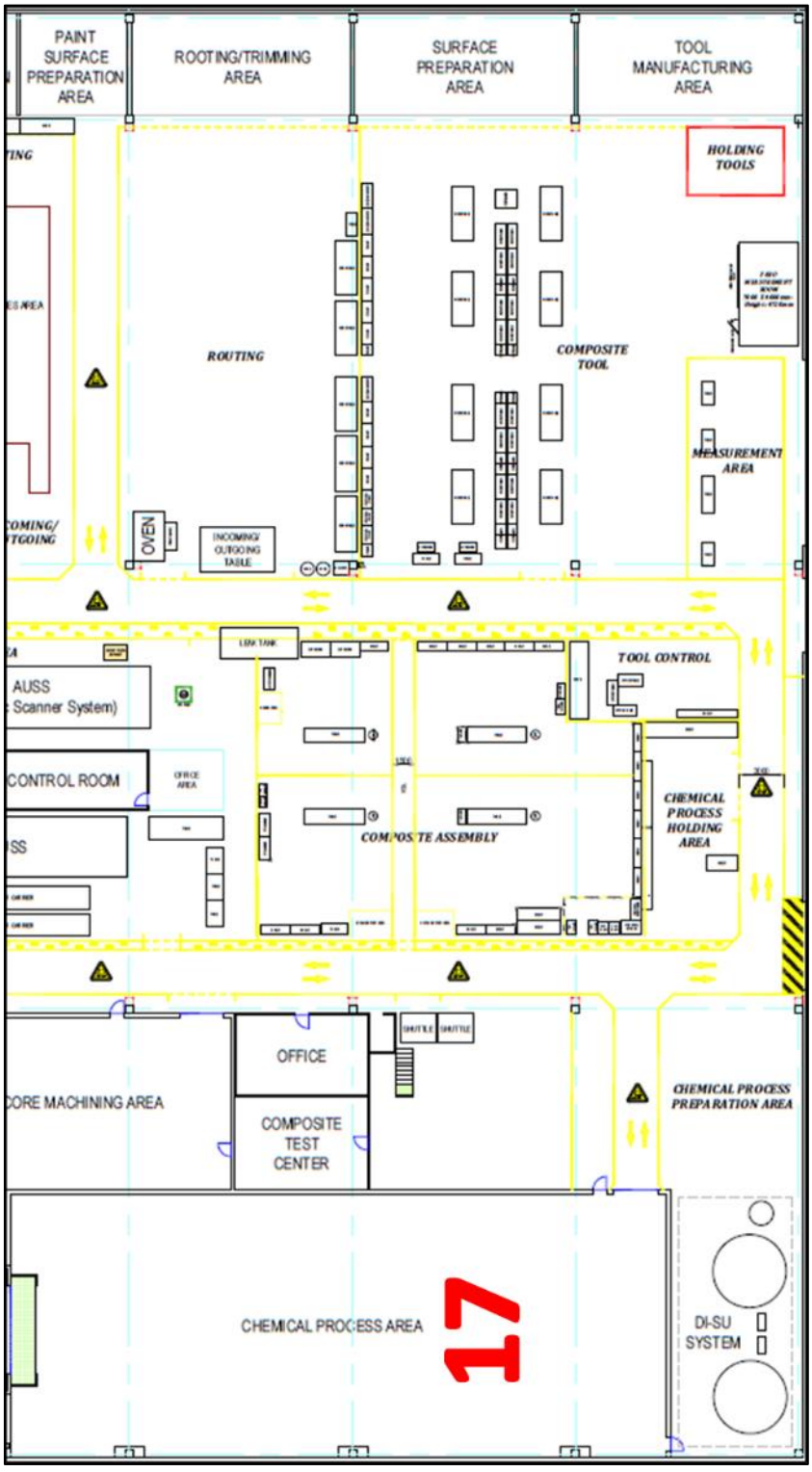


Figure 6.7. Surface operations in Building B220

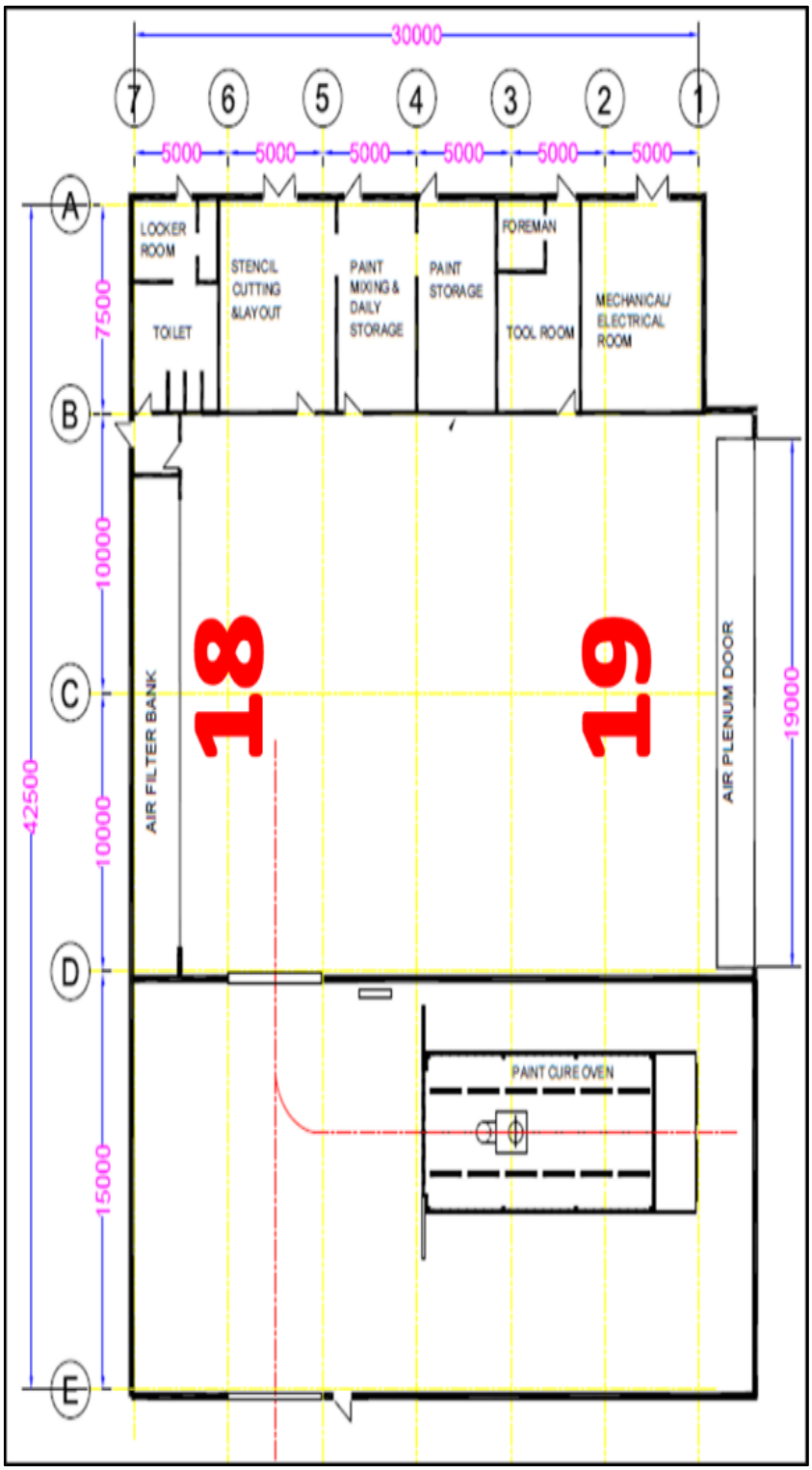


Figure 6.8. Painting process operations in Building B40

6.2.1. Data Analysis

The relevant data are collected with the help of Structural Query Language (SQL) from RDBMS. Related tables are connected to each other with the SQL queries in SAP Business Objects Crystal Reports 2008. Moreover, in order to retrieve the unit processing times of each panel in each stage for all occurrences, some functions are developed. Query codes with the syntax of Microsoft Object Linking and Embedding, DataBase (OLE DB) Provider for SQL Server and the functions with related attributes from the tables of RDBMS are compiled by Crystal Reports.

All of the outputs having overall 143,119 raw records are exported to MS Office Excel 2016 as a single spreadsheet for each panel type in a given stage with the average of processing times per quantity in a work order. Manufacturing stages 3, 4, 5, 7, 10, and 17 are metallurgical surface operations and they have standard processing times according to the specifications of the panels based on their manufacturing data packages. Similarly, machine failure data are obtained from JAVA-based Materials Requirement Planning (MRP) in ERP system.

All of the outputs having overall 1936 raw records are exported to MS Office Excel 2016 as spreadsheets for the stages 3, 4, 6, 7, 9, 10, 15, and 18. Upon collection of raw data related to unit processing times of panels for each stage and machine availability of each stage, data analysis and interpretation phase follow.

Two types of data, processing times and machine availability, are required to be analyzed due to their uncertainty. Since outliers heavily affect machine availability, in order to get realistic breakdown information from this data, we cleanse them through plotting. By this way, total of 1936 raw records are cleansed and reduced to 1143 meaningful data. In detail, machine availability data are reduced from 103 to 98 for stages 3, 7, and 10, from 101 to 73 for stage 4, from 500 to 276 for stage 9, from 118

to 107 for stage 15, and from 829 to 304 for stage 18. On the other hand, the number of machine availability data for stage 6 remain the same and equal 285.

Machine availability data are related to breakdown occurrences. Breakdown occurrences are interpreted by both durations and frequencies for the stages at the same time. By this way, necessary information about breakdowns is obtained for the DES model developed for the company's case scenario. The completion of data cleansing process is followed by fitting the data to one of the available distributions with the best goodness value in MATLAB R2018b.

After fitting the data to the best available distribution according to data's pattern observed on histogram, by using this distribution's parameter values, we calculate its mean to be determined as a processing time for each type of panel at each stage. This analysis is executed for the processing times of all panels at all stages of manufacturing.

A sample of data interpretations are accessible in Appendices (Appendix A). These data interpretations derived by the MATLAB function of Sheppard (2012) are used as the processing times of the panels at the manufacturing stages and the failures of parallel identical machines at the stages in the DES model's resource modules.

6.2.2. DES Model Design for an HFS Configuration

In order to start developing the DES model, first, the cycle time per panel is calculated based on Equation 6.1 according to the most demanded panel.

$$CT=\left\lfloor \frac{300\ \text{days}}{960\times 13\ \text{units}\times\text{types}} \times \frac{3\ \text{shifts}}{\text{days}} \times \frac{7.5\ \text{hrs}}{\text{shifts}} \times \frac{60\ \text{min}}{\text{hrs}} \right\rfloor = \frac{32\ \text{min}}{\text{units}\times\text{types}} \qquad (6.1)$$

Assembling the highest annual demand of fuselage which is 960 is supported by the resource capacity, i.e, 300 working days each with 3 shifts lasting 7.5 hours. Since the fuselage requires 13 types of panels, CT is calculated based on (960x13) panels/year. After that, CT in Equation 6.1 is rounded down to the largest integer to be introduced as an interval value having minutes as its unit of measure for the source module. Moreover, by being rounded down, CT value ensures that each type of panel can be manufactured during the cycle time.

For determining the number of parallel identical machines at each stage k, MoM_k is calculated based on Equation (6.2).

$$MoM_k = \left\lceil \frac{\sum_{j=1}^{n} p_{jk} \times D_j}{C_k} \right\rceil \qquad \forall k \in K \qquad (6.2)$$

D_j represents the demand of panel j per year. C_k identifies the yearly capacity of a machine at stage k. The example in Table 6.3 shows how MoM_k value is calculated for the cooling stage.

As it is observed in Table 6.1, in order to satisfy the yearly demand of panels, given the capacity of the cooling machine (refrigerator), the cooling stage requires at least 5 refrigerators.

Table 6.3. MoM_k calculation for the cooling stage

REFRIGERATOR					
panel, j	**p_{jk} (hrs)**	**D_j (per year)**	**$p_{jk} \times D_j$**	**C_k (300 days × 3 shifts × 7.5 hrs)**	**MoM_k**
1	0	960	0	6750	0
2	0	960	0	6750	0
3	0	960	0	6750	0

Comtinued on next page

Table 6.3. – *Continued from previous page*

REFRIGERATOR					
panel, j	**p_{jk} (hrs)**	**D_j (per year)**	**$p_{jk} \times D_j$**	**C_k (300 days × 3 shifts × 7.5 hrs)**	**MoM_k**
4	4	960	3840	6750	0.57
5	4	960	3840	6750	0.57
6	4	960	3840	6750	0.57
7	4	960	3840	6750	0.57
8	4	720	2880	6750	0.43
9	4	720	2880	6750	0.43
10	4	720	2880	6750	0.43
11	4	720	2880	6750	0.43
12	4	720	2880	6750	0.43
13	4	720	2880	6750	0.43
					4.84
				rounded up	**5**

Similarly, MoM_k calculations for the the other stages are accessible in Appendices (Appendix B). After a few runs with the DES model developed in Tecnomatix Plant Simulation 14, MoM_k values are updated iteratively with adjusted the D_j values and, as a result, the minimum number of parallel identical machines for each stage is determined in order to satisfy the highest annual demand. Table 6.4 lists the number of parallel identical machines for each stage as obtained from the DES model of the suggested HFS configuration for panel production in the company.

All of the required data are obtained now in order to illustrate the DES model physically.

DES model design starts with the creation of its source module shown in Figure 6.9. By changing “Mobile Unit (MU) selection” area from “Sequence Cyclical” to “Sequence”, the DES model is run for a single unit production of each panel.

Table 6.4. MoM_k values in the HFS

k	Stage	MoM_k
1	FIRST CUT	1
2	ROLL	1
3	CLEAN OPS1 (ALKALI CLEAN1 & VAPOR DEGREASE1)	1
4	HEAT TREATMENT	2
5	REFRIGERATOR	7
6	STRETCH PRESS	6
7	CLEAN OPS2 (ALKALI CLEAN2 & VAPOR DEGREASE2)	1
8	DEBURR1, DRILL HOLE & REMOVE TAB	3
9	MECHANIC MILL	7
10	VAPOR DEGREASE3	1
11	HAND FORM	1
12	DEBURR2 & HAND FINISH	6
13	CONDUCTIVITY, DIMENSIONAL & HARDNESS INSPECTION	2
14	PRE-PENETRANT ETCH OPS	1
15	NON-DESTRUCTIVE PENETRANT INSPECTION	2
16	MASK & WET BLAST & SURFACE INSPECTION OPS	1
17	TARTARIC SULFURIC ACID ANODIZE OPS	1
18	PAINT OPS	5
19	PAINT INSPECTION	1

This option is used for measuring the makespan values of different job (panel) sequences obtained by different scheduling methods. On the other hand, “Sequence Cyclical” option is only used for running the DES model based on the predetermined simulation length which is equal to 300 days according to CT calculation in order to determine the minimum number of parallel identical machines for each stage. Job sequence is altered via the panel type table shown in Figure 6.10. After creating the source module, stages are created as the resource modules, having determined the number of parallel identical machines at each stage. Each resource module represents the corresponding stage with an infinite buffer for WIP accumulation.

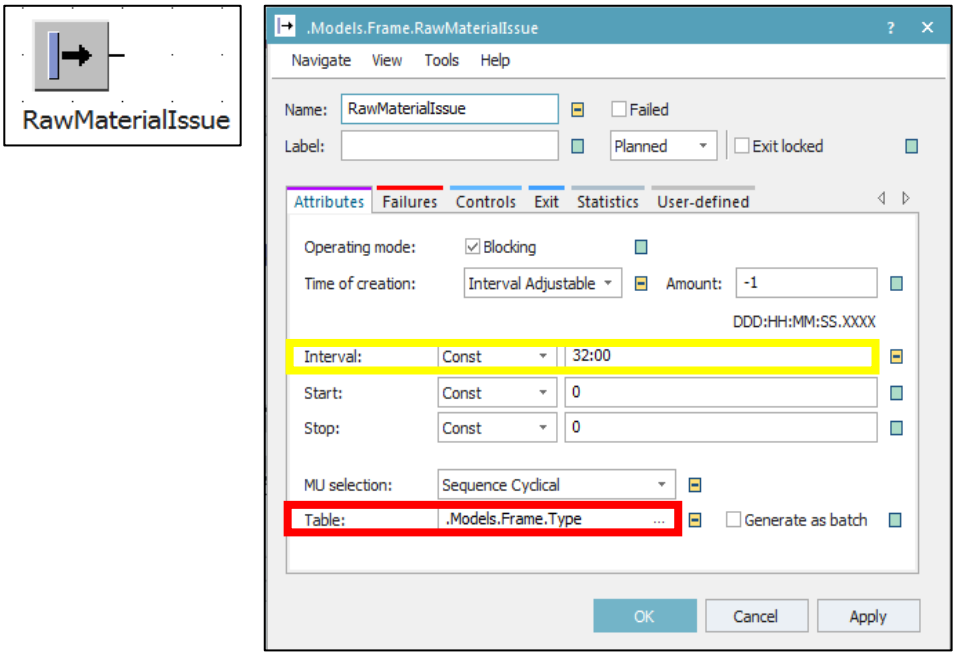
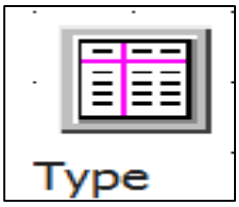


Figure 6.9. The source module of the DES model



	object 1	integer 2	string 3	table 4
string	MU	Number	Name	Attrib...
1	.MUs.Skin1	1	p1	
2	.MUs.Skin2	1	p2	
3	.MUs.Skin3	1	p3	
4	.MUs.Skin4	1	p4	
5	.MUs.Skin5	1	p5	
6	.MUs.Skin6	1	p6	
7	.MUs.Skin7	1	p7	
8	.MUs.Skin8	1	p8	
9	.MUs.Skin9	1	p9	
10	.MUs.Skin10	1	p10	
11	.MUs.Skin11	1	p11	
12	.MUs.Skin12	1	p12	
13	.MUs.Skin13	1	p13	

Figure 6.10. Initial sequence table of panel (skin) type inserted to source module

Figures 6.11, 6.12, 6.13, 6.14, 6.15, 6.16, and 6.17 are used to illustrate one of the resource modules, heat treatment stage, of the DES model with its infinite (capacity=500 panels) buffer (backlog of jobs).

