A SIMULATED ANNEALING-BASED HYPER-HEURISTIC FOR THE FLEXIBLE JOB SHOP SCHEDULING PROBLEM

KELVIN LIM CHING WEI

UNIVERSITI SAINS MALAYSIA

2023

A SIMULATED ANNEALING-BASED HYPER-HEURISTIC FOR THE FLEXIBLE JOB SHOP SCHEDULING PROBLEM

by

KELVIN LIM CHING WEI

Thesis submitted in fulfilment of the requirements
for the degree of
Master of Science

March 2023

ACKNOWLEDGEMENT

First and foremost, I would like to thank my supervisor, Associate Professor Dr. Wong Li Pei for his guidance and encouragement throughout this journey of research. Without Dr. Wong's strong advice and criticism, this research will never be executed smoothly. Apart from that, I also appreciate Dr. Wong's time that he has set aside to review my work and numerous one-on-one discussions about the research especially given his busy schedule. I would also like to thank my co-supervisor, Associate Professor Ir. Dr. Chin Jeng Feng for giving me the opportunity to work under his research grant which asserts financial security during the period of my study. Besides that, Dr. Chin has also provided me tonnes of valuable advice from the perspective of the engineering domain.

I am deeply grateful to my parents (Mr. Lim Seng Yew and Mdm. Ewe Cheng Kim) for being supportive and understanding during my study. I am glad that I was not left alone as my parents have accompanied me through the ups and downs especially when the movement control order is declared. With their love and continuous mental support, I am able to focus on my study with a peace of mind despite the change in the mode of study which none of us have ever experienced it before.

Finally, I would also like to express my gratitude to my research peers (Dr. Choong Shin Siang, Dr. Chuah How Siang, Mr. Fung Chey, Ms. Phang Yuen Chi, Mr. Thevendran A/L Marimuthu, Ms. Tye Yi Wei, Mr. Zhao Chunsheng, Mr. Yong Wei Lun, Mr. Andre Chua Rin Seng and Mr. Beh Boon Seng) for being kind in sharing their knowledge and thoughts with me throughout this research journey.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF SYMBOLS

a	Action in a heuristic block
B	Set of alternative machines
C_{max}	Makespan
d	Euclidean distance
f	Problem state feature
g`	Index of heuristic block in a heuristic scheme
h	Index of problem state feature in a heuristic scheme
i	Index of jobs
J	Set of jobs
j	Jobs, $J = \{j_1, j_2, \ldots, j_x\}$
k	Index of machines
M	Set of machines
m	Machines, $M = \{m_1, m_2, \ldots, m_y\}$
O	Set of operations
O_i	Set of operations of the i-th job, $O_i = \{o_{i,1}, o_{i,2}, \ldots, o_{i,z}\}$
$o_{i,l}$	The l-th operation of i-th job
P	Selection probability of heuristic (i.e. MAR-JSR pair)
$p_{i,l}$	The processing time of the l-th operation of i-th job
p_U	Total processing time of all scheduled operations
p_V	Total processing time of all pending operations
p_T	Total processing time of all actions
Q	Number of MARs
q	Index of MAR
R	Number of JSRs
R	Index of JSR
r_h	Value of the h-th problem state feature
S	Score matrix
T	Average makespan
u	Number of fitness evaluations
x	Number of jobs
y	Number of machines

z	Number of operations
μ	Mean
σ	Standard deviation
θ	Temperature

LIST OF ABBREVIATIONS

ABC	Artificial Bee Colony
ACGP	Adaptive Charting Genetic Programming
ASGP	Genetic Programming with Adaptive Surrogates
BS-HH	Backtracking Search based Hyper-heuristic
CC	Cooperative Co-evolution
CCGP	Genetic Programming with Cooperative Co-evolution
$CCGP^{c}$	Cooperative Co-evolution Genetic Programming with Subtree Selection by Crossover
$CCGP^{cm}$	Cooperative Co-evolution Genetic Programming with Subtree Selection by Crossover and Mutation
$CCGP^{m}$	Cooperative Co-evolution Genetic Programming with Subtree Selection by Mutation
CCGP-FC	Genetic Programming with Cooperative Co-evolution and Feature Construction
CCGP-SM	Surrogate-assisted Genetic Programming with Cooperative Co-evolution
CRO	Chemical Reaction Optimisation
DEHH	Differential Evolution based Hyper-heuristic
FJSP	Flexible Job Shop Scheduling Problem
GA	Genetic Algorithm
GEP	Gene Expression Programming
GLNSA	Global-local Neighbourhood Search Algorithm
GP	Genetic Programming
GPHH	Genetic Programming Hyper-heuristic
GSGP	Genetic Programming with Generation-range-based Surrogates
GWO	Grey Wolf Optimisation
HA	Heuristic Algorithm
HS	Heuristic Scheme
HS*	Global Best Heuristic Scheme
HS′	Neighbour Heuristic Scheme