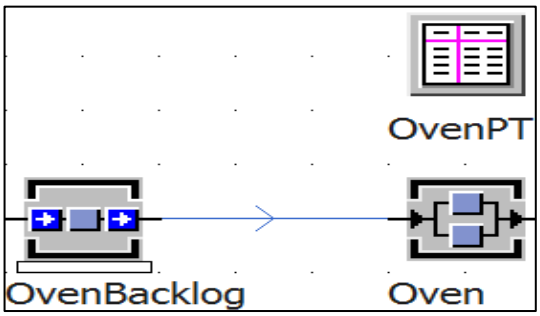


Figure 6.11. The resource module structure of heat treatment stage

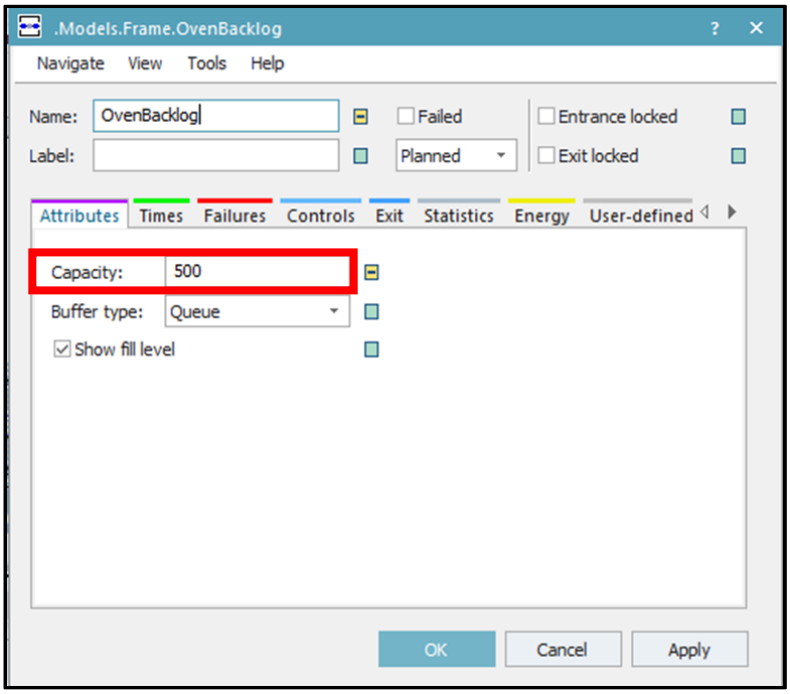


Figure 6.12. Heat treatment stage backlog

	string 1	time 2
string	MU type	Time
1	p1	0.0000
2	p2	0.0000
3	p3	0.0000
4	p4	1:00:00.0000
5	p5	1:00:00.0000
6	p6	1:00:00.0000
7	p7	1:00:00.0000
8	p8	1:00:00.0000
9	p9	1:00:00.0000
10	p10	1:00:00.0000
11	p11	1:00:00.0000
12	p12	1:00:00.0000
13	p13	1:00:00.0000

Figure 6.13. Heat treatment stage processing times (hrs) table of panels

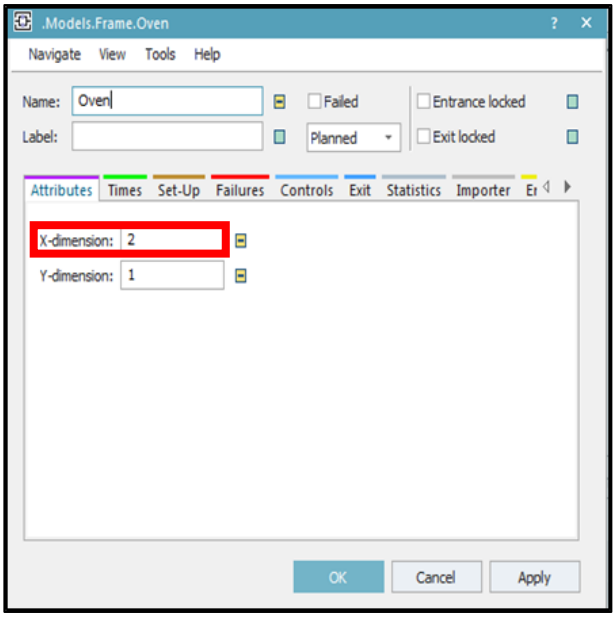


Figure 6.14. Heat treatment stage with 2 parallel identical ovens

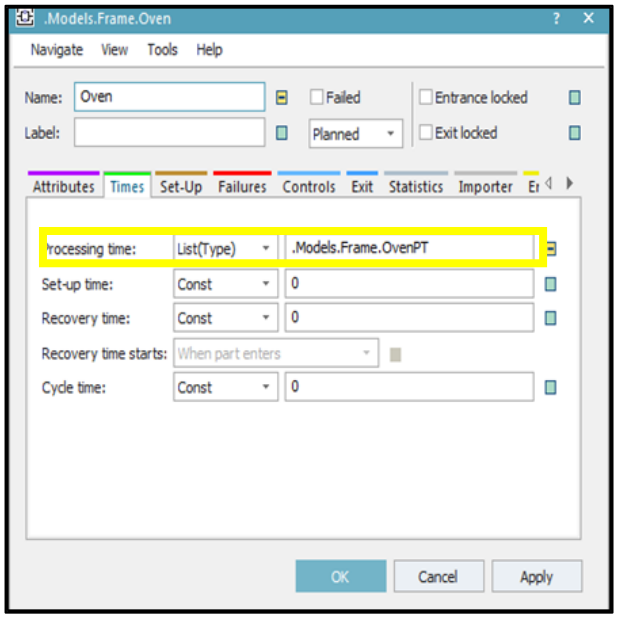


Figure 6.15. Heat treatment stage with the processing times table of panels

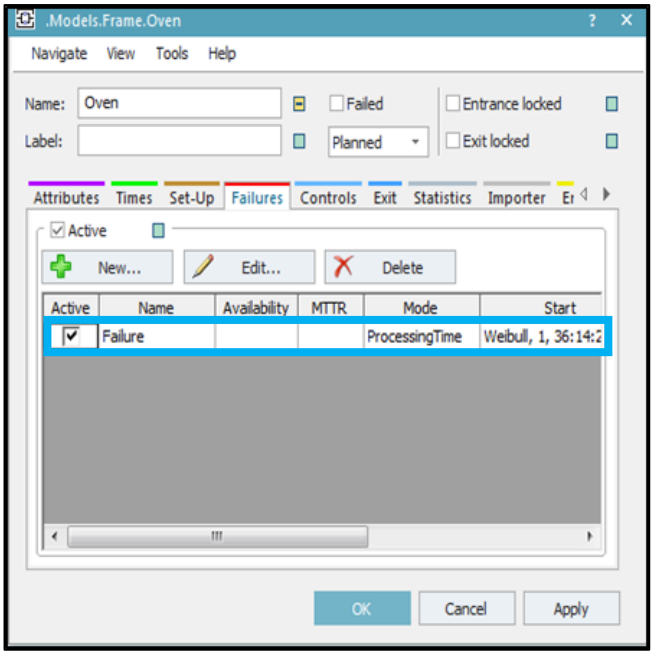


Figure 6.16. Heat treatment stage with failure distribution

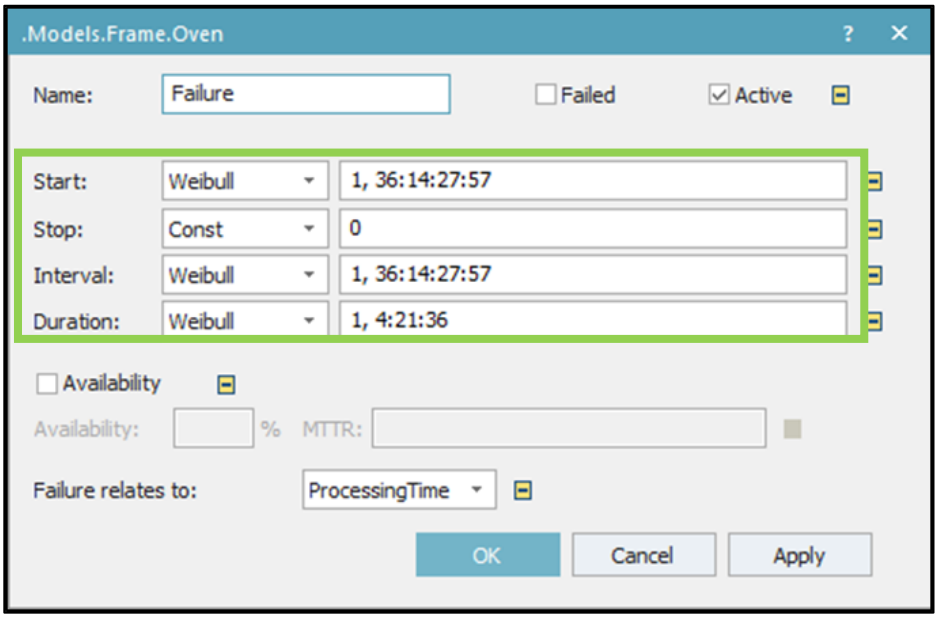


Figure 6.17. Heat treatment stage with failure frequency and duration

Some of the stages are skipped by some of the panels (Figure 6.13). Processing times of all panels at the stages are listed in Appendices (Appendix C).

After calculating the mean values of processing times of the panels for each stage based on the fit distribution, the mean processing times are inserted to the resource modules with panel type tables rather than the stochastic counterparts, since the processing times are assumed as deterministic in the DES model. However, fitting them to the best available distributions is only used for data analysis and interpretation phase in order to cleanse the data collected via SQL queries from RDBMS to obtain meaningful values from them based on the mean values of the best available distributions.

In the DES model designed with Tecnomatix Plant Simulation 14, Weibull distribution is exponentiated to obtain exponential distribution via Equation (6.3).

$$f(x)=\frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} \qquad x\geq 0 \text{ is the pdf of Weibull distribution} \qquad (6.3)$$

When k=1 in the pdf of the Weibull Distribution $f(x)=\lambda^{-1}e^{-x\lambda^{-1}}$ is the pdf of the exponential distribution. The failure of the resource occurs in Tecnomatix Plant Simulation 14 according to the Figure 6.18 for given start (a resource begins operating), duration (a failure occurs), and interval (a resource begins operating again) values based on the study by Bangsow (2015).

Figure 6.18. Failure settings of a resource in Tecnomatix Plant Simulation 14

The physical appearance of the DES model designed with Tecnomatix Plant Simulation 14 is shown in Figures 6.19 and 6.20.

First of all, event controller shown in Figure 6.21 determines the simulation speed and length. Simulation length is set as 300 days. Moreover, it generates a report which summarizes the simulation after a run is completed. The summary report of a DES model run is shown in Figure 6.22. The summary report gives statistics like which panel is drained by which drain module, mean life time of each panel, throughput of each panel, utilization percentages of production, transport and storage to derive value-added percentage with its portion bar. Summary report is used for checking the throughput for each panel at the end of simulation run in order to determine whether the simulated HFS configuration has sufficient capacity to satisfy panels' annual demand or not. The reason why value-added is low is that we try to find the minimum number of parallel identical machines utilized at each stage by providing infinite buffers between stages in order to satisfy panels' demand per year.

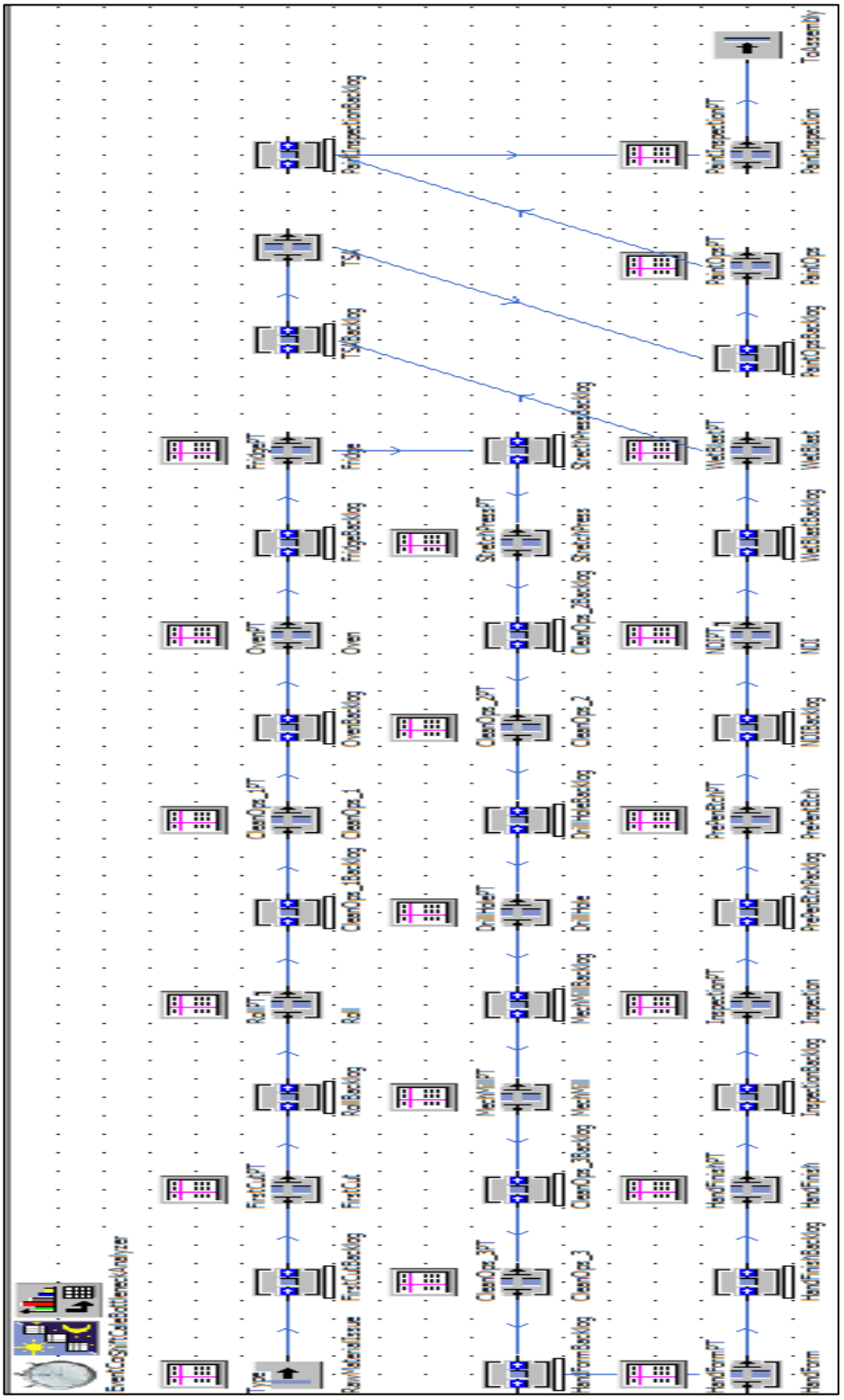


Figure 6.19. The DES model in Tecnomatix Plant Simulation 14

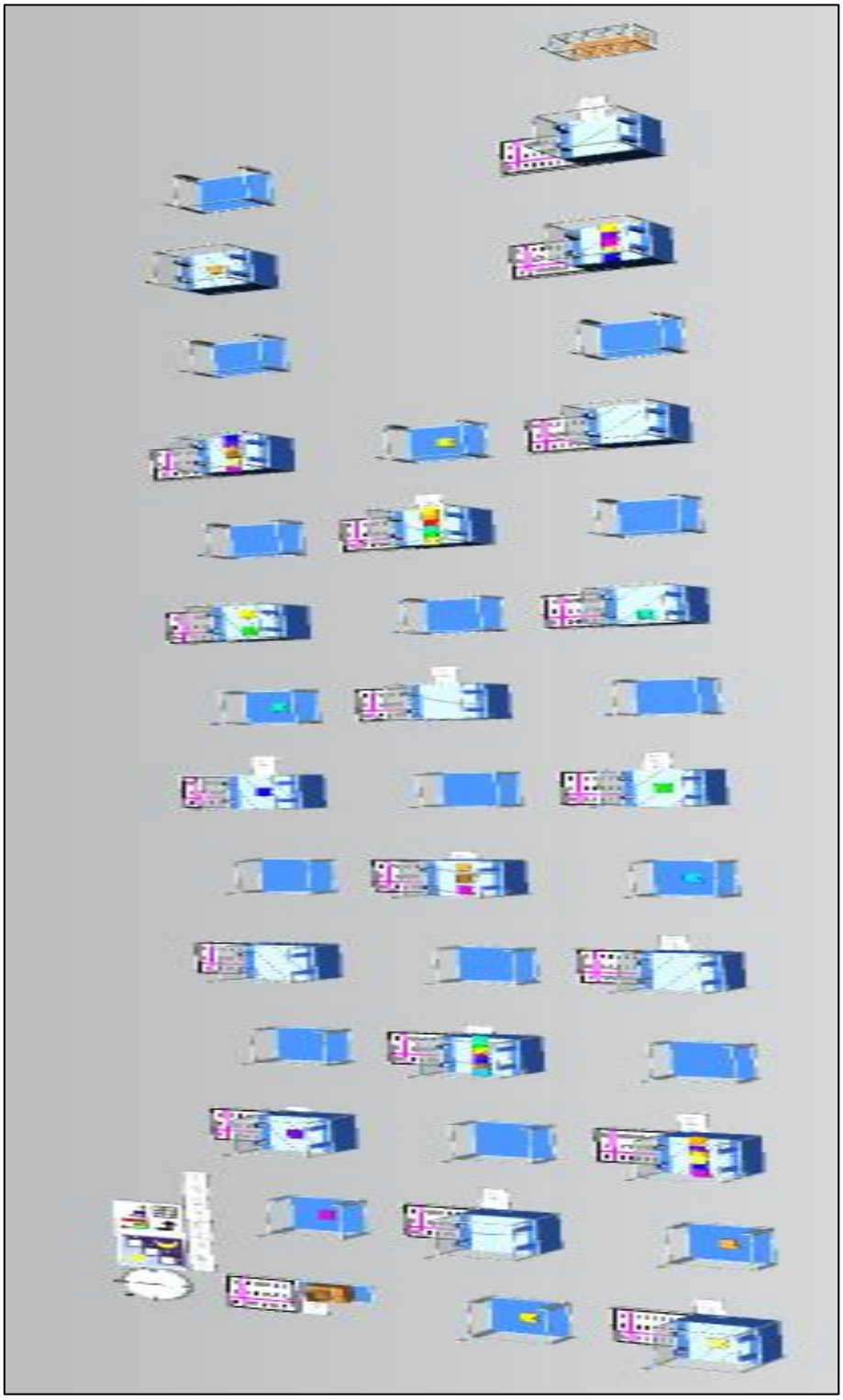


Figure 6.20. 3D version of the DES model in Tecnomatix Plant Simulation 14

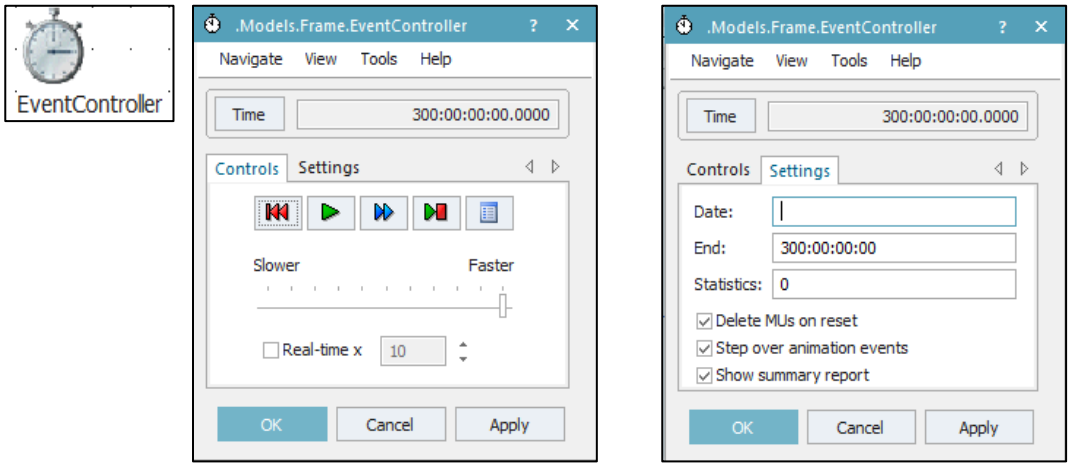


Figure 6.21. Event controller as one of the simulation settings

.Models.Frame

Simulation time: 300:00:00:00.0000

Cumulated Statistics of the Parts which the Drain Deleted

Object	Name	Mean Life Time	Throughput	TPH	Production	Transport	Storage	Value added	Portion
ToAssembly	p1	20:04:13:37.8458	962	0	2.42%	0.00%	97.58%	2.13%	
ToAssembly	p10	20:17:37:51.2153	960	0	5.73%	0.00%	94.27%	4.96%	
ToAssembly	p11	20:21:17:06.1238	959	0	6.44%	0.00%	93.56%	5.58%	
ToAssembly	p12	20:17:38:13.5406	960	0	5.77%	0.00%	94.23%	4.99%	
ToAssembly	p13	20:18:22:24.0929	959	0	5.94%	0.00%	94.06%	5.21%	
ToAssembly	p2	20:02:56:31.0368	962	0	2.33%	0.00%	97.67%	2.05%	
ToAssembly	p3	20:02:18:09.0369	962	0	1.84%	0.00%	98.16%	1.55%	
ToAssembly	p4	20:12:28:26.1015	961	0	4.77%	0.00%	95.23%	4.13%	
ToAssembly	p5	20:14:44:47.3187	961	0	4.87%	0.00%	95.13%	4.27%	
ToAssembly	p6	20:15:04:12.9723	961	0	4.64%	0.00%	95.36%	4.10%	
ToAssembly	p7	20:14:28:26.7778	960	0	4.79%	0.00%	95.21%	4.18%	
ToAssembly	p8	20:18:15:04.0043	960	0	5.42%	0.00%	94.58%	4.72%	
ToAssembly	p9	20:18:23:46.6526	960	0	5.75%	0.00%	94.25%	4.97%	

Figure 6.22. Summary report generated at the end of a simulation run

Therefore, storage has the highest utilization percentage due to the fact that buffers are highly utilized at all times during a simulation run. Another reason why value-added is low is that the processing times among stages vary drastically. While some of the operations last for hours, the others last for minutes, resulting in unbalances among the stages causing bulky WIP accumulation.

Secondly, shift calendar determines the number of shifts with hours for working days required. In the simulation run, there are 3 shifts, each of them with 7.5 hours in total 300 day/year. The frame of shift calendar is shown in Figure 6.23.

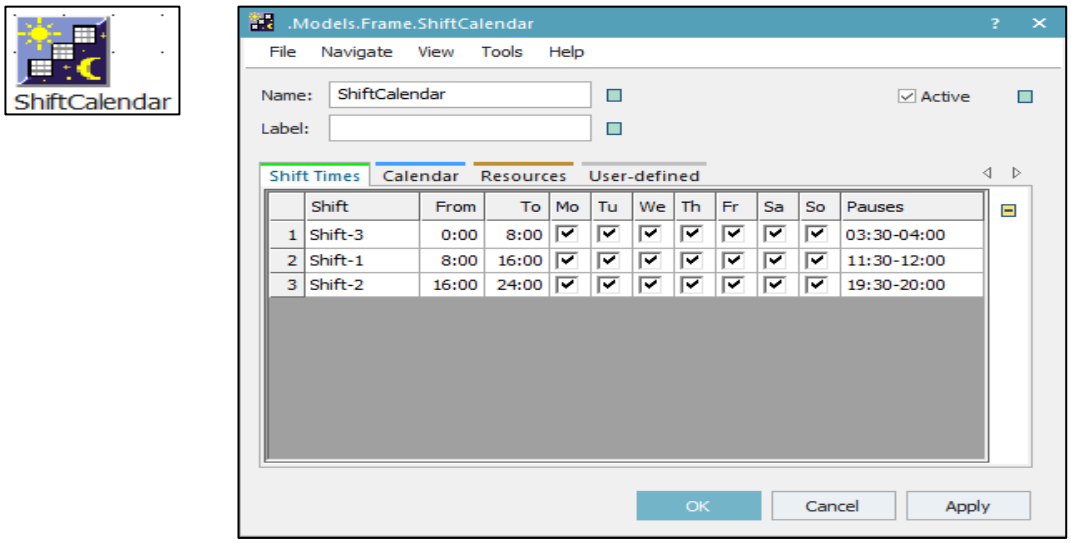
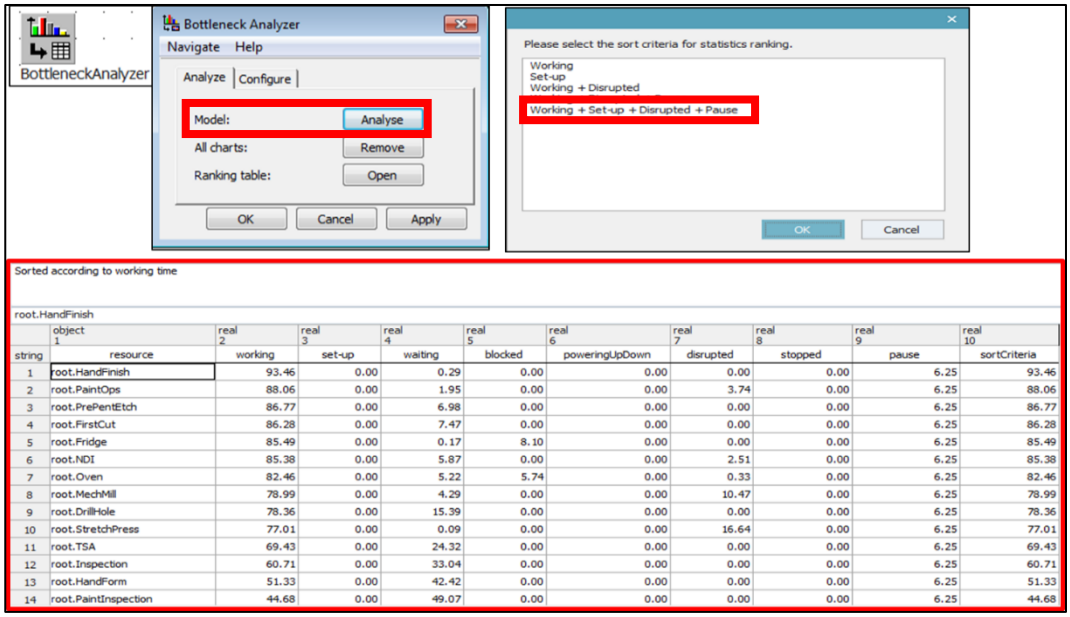


Figure 6.23. The frame of shift calendar

Finally, bottleneck analyzer shows the utilizations of the resource modules. It helps us to identify the bottleneck resources so that we can increase the number of identical parallel machines at a particular bottleneck stage. This tool and MoM_k values support each other by feedbacking one another iteratively based on the simulation runs results and at the end of the iterations, the minimum number of parallel identical machines, required for each stage, is determined. Figure 6.24 shows the bottleneck analyzer with its outcome. In order to determine the exact MoM_k values, the utilizations of the stages obtained from bottleneck analyzer's outcome shown on Figure 6.24 are used with the precalculated MoM_k values. For example, the stage with the highest utilization is the candidate whose number of parallel identical machines is increased by one based on simulation the runs results and its precalculated MoM_k value. Bottleneck analyzer also helps to configure its outcome with its frame for different options as shown in Figure 6.25.



string	object 1 resource	real 2 working	real 3 set-up	real 4 waiting	real 5 blocked	real 6 poweringUpDown	real 7 disrupted	real 8 stopped	real 9 pause	real 10 sortCriteria
1	root.HandFinish	93.46	0.00	0.29	0.00	0.00	0.00	0.00	6.25	93.46
2	root.PaintOps	88.06	0.00	1.95	0.00	0.00	3.74	0.00	6.25	88.06
3	root.PrePentEtch	86.77	0.00	6.98	0.00	0.00	0.00	0.00	6.25	86.77
4	root.FirstCut	86.28	0.00	7.47	0.00	0.00	0.00	0.00	6.25	86.28
5	root.Fridge	85.49	0.00	0.17	8.10	0.00	0.00	0.00	6.25	85.49
6	root.NDI	85.38	0.00	5.87	0.00	0.00	2.51	0.00	6.25	85.38
7	root.Oven	82.46	0.00	5.22	5.74	0.00	0.33	0.00	6.25	82.46
8	root.MechMill	78.99	0.00	4.29	0.00	0.00	10.47	0.00	6.25	78.99
9	root.DrillHole	78.36	0.00	15.39	0.00	0.00	0.00	0.00	6.25	78.36
10	root.StretchPress	77.01	0.00	0.09	0.00	0.00	16.64	0.00	6.25	77.01
11	root.TSA	69.43	0.00	24.32	0.00	0.00	0.00	0.00	6.25	69.43
12	root.Inspection	60.71	0.00	33.04	0.00	0.00	0.00	0.00	6.25	60.71
13	root.HandForm	51.33	0.00	42.42	0.00	0.00	0.00	0.00	6.25	51.33
14	root.PaintInspection	44.68	0.00	49.07	0.00	0.00	0.00	0.00	6.25	44.68

Figure 6.24. The frame of bottleneck analyzer

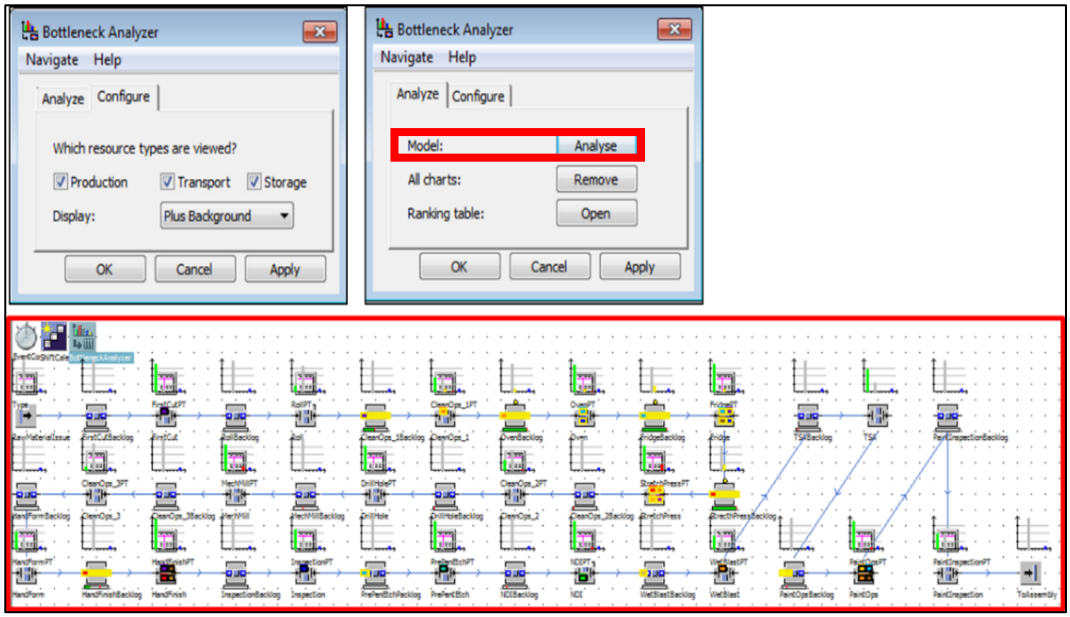
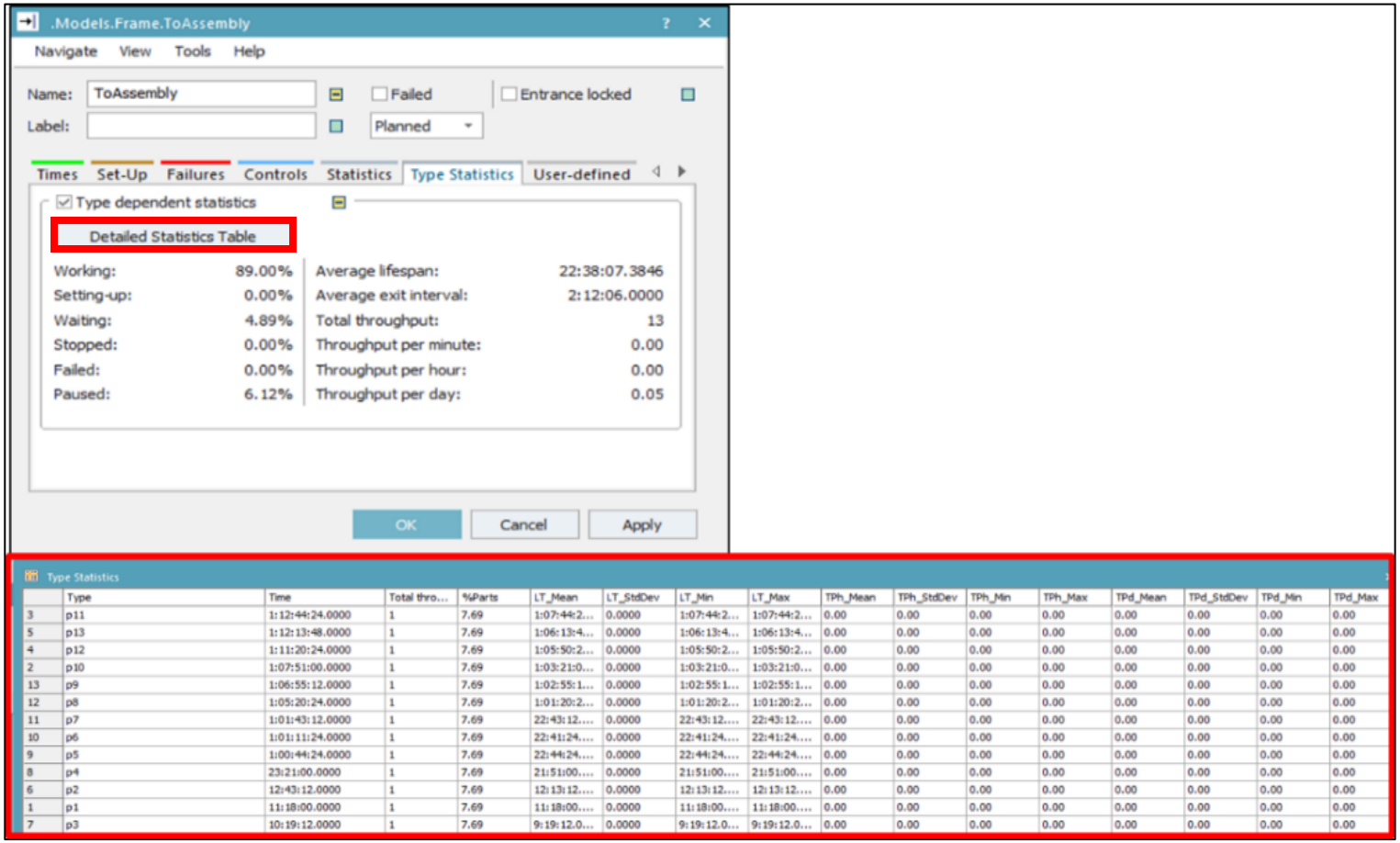


Figure 6.25. The frame of bottleneck analyzer with its outcomes displayed on the complete DES model

Another useful tool is the drain module which helps to measure the effectiveness of the HFS scheduling methods the sequences of which are inserted to DES model source module. Drain module provides detailed statistics table for each type of panel. Figure 6.26 shows the drain module frame with its statistics table.