HSDE	Hybrid Self-adaptive Differential Evolution Algorithm with Heuristic Strategies
HTGA	Hybrid Taguchi-genetic Algorithm
IGIT	Iterated Greedy Insertion Technique
IGIRT	Iterated Greedy Insertion Randomised Technique
IMA	Improved Memetic Algorithm
IMGP	Genetic Programming with Re-initialisation Strategy
IM^2GP	Genetic Programming with Re-initialisation and Adaptive Strategy by Mutation
JSP	Job Shop Scheduling Problem
JSR	Job Sequencing Rule
LLH	Low-level Heuristic
MACROG	Multi-agent Model based on Chemical Reaction Optimisation with Greedy Algorithm
MAR	Machine Assignment Rule
MBB	Multi-objective Parallel Branch-and-bound algorithm for Shared Memory Architectures
MILP	Mixed Integer Linear Programming
MIP	Mixed Integer Programming
MTGP	Multi-tree Genetic Programming
MUGP	Genetic Programming with Adaptive Strategy by Mutation
NiSuFS	Niching-genetic Programming Feature Selection Framework
NSGA-II	Non-sorted Genetic Algorithm II
P-FJSP	Flexible Job Shop Scheduling Problem with Partial Flexibility
PSF	Problem State Features
SA	Simulated Annealing
$SA(MO)_2H$	Self-adaptive Multi-operator and Multi-objective Hyper-heuristic
SA-HH	Simulated Annealing-based Hyper-heuristic
$SA\text{-}HH_{NO\text{-}PSF}$	Simulated Annealing-based Hyper-heuristic based on Heuristic Scheme without Problem State Features
$SA\text{-}HH_{PSF}$	Simulated Annealing-based Hyper-heuristic based on Heuristic Scheme with Problem State Features
SARSA	State–Action–Reward–State–Action
SBH	Shifting Bottleneck Heuristic

SBH-LS	Hybrid of Shifting Bottleneck Heuristic and Local Search
SBH-VNS	Hybrid of Shifting Bottleneck Heuristic and Variable Neighbourhood Search
SLGA	Self-Learning Genetic Algorithm
T-FJSP	Flexible Job Shop Scheduling Problem with Total Flexibility
TS	Tabu Search

LIST OF APPENDICES

HIPER-HEURISTIK BERDASARKAN SIMULASI PENYEPUHLINDAPAN UNTUK PENJADUALAN TUBUHAN KERJA FLEKSIBEL

ABSTRAK

Penjadualan tubuhan kerja fleksibel (FJSP) ialah suatu masalah pengoptimuman yang biasa ditemui dalam industri. Penggunaan mesin selari membolehkan sesuatu operasi diproses menggunakan salah satu mesin daripada sekelompok mesin alternatif. Ini seterusnya mencetuskan dua sub-masalah, iaitu masalah penugasan mesin dan masalah penjujukan kerja. Satu kaedah yang mudah untuk menyelesaikan FJSP adalah dengan mengaplikasikan sepasang peraturan penugasan mesin (MAR) dan peraturan penjujukan kerja (JSR), iaitu pasangan MAR-JSR. Akan tetapi, prestasi setiap pasangan MAR-JSR bersandar kepada ciri-ciri sesuatu masalah. Tambahan pula, dalam sesuatu pelaksanaan algoritma, pasangan MAR-JSR menunjukkan prestasi yang berbeza pada keadaan masalah yang berlainan. Dengan adanya pelbagai pilihatur MAR-JSR, pemilihan satu pasangan MAR-JSR yang sesuai untuk FJSP sememangnya satu cabaran. Hasil yang positif berkenaan pengunaan hiper-heuristik berdasarkan simulasi penyepuhlindapan (SA-HH) dalam menyelesaikan masalah penjadualan yang serupa telah dilaporkan dalam kajian lepas. Oleh itu, penyelidikan ini mencadangkan SA-HH untuk membentuk suatu skema heuristik (HS) yang terdiri daripada pasangan MAR-JSR dalam menyelesaikan FJSP. SA-HH juga menggabungkan satu set ciri-ciri keadaan masalah untuk memudahkan aplikasi pasangan-pasangan MAR-JSR dalam HS berdasarkan keadaan semasa FJSP. Penyelidikan ini mengkaji dua varian SA-HH, iaitu SA-HH berdasarkan HS dengan ciri-ciri keadaan masalah ($SA\text{-}HH_{PSF}$) dan tanpa ciri-ciri keadaan masalah ($SA\text{-}HH_{NO\text{-}PSF}$). Keputusan eksperimen berdasarkan set data penanda aras