As it is seen in Figure 6.26, with time attribute, the makespan for a given sequnce is obtained by sorting this attribute in descending order at the end of the simulation run. Therefore, at the end of simulation run, in time attribute, panel with the highest completion time value determines the makespan for a sequence inserted to DES model source module.



	Type	Time	Total thro...	%Parts	LT_Mean	LT_StdDev	LT_Min	LT_Max	TPh_Mean	TPh_StdDev	TPh_Min	TPh_Max	TPd_Mean	TPd_StdDev	TPd_Min	TPd_Max
3	p11	1:12:44:24.0000	1	7.69	1:07:44:2...	0.0000	1:07:44:2...	1:07:44:2...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	p13	1:12:13:48.0000	1	7.69	1:06:13:4...	0.0000	1:06:13:4...	1:06:13:4...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	p12	1:11:20:24.0000	1	7.69	1:05:50:2...	0.0000	1:05:50:2...	1:05:50:2...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	p10	1:07:51:00.0000	1	7.69	1:03:21:0...	0.0000	1:03:21:0...	1:03:21:0...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
13	p9	1:06:55:12.0000	1	7.69	1:02:55:1...	0.0000	1:02:55:1...	1:02:55:1...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12	p8	1:05:20:24.0000	1	7.69	1:01:20:2...	0.0000	1:01:20:2...	1:01:20:2...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11	p7	1:01:43:12.0000	1	7.69	22:43:12....	0.0000	22:43:12....	22:43:12....	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	p6	1:01:11:24.0000	1	7.69	22:41:24....	0.0000	22:41:24....	22:41:24....	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	p5	1:00:44:24.0000	1	7.69	22:44:24....	0.0000	22:44:24....	22:44:24....	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	p4	23:21:00.0000	1	7.69	21:51:00....	0.0000	21:51:00....	21:51:00....	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	p2	12:43:12.0000	1	7.69	12:13:12....	0.0000	12:13:12....	12:13:12....	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1	p1	11:18:00.0000	1	7.69	11:18:00....	0.0000	11:18:00....	11:18:00....	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	p3	10:19:12.0000	1	7.69	9:19:12.0...	0.0000	9:19:12.0...	9:19:12.0...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 6.26. The drain module

6.3. Computational Study

So far, data retrieval from RDBMS, processing the retrieved data and designing the DES model based on these data are covered. Having obtained an HFS configuration for the panel productşon for the fuselage under consideration, scheduling the HFS becomes the issue. Therefore, this section covers all aspects of the computational study on scheduling the HFS which is supported by the DES model in detail. For this purpose, several different software packages are used for different solution methods. MILP and CP models are coded via Optimization Programming Language (OPL) in IBM ILOG CPLEX 12.6. Dispatching rules are applied with LEKIN 3.3 and MATLAB R2018b. The renowned heuristic algorithms are implemented with both MS Excel 2019 and MATLAB R2018b. The proposed HA heuristic and GSO metaheuristic methods are coded in MATLAB R2018b. Moreover, LEKIN outputs and Excel spreadsheets of dispatching rules are provided also in Appendices (Appendix D). In this phase, the results of the computational studies with the case HFS scheduling problem are shared.

The HFS scheduling problem in the case study consists of 13 jobs and 19 stages some of which have more than one identical machine in parallel. Table 6.5 shows that the makespan values obtained from the scheduling methods some of whose job sequences are also inserted to the DES model in order to observe the job sequences under the condition of machine breakdowns.

Since the scheduling methods with the symbol * do not use the FAM and ECT strategies, for the first stage, the job sequences obtained from them are not inserted to the DES model source module. Therefore, there are no related C_{max} values for them at the end of a simulation run in the DES model also using the FAM and ECT strategies.

Table 6.5. The results of the scheduling methods for the case study

	Current Solution Methods										**Proposed Solution Methods**		
	CPLEX	**LEKIN**		**FAM and ECT**						**BH***	**CP Optimizer**	**FAM and ECT**	
	MILP*	**SPT***	**LPT***	**SIRO**	**STPT**	**LTPT**	**Palmer**	**CDS**	**NEH**		**CP***	**HA**	**GSO**
C_{max} (hrs)	29.18	32.87	33.13	31.78	33.03	32.59	32.53	31.08	29.76	32.53	28.98	29.29	29.34
CPU (sec)	8485.21	1	1	0.168	0.199	0.171	0.199	0.169	0.185	0.258	51.63	303.26	601.74
C_{max} (hrs) DES	-	-	-	34.96	37.84	37.67	36.74	34.59	33.32	-	-	32.29	32.42

MILP model is solved in 8485.21 seconds via CPLEX to near-optimality where relative MILP gap is equal to 1.2 % (a stopping criterion to obtain the best feasible solution), since the HFS scheduling problem is NP-hard in the strong sense (Table 6.6). This means that the obtained solution is very close to the optimal. On the other hand, the CP model solves the problem to optimality within a very short computational time (Table 6.7). Both MILP and CP models are measured in terms of C_{max} and CPU times. Therefore, the roles of the other solution methods have vital importance to determine the job sequence in order to obtain the best makespan value.

Table 6.6. The completion times with the MILP model

Job, *j*	c_{j19}
1	24.00
2	20.90
3	20.67

Continued on next page

Table 6.6 – *Continued from previous stage*

Job, *j*	c_{j19}
4	24.40
5	28.80
6	24.20
7	22.84
8	25.96
9	29.18
10	27.58
11	28.30
12	28.58
13	27.97

Table 6.7. The start and end times of jobs with the CP model

Job, *j*	**Stage, *k***	**Start**	**End**
1	19	23.72	23.93
2	19	23.18	23.42
3	19	27.63	27.78
4	19	28.12	28.32
5	19	22.23	22.47
6	19	27.03	27.23
7	19	20.52	20.70
8	19	25.13	25.38
9	19	28.60	28.98
10	19	26.68	27.03
11	19	27.78	28.12
12	19	28.32	28.60
13	19	27.23	27.63

The job sequences of the dispatching rules (SPT & LPT) are obtained by LEKIN 3.3 with ease and their C_{max} values are recorded. The results of the other algorithms SIRO, STPT, LTPT, CDS, Palmer and NEH are derived via MS Office Excel 2019 spreadsheets and their job sequences plugged into FAM and ECT strategies via MATLAB R2018b. The job sequences of CDS, Palmer and NEH are also presented

in Appendices (Appendix E). Moreover, in order to implement the NEH algorithm, DES model is reduced to the traditional flow shop for getting makespan values of partial job sequences in NEH algorithm. C_{max} values of the renowned heuristic algorithms are also recorded. Furthermore, BH is coded via MATLAB R2018b and its C_{max} value is also recorded with CPU time. According to this algorithm, the bottleneck stage is stage 12 and whole scheduling process is executed according to this stage based on the rules of BH as explained in Chapter 4.

Consequently, CP provides the optimal solution. The second-best solution method is MILP model. However, to supply this good solution, MILP model spends a remarkable amount of time. The third best and the fourth best solution methods are our proposed approaches which are HA and GSO. HA is set to a million iteration to yield this result. On the other hand, the setup of GSO has already been explained in detail in Chapter 5. Both of them provide promising results in terms of both effectiveness and efficiency with a good balance between solution quality and time. All solution methods are run on the computer with Intel® Core™ i5-8265U CPU @ 1.60 GHz 1.80 GHz and 7.82 / 8.00 GB RAM with 64-bit operating system.

According to the simulation runs results for the HFS scheduling problem represented as ($HFc, Pm|skip, unavail(brkdwn)|C_{max}$) configuration, the proposed HA and GSO yield the best results among other solution methods using FAM and ECT strategies, which is parallel to their solutions in ($HFc, Pm|skip|C_{max}$) configuration.

CHAPTER 7

COMPUTATIONAL STUDY

In order to assess the performance of the proposed CP, HA, and GSO, we use the test problems of Carlier and Neron (2001) and we compare the proposed algorithms against PSO of Liao et al. (2012), QIA of Niu et al. (2009), AIS of Engin and Doyen (2004), GA of Besbes et al. (2006), ACO and Ant Colony System (ACS) of Khalouli et al. (2009), and B&B of Carlier and Neron (2001).

The test problems vary from 10 jobs and 5 stages to 15 jobs and 10 stages. Processing times of the jobs are uniformly distributed between 3 and 20. The notation used for problem description is defined below through an example problem, that is, *j*10*c*5*a*2.

- *j*10: 10 jobs.
- *c*5: 5 stages.
- *a*: number of parallel identical machines at the stages.
- 2: index of a problem instance.

The number of parallel identical machines at the stages varies according to the "letter" before the index of an instance for a given problem:

- *a*: there is a single machine in the middle stage and there are three machines at other stages.

- *b*: there is a single machine at the first stage and there are three machines at the other stages.

- *c*: there are two machines at the middle stage and there are three machines at the other stages.

- *d*: there are three machines at all stages.

We solve 76 test problems and report the C_{max} values together with CPU times in Table 7.1. Instance 60 is discarded from the test runs because it has a structural fault. However, the indices of instances remain the same for comparison purposes.

For the algorithms with the symbol *, only the makespan values of them are reported in their papers. Furthermore, the problem instances in bold represent harder problems. The letter *a* in "CPU" column means that the solution of the instance could not be reached within 1600 seconds. The letter *b* in "CPU" column means that B&B could not reach the optimal solution within 1600 seconds. The letter *c* in "CPU" column means that the solution, which B&B reaches, is not optimal, and also, this solution is not reached within 1600 seconds.

Since *j*10*c*5*a** type problems are easy to solve, all algorithms yield the optimal solution in a short time. HA is slightly better than the other two proposed solution methods in terms of only solution time for all instances. However, solution times of the proposed algorithms are acceptable, since they are all below 1600 seconds.

Like *j*10*c*5*a**, *j*10*c*5*b** type problems are also easy to solve. Therefore, all algortihms yield the optimal solutions for all problems. The difference occurs in solution times on average, again, and PSO is slightly better than the other algorithms.

Table 7.1. Results of test problems

		PSO		QIA*	AIS		GA		ACO*	ACS*	B&B		CP		HA		GSO	
No	**Instance**	C_{max}	**CPU**	C_{max}	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	C_{max}	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	**CPU**
1	*j*10*c*5*a*2	88	0.00	88	88	0.05	88	0.23	88	88	88	13	88	2.35	88	0.10	88	242.99
2	*j*10*c*5*a*3	117	0.00	117	117	0.14	117	0.04	117	117	117	7	117	1.66	117	0.11	117	242.76
3	*j*10*c*5*a*4	121	0.00	121	121	0.17	121	0.12	121	121	121	6	121	1.46	121	0.11	121	245.99
4	*j*10*c*5*a*5	122	0.01	122	122	0.91	122	0.18	122	122	122	11	122	1.10	122	0.16	122	243.47
5	*j*10*c*5*a*6	110	0.71	110	110	3.58	110	0.25	110	110	110	6	110	1.17	110	0.21	110	245.04
	Average	111.60	0.15	111.60	111.60	0.97	111.60	0.16	111.60	111.60	111.60	8.60	111.60	1.55	111.60	0.14	111.60	244.05
6	*j*10*c*5*b*1	130	0.00	130	130	0.14	130	0.03	130	130	130	13	130	1.02	130	0.18	130	244.94
7	*j*10*c*5*b*2	107	0.00	107	107	0.80	107	0.05	107	107	107	6	107	1.16	107	0.18	107	255.93
8	*j*10*c*5*b*3	109	0.01	109	109	0.52	109	0.51	109	109	109	9	109	1.15	109	0.19	109	244.64
9	*j*10*c*5*b*4	122	0.25	122	122	1.66	122	0.34	122	122	122	6	122	1.14	122	0.16	122	249.21
10	*j*10*c*5*b*5	153	0.00	153	153	0.12	153	0.28	153	153	153	6	153	1.07	153	0.16	153	250.49
11	*j*10*c*5*b*6	115	0.00	115	115	0.14	115	0.02	115	115	115	11	115	1.09	115	0.17	115	247.08
	Average	122.67	0.05	122.67	122.67	0.56	122.67	0.21	122.67	122.67	122.67	8.50	122.67	1.11	122.67	0.17	122.67	248.71
12	***j*10*c*5*c*1**	68	0.33	69	68	31.62	68	0.38	68	68	68	28	68	1.60	68	5.83	68	257.46
13	***j*10*c*5*c*2**	74	0.54	76	74	4.14	74	4.41	75	75	74	19	74	2.29	74	6.62	75	247.59
14	***j*10*c*5*c*3**	71	37.00	74	72	a	72	a	72	72	71	240	71	1.68	72	1.55	72	236.13
15	***j*10*c*5*c*4**	66	0.22	75	66	2.64	66	1.35	66	66	66	1017	66	2.77	66	21.06	67	253..873
16	***j*10*c*5*c*5**	78	0.12	79	78	13.83	78	0.57	78	78	78	42	78	1.69	78	6.50	79	252.89
17	***j*10*c*5*c*6**	69	0.41	72	69	12.14	69	0.84	69	69	69	4865 (b)	69	2.88	69	9.97	70	242.85
	Average	71	6.43	74.17	71.17	a	71.17	a	71.33	71.33	71	1035.17	71	2.15	71.17	8.59	71.83	247.38

Continued on next page

Table 7.1. – *Continued from previous page*

		PSO		QIA*	AIS		GA		ACO*	ACS*	B&B		CP		HA		GSO	
No	**Instance**	C_{max}	**CPU**	C_{max}	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	C_{max}	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	**CPU**
18	***j*10*c*5*d*1**	66	0.19	69	66	4.89	66	0.18	66	66	66	6490 (b)	66	1.94	66	4.24	66	239.56
19	***j*10*c*5*d*2**	73	1.16	76	73	31.16	74	a	75	73	73	2617 (b)	73	2.17	74	2.06	74	252.96
20	***j*10*c*5*d*3**	64	0.10	68	64	14.78	64	0.18	64	64	64	481	64	3.51	64	7.25	64	279.47
21	***j*10*c*5*d*4**	70	0.34	75	70	4.50	70	0.19	70	70	70	393	70	2.37	70	23.90	70	261.72
22	***j*10*c*5*d*5**	66	0.52	71	66	1445.64	66	3.41	66	66	66	1627 (b)	66	2.52	67	12.53	67	259.14
23	***j*10*c*5*d*6**	62	0.38	64	62	7.61	62	0.48	62	62	62	6861 (b)	62	2.04	62	5.81	62	240.97
	Average	66.83	0.45	70.50	66.83	251.43	67	0.89	67.17	66.83	66.83	3078.17 (b)	66.83	2.43	67.17	9.30	67.17	255.64
24	*j*10*c*10*a*1	139	0.06	139	139	1.19	139	0.23	139	139	139	41	139	1.31	139	0.98	139	289.62
25	*j*10*c*10*a*2	158	0.87	158	158	17.78	158	0.53	158	158	158	21	158	1.22	158	32.13	158	283.78
26	*j*10*c*10*a*3	148	0.02	148	148	0.53	148	0.39	148	148	148	58	148	1.18	148	0.66	148	283.09
27	*j*10*c*10*a*4	149	0.09	149	149	1.77	149	0.32	149	149	149	21	149	1.20	149	0.30	149	292.62
28	*j*10*c*10*a*5	148	0.10	148	148	0.56	148	0.19	148	148	148	36	148	2.34	148	0.19	148	286.35
29	*j*10*c*10*a*6	146	0.24	146	146	3.56	146	0.57	146	146	146	20	146	1.86	146	8.06	146	286.76
	Average	148	0.23	148	148	4.23	148	0.37	148	148	148	32.83	148	1.52	148	7.05	148	287.03
30	*j*10*c*10*b*1	163	0.01	-	163	0.19	163	0.73	163	163	163	36	163	1.36	163	0.17	163	284.57
31	*j*10*c*10*b*2	157	0.22	-	157	0.69	157	0.22	157	157	157	66	157	1.29	157	0.38	157	287.49
32	*j*10*c*10*b*3	169	0.01	-	169	0.03	169	0.28	169	169	169	19	169	1.35	169	0.16	169	298.94
33	*j*10*c*10*b*4	159	0.02	-	159	0.11	159	0.25	159	159	159	20	159	1.37	159	0.17	159	287.40
34	*j*10*c*10*b*5	165	0.04	-	165	0.03	165	0.50	165	165	165	33	165	1.37	165	0.76	165	288.01
35	*j*10*c*10*b*6	165	0.06	-	165	0.39	165	0.10	165	165	165	34	165	1.37	165	0.22	165	287.68
	Average	163	0.06	-	163	0.24	163	0.35	163	163	163	34.67	163	1.35	163	0.31	163	289.01