menunjukkan bahawa prestasi SA-HH_{PSF} lebih baik daripada SA-$HH_{NO\text{-}PSF}$ dari segi purata tempoh masa mula kerja sehingga tamat dan keputusan ini disokong oleh ujian tanda. Dari segi tempoh masa mula kerja sehingga tamat yang terbaik, SA-HH_{PSF} juga mengatasi ataupun setanding lebih daripada 75% algoritma penanda aras dalam 8 daripada 10 masalah.

A SIMULATED ANNEALING-BASED HYPER-HEURISTIC FOR THE FLEXIBLE JOB SHOP SCHEDULING PROBLEM

ABSTRACT

Flexible job shop scheduling problem (FJSP) is a common optimisation problem in the industry. The use of parallel machines allows an operation to be executed on a machine assigned from a set of alternative machines, raising a combination of machine assignment and job sequencing sub-problems. A straightforward technique to solve the FJSP is to apply a pair of machine assignment rule (MAR) and job sequencing rule (JSR), i.e. a MAR-JSR pair. However, the performance of each MAR-JSR pair is problem-dependent. In addition, within an algorithm execution, the MAR-JSR pair performs differently at different problem states. Given a wide range of MAR-JSR permutations, selecting a suitable MAR-JSR pair for a FJSP becomes a challenge. Positive outcomes on the application of simulated annealing-based hyper-heuristic (SA-HH) in addressing similar scheduling problem has been reported in the literature. Hence, this research proposes the SA-HH to produce a heuristic scheme (HS) made up of MAR-JSR pairs to solve the FJSP. The proposed SA-HH also incorporates a set of problem state features to facilitate the application of MAR-JSR pairs in the HS according to the state of the FJSP. This research investigates two variants of SA-HH, i.e. SA-HH based on the HS with problem state features ($SA\text{-}HH_{PSF}$) and without problem state features ($SA\text{-}HH_{NO\text{-}PSF}$). The experimental results based on the benchmark dataset disclosed that $SA\text{-}HH_{PSF}$ outperforms $SA\text{-}HH_{NO\text{-}PSF}$ on the average makespan as supported by the sign test. $SA\text{-}HH_{PSF}$ also outperforms or on par with more than 75% of the benchmark algorithms on 8 out of 10 instances in terms of the best makespan.

CHAPTER 1

INTRODUCTION

Production scheduling is a process to plan and order the execution of a batch of jobs. The goal is to determine a schedule that specifies the execution order of jobs and the respective time of execution. An effective production scheduling is essential in ensuring the success of operating a production facility (Grobler, 2019). Therefore, production scheduling becomes a critical activity in the manufacturing cycle to determine an efficient schedule that optimises the objectives (e.g. makespan minimisation and flow time minimisation) subjected to a set of constraints (e.g. no pre-emption and job precedence).

The complexity of production scheduling increases following the growing requirements in the industry, e.g. the use of parallel machines and the consideration of stochastic job arrivals. This leads to a variety of problem formulations, namely in the order of increasing complexity: single machine scheduling, flow shop scheduling and job shop scheduling.

1.1 Job Shop Scheduling Problem (JSP)

A job shop scheduling problem (JSP) is defined such that a schedule is needed for a batch of x jobs $J = \{j_1, j_2, \ldots, j_x\}$ on a set of y machines $M = \{m_1, m_2, \ldots, m_y\}$. Each job comprises of a predefined sequence of z operations such that the set of operations for the i-th job is denoted as $O_i = \{o_{i,1}, o_{i,2}, \ldots, o_{i,z}\}$ and $o_{i,l}$ refers to the l-th operation of i-th job. A JSP instance can be denoted in a standard notation known as $x \times y$. Table 1.1 illustrates a simple 2×3 JSP instance with two jobs and three

machines. The values of each cell are written in the notation of $\left(m_{i,l},\ p_{i,l}\right)$ where $m_{i,l}$ refers to the machine allocated for $o_{i,l}$ and $p_{i,l}$ refers to the processing time of $o_{i,l}$.