Continued on next page

Table 7.1. – *Continued from previous page*

		PSO		QIA*	AIS		GA		ACO*	ACS*	B&B		CP		HA		GSO	
No	**Instance**	C_{max}	**CPU**	C_{max}	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	C_{max}	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	**CPU**
36	***j*10*c*10*c*1**	115	a	-	115	a	115	a	115	116	127	c	114	4.21	115	7.74	115	238.65
37	***j*10*c*10*c*2**	117	a	-	119	a	119	a	119	119	116	1100	116	1.90	119	4.53	119	259.20
38	***j*10*c*10*c*3**	116	a	-	116	a	116	a	118	119	133	c	116	3.49	116	15.60	116	259.22
39	***j*10*c*10*c*4**	120	a	-	120	a	120	a	123	124	135	c	119	6.58	120	2.29	120	272.19
40	***j*10*c*10*c*5**	125	a	-	126	a	126	a	126	129	145	c	125	5.60	126	97.59	126	274.68
41	***j*10*c*10*c*6**	106	a	-	106	a	106	a	108	109	112	c	104	6.04	106	12.28	106	246.54
	Average	116.50	a	-	117	a	117	a	118.17	119.33	128	c	115.67	4.64	117	23.34	117	258.42
42	*j*15*c*5*a*1	178	0.06	-	178	1.09	178	0.12	178	178	178	18	178	2.17	178	0.19	178	768.99
43	*j*15*c*5*a*2	165	0.01	-	165	0.25	165	0.17	165	165	165	35	165	1.34	165	0.16	165	770.05
44	*j*15*c*5*a*3	130	0.01	-	130	0.25	130	0.12	130	130	130	34	130	1.29	130	0.23	130	848.82
45	*j*15*c*5*a*4	156	0.01	-	156	1.39	156	0.27	156	156	156	21	156	1.49	156	0.18	156	873.65
46	*j*15*c*5*a*5	164	0.00	-	164	0.27	164	0.14	164	164	164	34	164	1.35	164	0.16	164	798.94
47	*j*15*c*5*a*6	178	0.01	-	178	0.27	178	0.27	178	178	178	38	178	1.49	178	0.22	178	811.27
	Average	161.83	0.02	-	161.83	0.59	161.83	0.18	161.83	161.83	161.83	30	161.83	1.52	161.83	0.19	161.83	811.95
48	*j*15*c*5*b*1	170	0.00	-	170	0.22	170	0.07	170	170	170	16	170	1.21	170	0.21	170	749.10
49	*j*15*c*5*b*2	152	0.01	-	152	0.22	152	0.06	152	152	152	25	152	1.44	152	0.17	152	771.05
50	*j*15*c*5*b*3	157	0.03	-	157	0.23	157	0.60	157	157	157	15	157	1.36	157	0.20	157	776.88
51	*j*15*c*5*b*4	147	0.00	-	147	0.25	147	0.17	147	149	147	37	147	1.39	147	0.22	147	779.55
52	*j*15*c*5*b*5	166	0.09	-	166	1.97	166	0.89	166	166	166	20	166	1.32	166	0.28	166	772.69
53	*j*15*c*5*b*6	175	0.02	-	175	0.61	175	0.08	175	175	175	23	175	1.39	175	0.16	175	772.18
	Average	161.17	0.02	-	161.17	0.58	161.17	0.31	161.17	161.50	161.17	22.67	161.17	1.35	161.17	0.21	161.17	770.24

Continued on next page

Table 7.1. – *Continued from previous page*

		PSO		QIA*	AIS		GA		ACO*	ACS*	B&B		CP		HA		GSO	
No	**Instance**	C_{max}	**CPU**	C_{max}	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	C_{max}	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	**CPU**	C_{max}	**CPU**
54	***j*15*c*5*c*1**	85	4.21	-	85	774.48	85	40.08	85	85	85	2131 (b)	85	3.54	86	532.93	86	770.83
55	***j*15*c*5*c*2**	90	1198	-	91	a	91	a	90	91	90	184	90	4.05	92	348.48	92	773.50
56	***j*15*c*5*c*3**	87	2.40	-	87	16.44	87	3.55	87	87	87	202	87	3.97	89	17.31	89	775.11
57	***j*15*c*5*c*4**	89	2.21	-	89	316.94	89	2	91	92	90	c	89	3.17	90	200.02	90	779.21
58	***j*15*c*5*c*5**	74	a	-	74	a	74	a	77	80	84	c	74	6.28	77	32.83	77	788.80
59	***j*15*c*5*c*6**	91	0.19	-	91	18.80	91	0.39	91	91	91	57	91	1.90	91	26.61	91	807.42
	Average	86	a	-	86.17	a	86.17	a	86.83	87.67	87.83	c	86	3.82	87.50	193.03	87.50	782.48
61	***j*15*c*5*d*2**	84	a	-	84	a	84	a	88	86	85	c	84	57.70	86	100.19	86	843.83
62	***j*15*c*5*d*3**	82	a	-	83	a	83	a	86	89	96	c	82	72.33	84	447.79	84	905.30
63	***j*15*c*5*d*4**	84	a	-	84	a	84	a	88	90	101	c	84	865.52	86	271.27	86	880.49
64	***j*15*c*5*d*5**	79	a	-	80	a	79	a	84	85	97	c	79	76.83	81	3.87	80	879.30
65	***j*15*c*5*d*6**	81	a	-	82	a	81	a	84	87	87	c	81	22.40	82	1.22	83	881.61
	Average	82	a	-	82.60	a	82.20	a	86	87.40	93.20	c	82	218.96	83.80	164.87	83.80	878.11
66	*j*15*c*10*a*1	236	0.02	236	236	0.57	236	0.03	236	236	236	40	236	2.15	236	0.221	236	986.54
67	*j*15*c*10*a*2	200	0.21	200	200	29.52	200	0.91	200	200	200	154	200	1.44	200	0.439	200	987.15
68	*j*15*c*10*a*3	198	0.17	198	198	4.31	198	1.86	198	198	198	45	198	2.12	198	0.717	198	989.96
69	*j*15*c*10*a*4	225	0.07	225	225	11.50	225	0.91	225	227	225	78	225	1.64	225	0.527	225	998.13
70	*j*15*c*10*a*5	182	0.51	182	182	1.84	182	0.19	182	182	183	c	182	1.53	182	0.651	182	1012.37
71	*j*15*c*10*a*6	200	0.47	200	200	1.72	200	0.93	200	200	200	44	200	1.94	200	0.786	200	930.55
	Average	206.83	0.24	206.83	206.83	8.24	206.83	0.81	206.83	207.17	207.00	c	206.83	1.80	206.83	0.56	206.83	984.12
72	*j*15*c*10*b*1	222	0.02	222	222	2.66	222	0.04	222	222	222	70	222	1.83	222	0.42	222	1057.03
73	*j*15*c*10*b*2	187	0.01	187	187	0.48	187	0.33	187	187	187	80	187	1.75	187	0.27	187	1091.79
74	*j*15*c*10*b*3	222	0.01	222	222	0.44	222	0.02	222	222	222	80	222	1.50	222	0.22	222	1072.76
75	*j*15*c*10*b*4	221	0.01	221	221	0.45	221	0.03	221	221	221	84	221	1.74	221	0.21	221	1076.22
76	*j*15*c*10*b*5	200	0.14	200	200	0.47	200	0.77	200	200	200	84	200	1.65	200	0.40	200	1078.70
77	*j*15*c*10*b*6	219	0.01	219	219	0.45	219	0.02	219	219	219	67	219	1.94	219	0.19	219	1084.46
	Average	211.83	0.03	211.83	211.83	0.82	211.83	0.20	211.83	211.83	211.83	77.50	211.83	1.74	211.83	0.28	211.83	1076.83

However, the solution times of the proposed algorithms, CP, HA, and GSO, are acceptable, since they are below 1600 seconds.

*j*10*c*5*c** type problems are one of the hardest problem group. PSO, B&B and CP reach optimality for all instances. On the other hand, HA could not solve only *j*10*c*5*c*3 instance to optimality. Moreover, GSO solves only *j*10*c*5*c*1 to optimality. However, the solutions of the problems that HA and GSO could not solve to optimality, are only 1 time unit away from the optimal solution. Therefore, the results of HA and GSO are still promising and their solution times are below 1600 seconds. Furthermore, CP is faster than PSO for *j*10*c*5*c*3 instance. CP is dominantly better than the other solution algorithms. For every instance, HA is better than QIA. For only instance 16, GSO performs equal to QIA, for other instances, GSO is better than QIA. For instance 14, HA and GSO perform also better than AIS and GA, since they solve this instance within a solution time below 1600 seconds. For instance 19, HA and GSO performs better than GA, since they reach a promising solution only 1 unit away from the optimal solution within 1600 seconds for this instance.

Like *j*10*c*5*c**, *j*10*c*5*d** type problems are one of the hardest group. Despite the fact that they are hard to solve, HA and GSO could not solve only *j10c5d2* and *j10c5d5* instances to optimality. However, since their makespan values are only 1 unit away from the optimal solution, both HA and GSO are promising in terms of effectiveness and efficiency. PSO, B&B and CP solve all instances to optimality. For instance 18, HA and GSO perform also better than B&B in terms of the solution time. For instance 19, HA and GSO perform better than ACO in terms of solution quality. For instance 20, HA is faster than B&B. For instance 23, HA and GSO perform faster than B&B to reach optimality.

*j*10*c*10*a** is another easy problem type whose instances are all solved to optimality by all algorithms. Since the solution times of the proposed solution methods are below

1600 seconds, the solutions provided are acceptable, and hence, CP, HA, and GSO are still promising.

Like *j*10*c*10*a**, *j*10*c*10*b** type problems are easy to solve, and they are solved to optimality by all algorithms (except QIA) within a solution time of 1600 seconds.

One of the hardest problem types, *j*10*c*10*c**, could not be solved optimality with ease. Instance 36 is solved to optimality by only CP and almost optimal solutions are reached by HA and GSO within a solution time below 1600 seconds. Therefore, even though PSO, AIS, and GA reach the makespan value equal to 115 for instance 36, they could not reach this solution within 1600 seconds. CP also solves instance 37 to optimality while the other algorithms, except B&B, fail to do so. However, for instance 37, PSO's solutions are better than HA and GSO. Nevertheless, PSO could not reach its solution within 1600 seconds and the solutions of HA and GSO are only 2 units away from PSO's solution and only 3 units away from CP's solution which is optimal. Moreover, HA and GSO reach their solutions within 1600 seconds for instance 37. Instance 38 is the only problem in this set to be solved to optimality by all solution methods except ACO, ACS, and B&B. However, PSO, AIS, and GA could not reach to optimality within 1600 seconds. Therefore, for this instance, it can be said that the proposed solution methods are better than PSO. Instance 39 is solved to optimality by only CP. On the other hand, PSO, AIS, GA, HA and GSO reach a solution which is only 1 unit far from the optimal solution. However, PSO, AIS, and GA reach the same solution but in a larger time than 1600 seconds for this instance. Therefore, it can be said that the proposed solution methods perform better than PSO, AIS and GA for this instance. Instance 40 is solved to optimality by only PSO and CP. However, PSO could not reach its solution within 1600 seconds for this instance. Therefore, the proposed CP model is better than PSO for this instance. Moreover, HA and GSO reach a solution only 1 unit far from the optimal solution within 1600 seconds. Therefore, the proposed HA heuristic and GSO metaheuristic still seem promising. Instance 41 is solved to optimality by only CP, while PSO, AIS, GA, HA,

and GSO reach a solution only 2 units far from the optimal solution. While HA and GSO reach their solutions within 1600 seconds, the other algorithms could not reach the same solution within 1600 seconds. Therefore, for this instance, CP is the superior method, and HA and GSO perform better than PSO only in terms of efficiency. CP is superior to others in terms of both effectiveness and efficiency. Both HA and GSO are better than AIS and GA in terms of only solution time, since they reach the same solution with the same gap from the optimal solution provided by CP. For instances 38, 39, and 41, HA and GSO perform better than ACO. While CP is superior to all of the solution methods again, HA and GSO are better than ACS and B&B for all of the instances except the instance 37, since B&B provides the optimal solution within 1600 seconds. However, for other instances, B&B provides solutions which are not optimal within solution times higher than 1600 seconds. According to the results, GSO and especially CP are promising.

Another easy problem type *j*15*c*5*a** is solved to optimality by all solution methods. By the increase in the number of jobs from 10 to 15, the solution time of GSO increases. However, since the solution times of GSO are below 1600 seconds, GSO is still promising.

Like *j*15*c*5*a**, *j*15*c*5*b** type problems are also easy to solve. Therefore, all of the solution methods solve all of the instances to optimality in short computational times.

Another group of the hardest problems is *j*15*c*5*c** type problem. Only instance 59 is solved to optimality by all of the solution methods. On the other hand, other instances are solved to optimality by only PSO and CP. PSO uses too much time to solve instance 55, while the proposed CP model solves it in a very short time. Moreover, PSO could not reach the optimal solution for instance 58 within 1600 seconds, while CP could solve it in 7 seconds only. Therefore, it can be said that for these instances, CP is better than PSO in general. On the other hand, HA and GSO provide promising solutions within acceptable amounts of time.

Another hard problem group is *j*15*c*5*d**. Both PSO and CP solve all of the instances to optimality. However, PSO could not reach these solutions within 1600 seconds. Therefore, CP is better than PSO for these instances. On the other hand, HA and GSO find solutions only 1 unit or 2 units far from the optimal solution within 1600 seconds. Thus, they are still promising solution methods. GSO is better than HA for instance 64, while HA is better than GSO for instance 65. CP is the best solution method in terms of both effectiveness and efficiency as expected, since CP solves all of the instances to optimality. On the other hand, HA and GSO are better than ACO for the instances 57, 61, 62, 63, 64, and 65. Moreover, GSO is also better than AIS and HA for the instance 64 in terms of efficiency compared to AIS and in terms of effectiveness compared to HA. HA and GSO are better than ACS and B&B for the instances 57, 58, 62, 63, 64, and 65.

Although the number of stages increases, since *a* type configuration is easy to solve, all of the instances in *j*15*c*10*a** group are solved to optimality by all solution methods. Due to the increments in both the number of jobs and stages, GSO requires more time to solve the problems. However, since the solution time is below 1600 seconds for all instances, GSO still seems to be promising.

Similar to *j*15*c*10*a**, *j*15*c*10*b** type problems are also easy to solve, because *b* type configuration is easy to solve. All of the solution methods reach optimality within solution times below 1600 seconds. Therefore, the proposed solution methods are still promising.

Since PSO is stronger than at least one of QIA, AIS, GA, ACO, ACS, and B&B for hard cases (*c* and *d* type configurations), the proposed solution methods are also better than at least one of QIA, AIS, GA, ACO, ACS, and B&B for these cases.

According to the results of the computational study, the proposed solution methods seem to be promising even for hard problem instances.

Moreover, regardless of the problem type, CP provides optimal solution for all problem instances. Furthermore, HA and GSO never violate 1600 seconds rule and provide optimal solutions for most of the instances. Another important deduction is that if the bottleneck stage is explicitly shown, the problem becomes easier to solve. For example, *a* and *b* type configurations have a single machine in the middle and the first stages, respectively. If the processing times of the jobs do not vary drastically from stage to stage, it is expected that the stage having one machine is the bottleneck. Since, for these configurations, it is easy to identify the bottleneck stage, the problem is open to manipulation, and thus, it becomes easier to solve. On the other hand, if the bottleneck stage is hard to spot like in configurations *c* and *d*, it becomes harder to obtain the optimal solution . Briefly, it can be said that an HFS scheduling problem could be easier to solve, if one of the stages has only one machine. So, there is a high probability that the stages having only one machine are candidates to be the bottleneck. Nevertheless, the impact of the jobs' processing times should not be underestimated while identifying the bottleneck stage as explained in Chapter 4 as one of the steps for BH.

According to Table 7.2, for most of the instances, LB (Neron et al., 2001) is equal to GLB (Santos et al., 1995). However, LB is better than GLB when LB is not equal to GLB. Therefore, it can be said that LB is strong enough to represent HFS problems. On the other hand, since, for most of the instances, LB is equal to GLB and it is simple to calculate GLB in terms of method and time, GLB is also useful lower bound to represent HFS problems and thus, it can easily be used as a part of an algorithm like in our proposed heuristic solution method HA.

Table 7.2. LB vs. GLB in the test problems

Comparison	Number of test problems	%
GLB>LB	5	6.58
LB>GLB	24	31.58
GLB=LB	47	61.84

CHAPTER 8

CONCLUSION AND FURTHER RESEARCH ISSUES

Although the HFS scheduling problem is NP-hard in the strong sense, HFS configuration is still a good and common choice for having a flexible production system, when many problems related to the deliveries of products are encountered in industry. Obviously, these problems are caused by lack of quality and longer lead times due to insufficient capacity and materials handling systems. Also, HFS configuration is can be improved either from product layout when conventional flow shop production is not sufficient or from process layout by means of separating parts with relatively high demand that have similar manufacturing routings. In this study, motivated by the HJS in fuselage's panel production in the aerospace company, we propose a DES-based framwework that helps in improving the existing HJS configuration towards an HFS configuration. The DES model thus developed for an HFS can be used at least for determining the number of parallel identical machines at each stage. The DES tool can help in designing an HFS to streamline the material flow for some parts/products that are similar in their processing requirements, based on expected demand volumes and cycle time as well. The DES model developed especially fits well for (*HFc, Pm|skip, unavail(brkdwn)|*C_{max}) problem environment. For the scheduling problem in HFS, we propose a CP model, an HA heuristic and a GSO metaheuristic with an IH and an LS algorithm as alternatives to the available solution methods in the literature. It is shown that the proposed solution methods yield better results in terms of effectiveness compared to these solution methods and in terms of efficiency compared to MILP model. The proposed solution methods also provide better results for the case study as a large instance, when compared to the renowned heuristic algorithms in the literature. Since each iteration is tried to be completely independent from the other iterations, HA is ensured to be not stuck in

local optima with its random search strategy. Furthermore, HA's random search strategy is supported with the hybridization of a machine allocation rule FAM and a job sequencing rule ECT that are particularly effective for HFS scheduling problems. By this way, the same permutation schedule through the stages is prevented and the schedule is tried to be made as close as possible to a non-delay structure. Since the optimal schedule is located in a subset of a non-delay schedule set, by the use of HA, it is aimed to have optimal schedules for any problem instances, if possible. If not, a strong GLB, inspired by a previous study in the literature, is calculated in order to cope with the large instance sizes for which getting the optimal solution is not possible within a polynomial time. With the guidance of GLB in HA, the results show that *a* and *b* type configurations are solved to optimality, while *c* and *d* type configurations (hard ones) are solved to near-optimality.

On the other hand, CP model has its own unique structure. Since it has a declarative programming language rather than an imperative programming language like MILP model has, it becomes easy to model an HFS scheduling problem by constraint programming. Moreover, CP model's constraint propagation and depth first search techniques accelarate the reduction process of decision variables domain. Thus, CP model is highy efficient. Furthermore, CP model is the strongest method among all solution methods covered in this study regardless of its simplicity. Therefore, we highly recommend the use of the CP model, since it is the key to solve HFS scheduling problems to optimality. Because it is proven that HFS scheduling problems are strongly NP-hard, what CP model provides is absolutely incredible. CP model's unmatched power is shown with both the case study and test problems by comparing it to other solution methods developed earlier in the literature and here in this study. Therefore, we address our proposed CP model as one of exact approaches including B&B and MILP model to solve the HFS scheduling problems to optimality.

The other proposed solution method is the GSO method enhanced with both IH and LS algorithms. GSO's exploration and exploitation phases are developed carefully in

order to escape from local optima. The parameter setup of GSO is inspired by the study published earlier in the literature. The performance of GSO is promising, due to the fact that it yields good solutions for the case study and test problems in terms of both effectiveness and efficiency. The GSO method is the first one which effectively and efficiently applies GSO to HFS scheduling problems.

Due to the fact that this study is inspired by a case study in an aerospace company, not only are these proposed solution methods useful for researchers to make them extend for further studies, but also they are so capable that they are adaptive for other real-world scenarios in different industries for practitioners.

Similar to the literature, in this study, the objective is also minimizing the makespan. However, real world scenarios require more than one objective, not only the makespan but also energy consumption rates and especially cost items. Therefore, there is a potential that this study can be extended to cover these objectives at the same time as a Multi-Criteria Decision Making (MCDM) problem or one by one as smaller subproblems. However, these real world scenario objectives, like energy consumption rates and cost items, include the usage of electricity power and the cost of inventory holding or more. Measuring these objectives is not easy for most of the cases.

Other than the objective functions, there is an another potential for an extension in the constraints of the HFS configuration. Rather than s_{nsd} included in the processing times of the jobs, s_{sd} may be used to represent the setup times where setup changeovers depend on the sequence of the jobs. Furthermore, for much smaller but more products, there is a possibility to cover them with a group technology method in order to form product families for batching them in the HFS configuration. By this way, new objectives arise such as the number of tool changeovers between consecutive batch families or the batch family with the maximum completion time. Nevertheless, approximately 60% of the literature, the makespan is the most common objective function in HFS scheduling problems.

With the advancements in technology, solution methods are enhanced or hybridized with different techniques in order to improve the solution quality. In this study, the proposed HA guided with GLB and GSO that are enhanced with IH and LS focus on the exploration of different regions for a predetermined number of iterations as a diversification strategy hybridized with the DES model with its parameters CT and MoM_k. Since HA and GSO dynamically improve the makespan according to FAM and ECT strategies, they are also addressed as hybrid solution methods. Moreover, rather than metaheuristics, hyper-heurisctic is the new kid on the block which seeks to select, combine, generate or adapt several simpler heuristics with the contribution of Machine Learning (ML) techniques. The advantage of hyper-heuristics is that they do not try to solve the problem directly like a metaheuristic whose search region is bounded. Instead, they try to find the best metaheuristic, for example, the one which yields the better results among others. Hence, we address this study's approach as a hyper-heuristic, beacuse with the design of the DES model, several solution methods can be simulated in almost-real HFS environment including uncertainty sources as well and the best among them all can be selected as the best scheduling method.

REFERENCES

Adler, L., Fraiman, N., Kobacker, E., Pinedo, M., Plotnicoff, J.C., and Wu T. P. (1993). BPSS: a scheduling support system for the packaging industry. *Operations Research, 41(4),* 641–648.

Alharkan, I. M. (2005). Algorithms for sequencing and scheduling. *Industrial Engineering Department, College of Engineering, King Saud University, Riyadh, Saudi Arabia*.

Arthanary, T. S. and Ramaswamy, K. G. (1971). An extension of two machine sequencing problem. *OPSEARCH: The Journal of the Operational Research Society of India, 8(4),* 10–22.

Askin, R. G. and Standridge, C. R. (1993). Modeling and analysis of manufacturing systems. *John Wiley*.

Bangsow, S. (2015). Tecnomatix plant simulation modeling and programming by means of examples. *Springer*, 109.

Besbes, W., Loukil, T., and Teghem, J. (2006). Using genetic algorithm in the multiprocessor flow shop to minimize the makespan. *Proceedings of the International Conference on Service Systems and Service Management*, 1228–1233.

Brah, S. A. (1988). Scheduling in a flow shop with multiple processors. University of Houston, Houston, TX, *Unpublished Ph.D. Dissertation*, 30-33.

Brah, S. A. and Hunsucker, J. L. (1991). Branch and bound algorithm for the flow-shop with multiple processors. *European Journal of Operational Research, 51(1)*, 88–99.

Brah, S. A. and Loo, L. L. (1999). Heuristics for scheduling in a flow shop with multiple processors. *European Journal of Operational Research*, *113(1)*, 113–122.

Campbell, H. G., Dudek, R. A., and Smith, M. L. (1970). A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, *16(10)*, 630–637.

Carlier, J. and Neron, E. (2001). An exact method for solving the multiprocessor flowshop, *R.A.I.R.O: Operations Research*, *34*, 1–25.

Chamnanlor, C., Sethanan, K., Gen, M., and Chien, C. F. (2017). Embedding ant system in genetic algorithm for re-entrant hybrid flow shop scheduling problems with time window constraints. *Journal of Intelligent Manufacturing*, *28(8)*, 1915–1931.

Chen, B. (1995). Analysis of classes of heuristics for scheduling a two-stage flow-shop with parallel machines at one stage. *Journal of the Operational Research Society, 46(2)*, 234–244.

Chen, L., Bostel, N., Dejax, P., Cai, J. C., and Xi, L. F. (2007). A tabu search algorithm for the integrated scheduling problem of container handling systems in a maritime terminal. *European Journal of Operational Research*, *181(1)*, 40–58.

Dugardin, F., Yalaoui, F., and Amodeo, L. (2010). New multi-objective method to solve reentrant hybrid flow shop scheduling problem. *European Journal of Operational Research, 203(1)*, 22–31.

Eberhart, R. C. and Kennedy, J. (1995). A new optimizer using particle swarm theory. *Proceedings of the Sixth International on Micro Machines and Human Science*, 39–43.

Engin, O. and Doyen, A. (2004). A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Computer Systems*, *20(6)*, 1083–1095.

Engin, O., Ceran, G., and Yilmaz, M. K. (2011). An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems. *Applied Soft Computing*, *11(3)*, 3056–3065.

Google OR-Tools. (2018). https://developers.google.com/optimization/cp/queens.

Grabowski, J. and Pempera, J. (2000). Sequencing of jobs in some production system. *European Journal of Operational Research*, *125(3)*, 535–550.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, *5*, 287–326.

Gupta J. N. D. (1988). Two-stage, hybrid flow shop scheduling problem. *Journal of the Operational Research Society*, *39(4)*, 359–364.

Gupta, J. N. D., Hariri, A. M. A., and Potts, C. N. (1997). Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. *Annals of Operations Research*, *69*, 171–191.

Gupta, J. N. D., Kruger, K., Lauff, V., Werner, F., and Sotskov, Y. N. (2002). Heuristics for hybrid flow shops with controllable processing times and assignable due dates. *Computers & Operations Research*, *29(10)*, 1417–1439.

Ho, J. C. (1995). Flowshop sequencing with mean flowtime objective. *European Journal of Operational Research*, *81*, 571–578.

Hoogeveen, J. A., Lenstra, J. K., and Veltman, B. (1996). Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. *European Journal of Operational Research*, *89(1)*, 172–175.

Hundal, T. S., and Rajgopal, J., (1988). An extension of Palmer's heuristic for the flow-shop scheduling problem. *International Journal of Production Research*, *26 (6)*, 1119–1124.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistic Quarterly*, *1(1)*, 61–68.

Khalouli, S., Ghedjati, F., and Hamzaoui, A. (2009). An integrated ant colony optimization algorithm for the hybrid flow shop scheduling problem. *Proceedings of International Conference on Computers & Industrial Engineering*, 554–559.

Kochhar, S. and Morris, R. J. T. (1987). Heuristic methods for flexible flow line scheduling. *Journal of Manufacturing Systems*, *6(4)*, 299–314.

Kurz, M. E. and Askin, R. G. (2003). Comparing scheduling rules for flexible flow lines. *International Journal of Production Economics*, *85(3)*, 371–388.

Laborie, P., Rogerie, J., Shaw, P., Vilím, P., and Katai, F. (2011). Interval-based language for modeling scheduling problems: An extension to constraint programming. *Applied Optimization*, 111–143.

Lee, C. Y. and Vairaktarakis, G. L. (1994). Minimizing makespan in hybrid flowshops. *Operations Research Letters, 16(3)*, 149–158.

Li, J. and Pan, Q. (2015). Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm. *Information Sciences, 316*, 487–502.

Li, J., Sang, H., Han, Y., Wang, C., and Gao, K. (2018). Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions. *Journal of Cleaner Production, 181*, 584–598.

Liao, C. J., Tjandradjaja, E., and Chung T. P. (2012). An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. *Applied Soft Computing, 12(6)*, 1755–1764.

Luo H., Du, B., Huang, G. Q., Chen, H., and Li, X. (2013). Hybrid flow shop scheduling considering machine electricity consumption cost. *International Journal of Production Economics, 146(2)*, 423–439.

Marichelvam, M. K., Prabaharan, T., and Yang, X. S. (2014). Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. *Applied Soft Computing, 19*, 93–101.

Muthiah-Nakarajan, V. and Noel, M. M. (2016). Galactic swarm optimization: A new global optimization metaheuristic inspired by galactic motion. *Applied Soft Computing, 38*, 771–787.

Naderi, B., Zandieh, M., Khaleghi Ghoshe Balagh, A., and Roshanaei, V. (2009). An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. In press at *Expert Systems with Applications*.

Narasimhan, S. L. and Panwalkar, S. S. (1984). Scheduling in a two-stage manufacturing process. *International Journal of Production Research, 22(4)*, 555–564.

Nawaz, M., Enscore Jr., E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11(1)*, 91–95.

Neron, E., Baptiste, P., and Gupta, J. N. D. (2001). Solving hybrid flow shop problem using energetic reasoning and global operations. *Omega-International Journal of Management Science, 29(6)*, 501–511.

Niu, Q., Zhou, T., and Ma, S. (2009). A quantum-inspired immune algorithm for hybrid flow shop with makespan criterion. *Journal of Universal Computer Science, 15*, 765–785.

Nowicki, E. and Smutnicki, C. (1998). The flow shop with parallel machines: A tabu search approach. *European Journal of Operational Research, 106(2-3)*, 226–253.

Oguz, C. and Ercan, M. (2005). A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. *Journal of Scheduling, 8(4)*, 323–351.

Oguz, C., Zinder, Y. Do, V. H., Janiak, A., and Lichtenstein, M. (2004). Hybrid flow-shop scheduling problems with multiprocessor task systems. *European Journal of Operational Research, 152*, 115–131.

Palmer, D. S. (1965). Sequencing jobs through a multi-stage process in the minimum total time: A quick method of obtaining a near optimum. *Operational Research Quarterly, 16(1)*, 101.

Paternina Arboleda, C. D., Montoya Torres, J. R., Acero Dominguez, M. J., and Herrera Hernandez, M. C. (2008). Scheduling jobs on k-stage flexible flow-shop. *Annals of Operations Research, 164*, 29–40.

Paul, R. J. (1979). Production scheduling problem in the glass-container industry. *Operations Research, 27(2)*, 290–302.

Pinedo, M. L. (2016). Scheduling: Theory, algorithms, and systems, 5th edition, 13–19.

Puaar, P. (2017). Job Sequencing - 9 Processing 'n' jobs on 'm' machines Part 1 of 2 Conditions and Job order.

Rajendran, C. and Chaudhuri, D. (1992). A multistage parallel-processor flowshop problem with minimum flowtime. *European Journal of Operational Research, 57(1)*, 111–122.

Rao, T. B. K. (1970). Sequencing in the order a, b, with multiplicity of machines for a single operation. *OPSEARCH: Journal of the Operational Research Society of India, 7*, 135–144.

Ribas, I., Leisten, r., and Framiñan, J. (2010). Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research, 37(8)*, 1439–1454.

Ruiz, R. and 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. and Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. *European Journal of Operational Research, 205(1)*, 1–18.

Ruiz, R., Serifoglu, F. S., and Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research, 35(4)*, 1151–1175.

Salvador, M. S. (1973). A solution to a special case of flow shop scheduling problems. *Lecture Notes in Economics and Mathematical Systems, 86*, 83–91.

Santos, D. L., Hunsucker, J. L., and Deal, D. E. (1995). Global lower bounds for flow shops with multiple processors. *European Journal of Operational Research, 80(1)*, 112–120.

Shahvari, O. and Logendran, R. (2016). Hybrid flow shop batching and scheduling with a bi-criteria objective. *International Journal of Production Economics, 179*, 239 – 258.

Sheppard., M. (2012). A MATLAB function: allfitdist. *MIT Lincoln Laboratory*.

Sherali, H. D., Sarin, S. C., and Kodialam, M. S. (1990). Models and algorithms for a two-stage production process. *Production Planning & Control, 1(1)*, 27–39.

Sriskandarajah, C. and Sethi, S. P. (1989). Scheduling algorithms for flexible flowshops: Worst and average case performance. *European Journal of Operational Research, 43(2)*, 143–160.

Taşgetiren, M. F., Liang, Y. C., Sevkli, M., and Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research, 177(3)*, 1930–1947.

Wittrock, R. J. (1985). Scheduling algorithms for flexible flow lines. *IBM Journal of Research and Development, 29(4)*, 401–412.

Yang, B. P., Pegden, C. D., and Enscore, E. (1984). A survey and evaluation of static flowshop scheduling heuristics. *International Journal of Production Research, 22 (1)*, 127–141.

APPENDICES

A. Some Plots for the Collection, Analysis and Interpretation of the Data

Processing Times Data

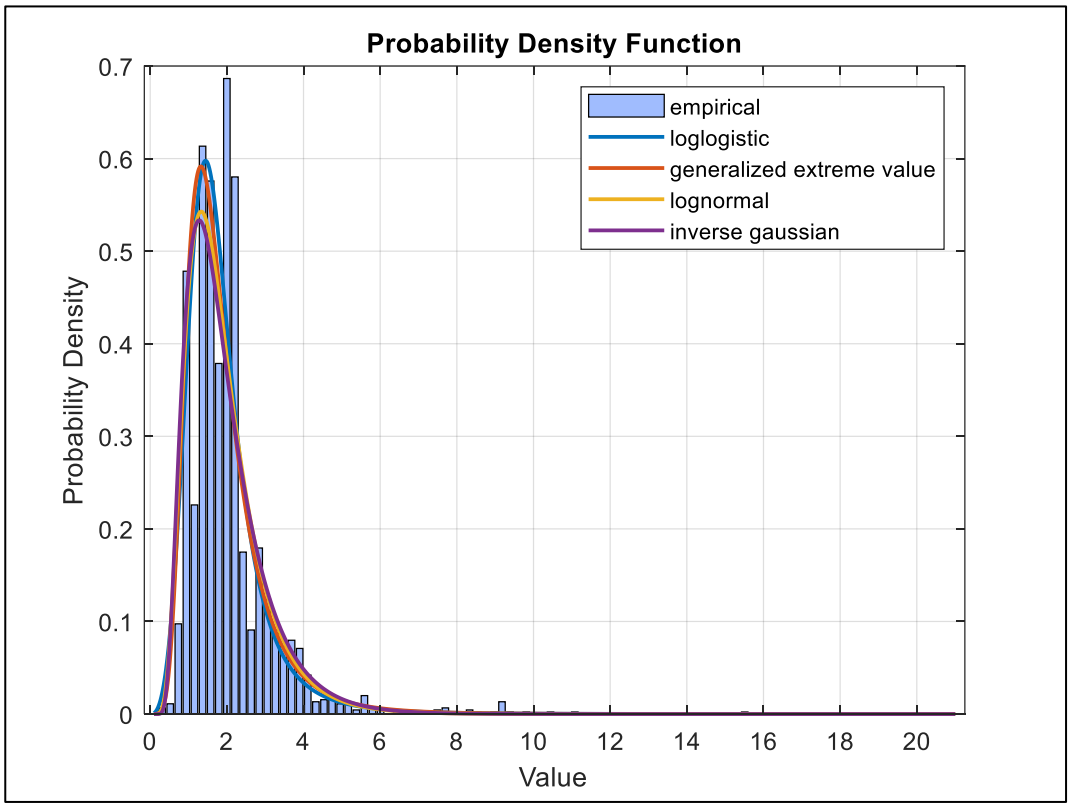


Figure 0.1. The histogram of panel 2's stretching stage processing time data from 2140 raw records

Property	Value
mu	0.5161
sigma	0.2685
DistributionName	'Log-Logistic'
NumParameters	2
ParameterNames	*1x2 cell*
ParameterDescription	*1x2 cell*
ParameterValues	[0.5161,0.2685]
Truncation	*[ ]*
IsTruncated	0
ParameterCovariance	[0,0;0,0]
ParameterIsFixed	*1x2 logical*
InputData	*[ ]*

Figure 0.2. The best distribution with its mean=1.8920 hrs for panel 2's stretching stage processing time data

Machine Breakdown Data

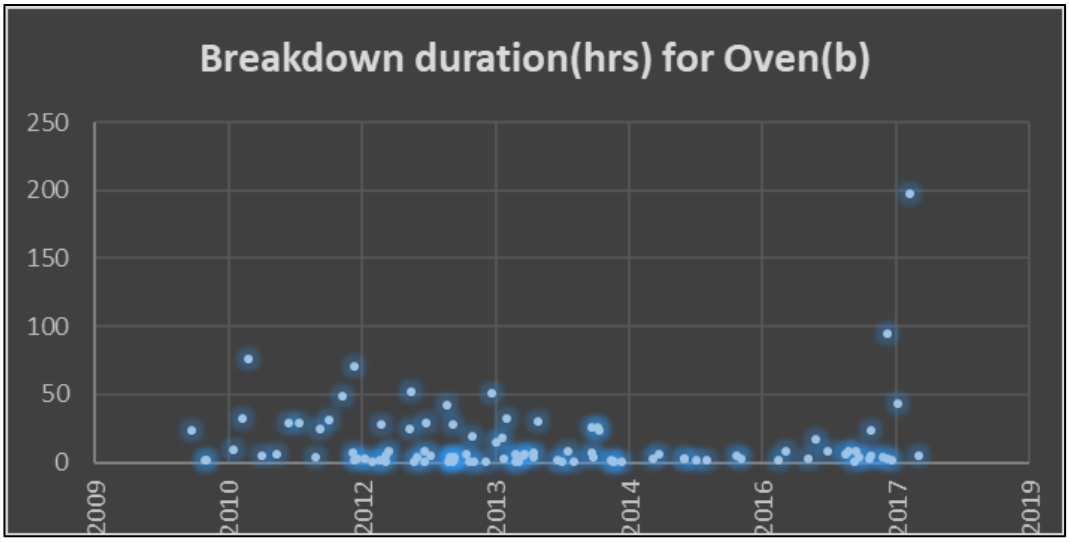


Figure 0.3. Breakdown duration plots of heat treatment stage before cleansing process (2010-2017 data)

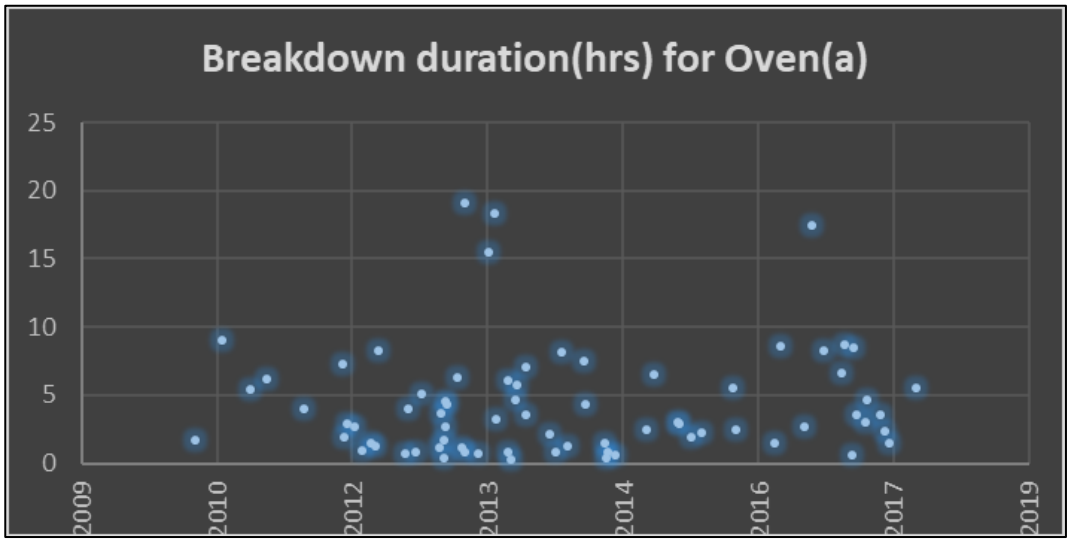


Figure 0.4. Breakdown duration plots of heat treatment stage after cleansing process (2010-2017 data)

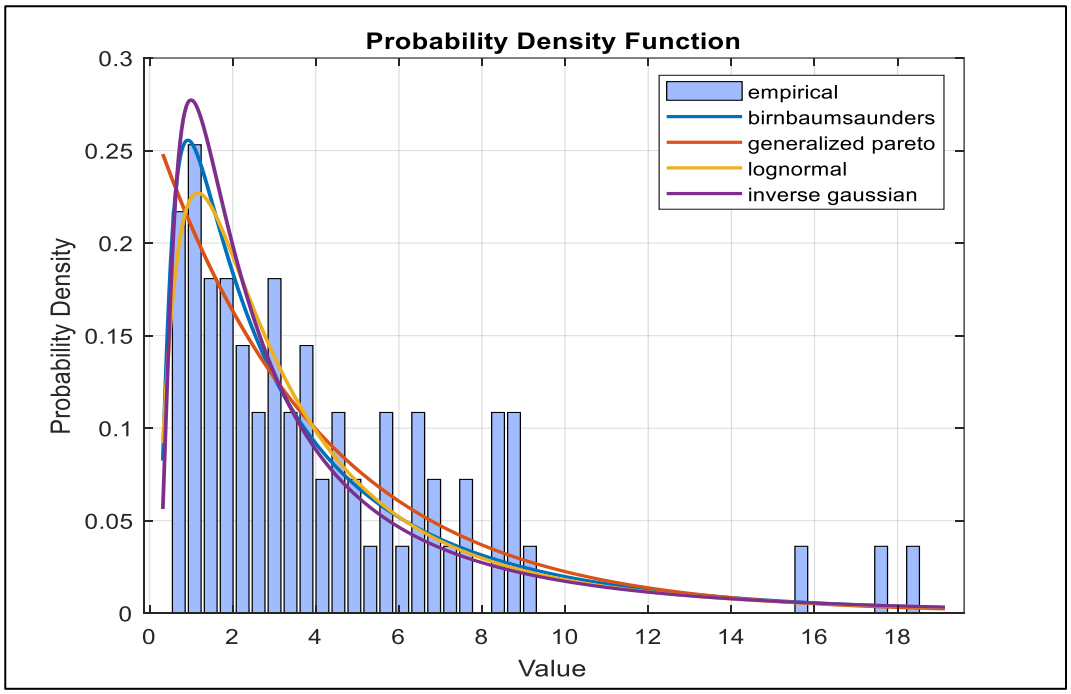


Figure 0.5. The histogram of heat treatment stage's breakdown duration from 73 cleansed records (2010-2017 data)

Property	Value
beta	2.8078
gamma	1.0415
Distributio...	'Birnbaum-Saunders'
NumPara...	2
Parameter...	*1x2 cell*
Parameter...	*1x2 cell*
Parameter...	[2.8078,1.0415]
Truncation	*[ ]*
IsTruncated	0
Parameter...	[0,0;0,0]
ParameterI...	*1x2 logical*
InputData	*[ ]*

Figure 0.6. The best distribution with its mean=4.3306 hours for heat treatment stage's breakdown duration (2010-2017 data)

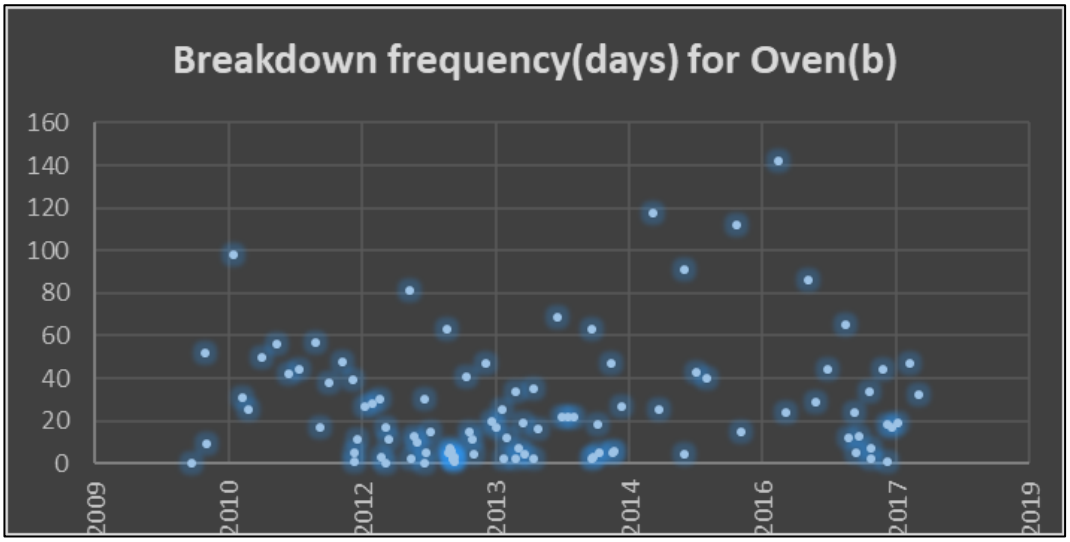


Figure 0.7. Breakdown frequency plots of heat treatment stage before cleansing process (2010-2017 data)

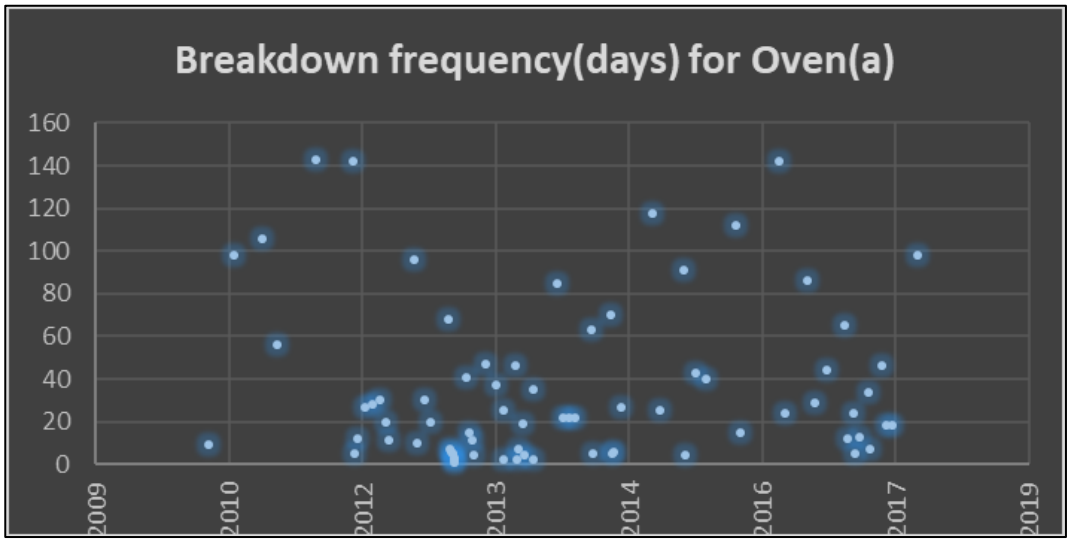


Figure 0.8. Breakdown frequency plots of heat treatment stage after cleansing process (2010-2017 data)

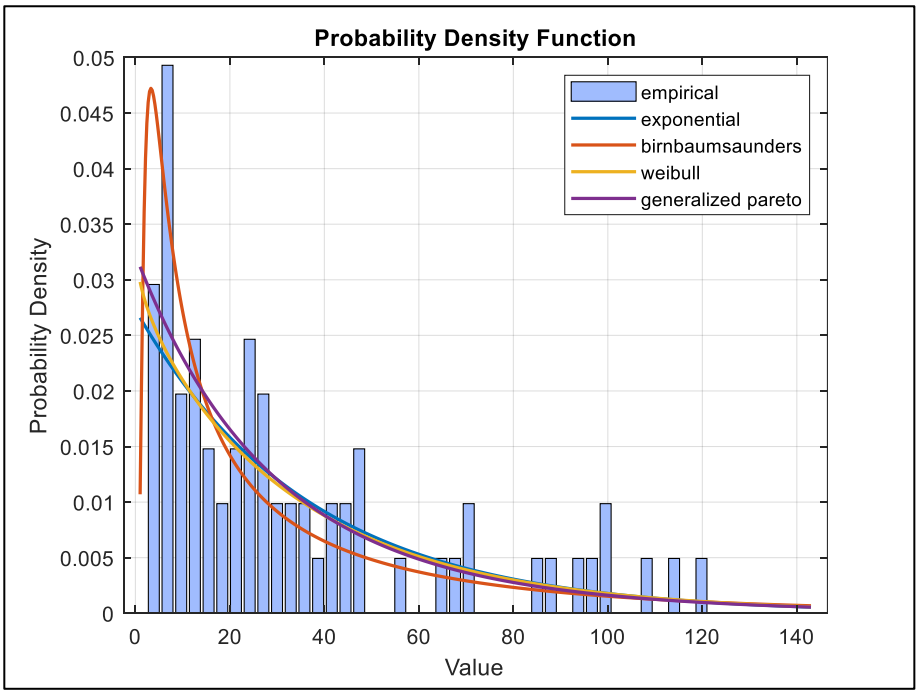


Figure 0.9. The histogram of heat treatment stage's breakdown frequency from 73 cleansed records (2010-2017 data)

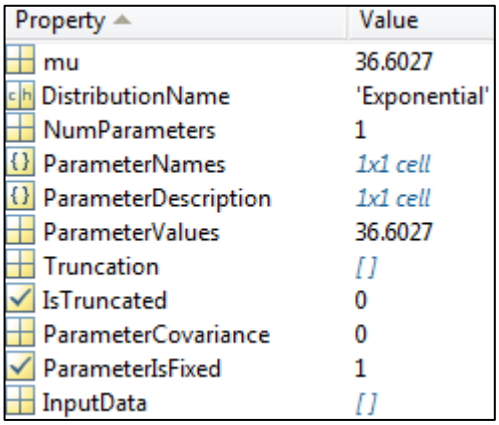


Figure 0.10. The best distribution with its mean=36.6027 days for heat treatment stage's breakdown frequency (2010-2017 data)

B. MoM_k Calculations for the Case Study

Table 0.1. MoM_k calculation table

k \ j	1	2	3	4	5	6	7	8	9	10	11	12	13	sum[(p_{jk}×D_j)/C_k]	MoM_k	Final MoM_k (DES)
	p_{1k}×D_1	p_{2k}×D_2	p_{3k}×D_3	p_{4k}×D_4	p_{5k}×D_5	p_{6k}×D_6	p_{7k}×D_7	p_{8k}×D_8	p_{9k}×D_9	p_{10k}×D_{10}	p_{11k}×D_{11}	p_{12k}×D_{12}	p_{13k}×D_{13}			
1	0.0	441.0	462.0	0.0	0.0	0.0	0.0	661.5	777.0	945.0	1092.0	966.0	546.0	0.87	1	1
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1008.0	0.15	1	1
3	262.5	0.0	0.0	262.5	262.5	262.5	262.5	262.5	262.5	262.5	262.5	262.5	262.5	0.43	1	1
4	0.0	0.0	0.0	1050.0	1050.0	1050.0	1050.0	1050.0	1050.0	1050.0	1050.0	1050.0	1050.0	1.56	2	2
5	0.0	0.0	0.0	4200.0	4200.0	4200.0	4200.0	4200.0	4200.0	4200.0	4200.0	4200.0	4200.0	6.22	7	7
6	2110.5	1984.5	1942.5	2677.5	2667.0	2593.5	2688.0	2331.0	3328.5	3423.0	3528.0	3601.5	2425.5	5.23	6	6
7	262.5	262.5	262.5	0.0	0.0	0.0	0.0	262.5	262.5	262.5	262.5	262.5	262.5	0.35	1	1
8	0.0	0.0	0.0	1732.5	1680.0	1806.0	1774.5	1858.5	1365.0	1564.5	1879.5	1239.0	1144.5	2.38	3	3
9	1365.0	903.0	871.5	3874.5	3853.5	3717.0	3759.0	3465.0	3654.0	3675.0	5460.0	4095.0	3496.5	6.25	7	7
10	262.5	262.5	262.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.12	1	1
11	1207.5	787.5	451.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1543.5	0.59	1	1
12	2530.5	2572.5	1197.0	2919.0	3013.5	3003.0	2919.0	3801.0	4924.5	4137.0	4977.0	4000.5	3937.5	6.51	7	6
13	168.0	157.5	126.0	892.5	892.5	871.5	829.5	546.0	735.0	609.0	619.5	651.0	567.0	1.14	2	2
14	0.0	0.0	0.0	651.0	714.0	682.5	682.5	682.5	682.5	682.5	682.5	682.5	682.5	1.01	2	1
15	0.0	0.0	0.0	987.0	966.0	882.0	924.0	714.0	724.5	714.0	934.5	777.0	735.0	1.24	2	2
16	619.5	651.0	640.5	0.0	693.0	0.0	588.0	0.0	0.0	0.0	0.0	0.0	0.0	0.47	1	1
17	420.0	420.0	420.0	420.0	420.0	420.0	420.0	420.0	420.0	420.0	420.0	420.0	420.0	0.81	1	1
18	1407.0	1701.0	1071.0	1491.0	1543.5	1617.0	1396.5	4179.0	3244.5	3591.0	3643.5	3591.0	4588.5	4.90	5	5
19	220.5	241.5	147.0	210.0	231.0	210.0	189.0	262.5	399.0	357.0	346.5	294.0	409.5	0.52	1	1

C. Processing Times of the Jobs

Table 0.2. The processing times of the jobs at each stage (hours)

k \ j	1	2	3	4	5	6	7	8	9	10	11	12	13
	p_{1k}	p_{2k}	p_{3k}	p_{4k}	p_{5k}	p_{6k}	p_{7k}	p_{8k}	p_{9k}	p_{10k}	p_{11k}	p_{12k}	p_{13k}
1	0	0.42	0.44	0	0	0	0	0.63	0.74	0.9	1.04	0.92	0.52
2	0	0	0	0	0	0	0	0	0	0	0	0	0.96
3	0.25	0	0	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
4	0	0	0	1	1	1	1	1	1	1	1	1	1
5	0	0	0	4	4	4	4	4	4	4	4	4	4
6	2.01	1.89	1.85	2.55	2.54	2.47	2.56	2.22	3.17	3.26	3.36	3.43	2.31
7	0.25	0.25	0.25	0	0	0	0	0.25	0.25	0.25	0.25	0.25	0.25
8	0	0	0	1.65	1.6	1.72	1.69	1.77	1.3	1.49	1.79	1.18	1.09
9	1.3	0.86	0.83	3.69	3.67	3.54	3.58	3.3	3.48	3.5	5.2	3.9	3.33
10	0.25	0.25	0.25	0	0	0	0	0	0	0	0	0	0
11	1.15	0.75	0.43	0	0	0	0	0	0	0	0	0	1.47
12	2.41	2.45	1.14	2.78	2.87	2.86	2.78	3.62	4.69	3.94	4.74	3.81	3.75
13	0.16	0.15	0.12	0.85	0.85	0.83	0.79	0.52	0.7	0.58	0.59	0.62	0.54
14	0	0	0	0.62	0.68	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
15	0	0	0	0.94	0.92	0.84	0.88	0.68	0.69	0.68	0.89	0.74	0.7
16	0.59	0.62	0.61	0	0.66	0	0.56	0	0	0	0	0	0
17	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
18	1.34	1.62	1.02	1.42	1.47	1.54	1.33	3.98	3.09	3.42	3.47	3.42	4.37
19	0.21	0.23	0.14	0.2	0.22	0.2	0.18	0.25	0.38	0.34	0.33	0.28	0.39

D. LEKIN Outputs and Excel Spreadsheets of Dispatching Rules

SPT (C_{max} in minutes)

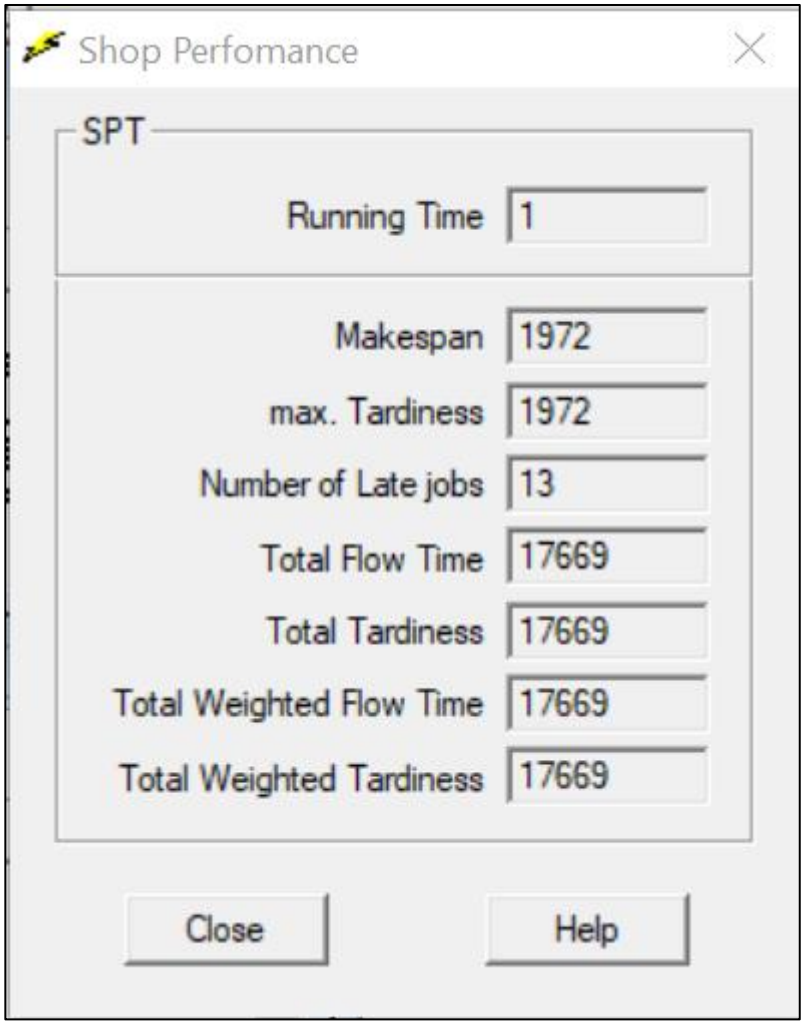


Figure 0.11. SPT rule performance table

LPT (C_{max} in minutes)

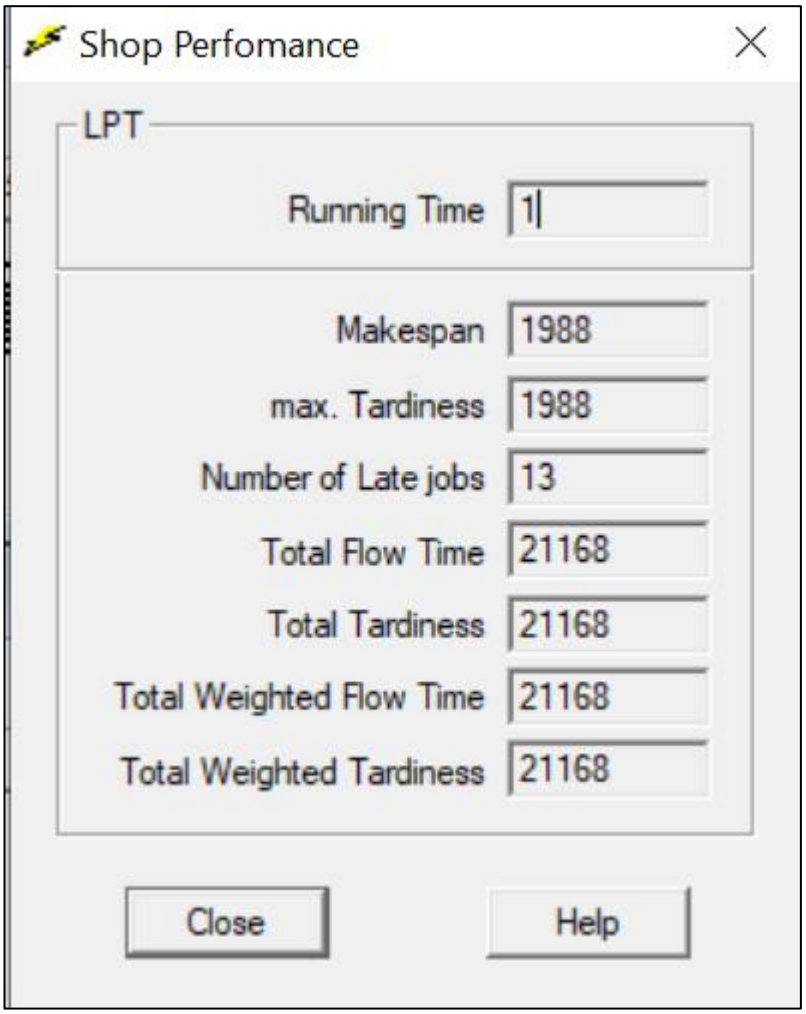


Figure 0.12. LPT rule performance table

STPT

Table 0.3. The job sequence with the STPT rule

k \ *j*	**3**	**2**	**1**	**6**	**4**	**7**	**5**	**8**	**10**	**9**	**12**	**13**	**11**
1	0.44	0.42	0.00	0.00	0.00	0.00	0.00	0.63	0.90	0.74	0.92	0.52	1.04
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.00
3	0.00	0.00	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
4	0.00	0.00	0.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
5	0.00	0.00	0.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
6	1.85	1.89	2.01	2.47	2.55	2.56	2.54	2.22	3.26	3.17	3.43	2.31	3.36
7	0.25	0.25	0.25	0.00	0.00	0.00	0.00	0.25	0.25	0.25	0.25	0.25	0.25
8	0.00	0.00	0.00	1.72	1.65	1.69	1.60	1.77	1.49	1.30	1.18	1.09	1.79
9	0.83	0.86	1.30	3.54	3.69	3.58	3.67	3.30	3.50	3.48	3.90	3.33	5.20
10	0.25	0.25	0.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11	0.43	0.75	1.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.47	0.00
12	1.14	2.45	2.41	2.86	2.78	2.78	2.87	3.62	3.94	4.69	3.81	3.75	4.74
13	0.12	0.15	0.16	0.83	0.85	0.79	0.85	0.52	0.58	0.70	0.62	0.54	0.59
14	0.00	0.00	0.00	0.65	0.62	0.65	0.68	0.65	0.65	0.65	0.65	0.65	0.65
15	0.00	0.00	0.00	0.84	0.94	0.88	0.92	0.68	0.68	0.69	0.74	0.70	0.89
16	0.61	0.62	0.59	0.00	0.00	0.56	0.66	0.00	0.00	0.00	0.00	0.00	0.00
17	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40
18	1.02	1.62	1.34	1.54	1.42	1.33	1.47	3.98	3.42	3.09	3.42	4.37	3.47
19	0.14	0.23	0.21	0.20	0.20	0.18	0.22	0.25	0.34	0.38	0.28	0.39	0.33
Total	7.48	9.89	10.32	20.30	20.35	20.65	21.13	23.52	24.66	24.79	24.85	25.98	27.96

LTPT

Table 0.4. The job sequence with the LTPT rule

k \ *j*	**11**	**13**	**12**	**9**	**10**	**8**	**5**	**7**	**4**	**6**	**1**	**2**	**3**
1	1.04	0.52	0.92	0.74	0.90	0.63	0.00	0.00	0.00	0.00	0.00	0.42	0.44
2	0.00	0.96	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.00	0.00
4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00
5	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	0.00	0.00	0.00
6	3.36	2.31	3.43	3.17	3.26	2.22	2.54	2.56	2.55	2.47	2.01	1.89	1.85
7	0.25	0.25	0.25	0.25	0.25	0.25	0.00	0.00	0.00	0.00	0.25	0.25	0.25
8	1.79	1.09	1.18	1.30	1.49	1.77	1.60	1.69	1.65	1.72	0.00	0.00	0.00
9	5.20	3.33	3.90	3.48	3.50	3.30	3.67	3.58	3.69	3.54	1.30	0.86	0.83
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.25	0.25	0.25
11	0.00	1.47	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.15	0.75	0.43
12	4.74	3.75	3.81	4.69	3.94	3.62	2.87	2.78	2.78	2.86	2.41	2.45	1.14
13	0.59	0.54	0.62	0.70	0.58	0.52	0.85	0.79	0.85	0.83	0.16	0.15	0.12
14	0.65	0.65	0.65	0.65	0.65	0.65	0.68	0.65	0.62	0.65	0.00	0.00	0.00
15	0.89	0.70	0.74	0.69	0.68	0.68	0.92	0.88	0.94	0.84	0.00	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00	0.00	0.66	0.56	0.00	0.00	0.59	0.62	0.61
17	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40
18	3.47	4.37	3.42	3.09	3.42	3.98	1.47	1.33	1.42	1.54	1.34	1.62	1.02
19	0.33	0.39	0.28	0.38	0.34	0.25	0.22	0.18	0.20	0.20	0.21	0.23	0.14
Total	27.96	25.98	24.85	24.79	24.66	23.52	21.13	20.65	20.35	20.30	10.32	9.89	7.48

E. Outputs of the Renowned Heuristic Algorithms

Palmer's Heuristic (Results after flow shop reduction)

Table 0.5. The iob sequence with the Palmer's heuristic

j \ k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	A_j
	$-(m-(2\times k-1))$																			
	-18	-16	-14	-12	-10	-8	-6	-4	-2	0	2	4	6	8	10	12	14	16	18	
1	0.0	0.0	-3.5	0.0	0.0	-2.7	-1.5	0.0	-0.4	0.0	2.3	1.6	0.5	0.0	0.0	7.1	5.6	4.3	3.8	17.1
5	-7.6	0.0	0.0	0.0	0.0	-2.5	-1.5	0.0	-0.3	0.0	1.5	1.6	0.5	0.0	0.0	7.4	5.6	5.2	4.1	14.9
2	-7.9	0.0	0.0	0.0	0.0	-2.5	-1.5	0.0	-0.2	0.0	0.9	0.8	0.4	0.0	0.0	7.3	5.6	3.3	2.5	14.1
7	0.0	0.0	-3.5	-6.0	-5.7	-3.4	0.0	-2.2	-1.1	0.0	0.0	1.9	2.6	5.0	4.7	0.0	5.6	4.5	3.6	11.7
3	0.0	0.0	-3.5	-6.0	-5.7	-3.4	0.0	-2.1	-1.1	0.0	0.0	1.9	2.6	5.4	4.6	7.9	5.6	4.7	4.0	8.6
6	0.0	0.0	-3.5	-6.0	-5.7	-3.3	0.0	-2.3	-1.0	0.0	0.0	1.9	2.5	5.2	4.2	0.0	5.6	4.9	3.6	6.1
4	0.0	0.0	-3.5	-6.0	-5.7	-3.4	0.0	-2.3	-1.0	0.0	0.0	1.9	2.4	5.2	4.4	6.7	5.6	4.3	3.2	5.9
8	-11.3	0.0	-3.5	-6.0	-5.7	-3.0	-1.5	-2.4	-0.9	0.0	0.0	2.4	1.6	5.2	3.4	0.0	5.6	12.7	4.5	1.1
9	-13.3	0.0	-3.5	-6.0	-5.7	-4.2	-1.5	-1.7	-1.0	0.0	0.0	3.1	2.1	5.2	3.5	0.0	5.6	9.9	6.8	-0.8
13	-16.2	0.0	-3.5	-6.0	-5.7	-4.4	-1.5	-2.0	-1.0	0.0	0.0	2.6	1.7	5.2	3.4	0.0	5.6	10.9	6.1	-4.6
10	-18.7	0.0	-3.5	-6.0	-5.7	-4.5	-1.5	-2.4	-1.5	0.0	0.0	3.2	1.8	5.2	4.5	0.0	5.6	11.1	5.9	-4.6
12	-16.6	0.0	-3.5	-6.0	-5.7	-4.6	-1.5	-1.6	-1.1	0.0	0.0	2.5	1.9	5.2	3.7	0.0	5.6	10.9	5.0	-5.7
11	-9.4	-15.4	-3.5	-6.0	-5.7	-3.1	-1.5	-1.5	-1.0	0.0	2.9	2.5	1.6	5.2	3.5	0.0	5.6	14.0	7.0	-6.6

CDS Algorithm (Results after flow shop reduction)

Table 0.6. The job sequence with the CDS algorithm

Iteration	Job position													C_{max}
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	
1	1	4	5	6	7	13	9	10	11	12	8	2	3	33.12
2	1	4	5	6	7	2	8	9	10	12	13	11	3	32.53
3	1	4	5	6	7	2	3	8	9	10	12	11	13	33.55
4	1	2	3	4	5	6	7	8	13	11	10	9	12	32.57
5	1	2	3	4	5	6	7	13	11	8	10	9	12	32.47
6	1	2	3	6	5	4	7	8	13	11	10	9	12	32.57
7	1	2	3	6	5	4	7	8	13	11	9	10	12	32.57
8	1	2	3	5	4	6	7	8	9	11	13	10	12	32.92
9	1	2	3	5	6	4	7	13	11	9	10	12	8	32.92
10	1	2	3	5	6	4	7	13	11	9	10	12	8	32.92
11	3	2	1	5	6	4	7	8	9	13	11	12	10	33.17
12	3	2	1	5	4	6	7	8	13	11	9	10	12	32.82
13	3	2	1	7	6	4	5	8	13	11	9	10	12	32.82
14	3	2	1	4	7	6	5	13	11	9	10	12	8	33.17
15	3	2	1	6	7	4	5	8	13	11	9	10	12	32.82
16	3	2	1	6	4	7	8	5	9	10	12	11	13	33.85
17	2	1	6	4	7	8	5	9	10	12	13	11	3	32.82
18	1	6	4	7	5	13	11	9	10	12	8	2	3	32.67

NEH Algorithm (Results after flow shop reduction)

Table 0.7. The job sequence with the NEH algorithm

j	T_j
13	9.30
11	8.11
10	7.37
12	7.34
9	7.29
8	6.93
5	6.42
7	6.18
4	5.69
6	5.68
1	4.37
2	4.17
3	3.40

Iteration	**Job Position**												
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
1	11	13											
2	11	13	10										
3	11	13	12	10									
4	11	13	12	10	9								
5	11	8	13	12	10	9							
6	11	5	8	13	12	10	9						
7	11	5	8	13	7	12	10	9					
8	4	11	5	8	13	7	12	10	9				
9	4	11	5	8	6	13	7	12	10	9			
10	4	11	5	8	6	13	1	7	12	10	9		
11	4	11	5	8	6	13	1	7	12	10	9	2	
12	4	11	5	8	6	13	1	7	12	10	9	3	2