Table 1.1 A 2 × 3 JSP instance

Job	**Operation (Machine, Processing Time)**		
	$o_{i,1}$	$o_{i,2}$	$o_{i,3}$
j_1	(3, 8)	(1, 3)	(2, 6)
j_2	(3, 1)	(2, 5)	(1, 10)

Based on Table 1.1, j_1 should be executed in the order of m_3 for 8 units of time, m_1 for 3 units of time and m_2 for 6 units of time. Meanwhile, j_2 should be executed in the order of m_3 for 1 units of time, m_2 for 5 units of time and m_1 for 10 units of time. In a job shop, each machine can execute only one operation at a time without pre-emption, e.g., j_2 has to be placed in a queue while m_3 is executing $o_{1,1}$ of j_1. In addition, the execution of each job is subjected to the precedence constraint. For instance, the execution of $o_{1,2}$ could only begin upon completing $o_{1,1}$.

A scheduling conflict occurs when two jobs compete over one another for the same resource. At this point, a decision is needed to determine the job to be prioritised for execution. For instance, j_1 and j_2 will compete for m_3 for the execution of its first operation. Such conflict may occur throughout the scheduling process which eventually raises the JSP.

1.2 Flexible Job Shop Scheduling Problem (FJSP)

A job shop is said to be flexible when one or more operations of a job can be executed by on a machine selected from a set of identical parallel machines

(Waschneck *et al.*, 2017). An industrial example of a flexible job shop can be seen in an aero-engine blade manufacturing plant as described by Zhou, Yang & Zheng (2019b) in a case study where several copies of machines are used to minimise bottlenecks caused by a complicated setup which could takes up to hours compared to its processing time which could be completed within minutes.

Unlike the JSP which focuses solely on the sequencing of jobs, the presence of parallel machines in a flexible job shop creates an additional sub-problem where each operation has to be properly assigned to a suitable machine. This introduces the flexible job shop scheduling problem (FJSP). The FJSP can be further decomposed into the machine assignment and job sequencing sub-problem (Zhou, Yang & Huang, 2020). The machine assignment sub-problem allocates a machine from a given set of alternative machines to each operation of a job. Subsequently, the job sequencing sub-problem orders the execution sequence of the operations allocated to the respective machine. The FJSP has received significant attention among researchers with numerous applications (Xie *et al.*, 2019). This inspires the FJSP to be made the focus of investigation of this research.

In the FJSP, a schedule is needed for a batch of x jobs $J = \{j_1, j_2, \ldots, j_x\}$ and a set of y machines $M = \{m_1, m_2, \ldots, m_y\}$. Each job consists of a set of operations $O_i = \{o_{i,1}, o_{i,2}, \ldots, o_{i,z}\}$ where each operation can be processed on exactly one machine selected from a set of parallel machines $B(o_{i,l}) \subseteq M$, $|B| \geq 1$. An example of 2×3 FJSP instance is described in Table 1.2. The values of each cell denote $p_{i,l,k}$ which refer to the processing time of $o_{i,l}$ using machine m_k.

Table 1.2 A 2 × 3 FJSP instance

Job	**Operation**	**Processing Time (units of time)**		
		m_1	m_2	m_3
j_1	$o_{1,1}$	–	2	3
	$o_{1,2}$	3	1	–
	$o_{1,3}$	5	6	–
j_2	$o_{2,1}$	–	8	–
	$o_{2,2}$	3	–	5
	$o_{2,3}$	10	8	6

When an operation can be executed on a set of alternative machines, the machine assignment sub-problem is raised. For example, $o_{1,1}$ can be assigned to either m_2 or m_3 according to Table 1.2. A decision is needed to determine a suitable machine for the execution of the affected operation. Subsequently, when two or more operations compete for the same resource, the job sequencing sub-problem is raised. Assuming that m_2 was assigned to $o_{1,1}$ in the earlier step, $o_{1,1}$ and $o_{2,1}$ will eventually compete for m_2. To resolve the scheduling conflict, the job with the highest priority is prioritised. The complete schedule generation process for a FJSP will be elaborated in Section 4.3.

While the FJSP consists of two decision levels within a single scheduling problem, it is said to have a greater complexity than JSP (Nouiri *et al.*, 2018). Moreover, the FJSP is NP-hard (Lunardi & Voos, 2018). Generally, solving a combinatorial optimisation problem involves a search process of an optimal solution from a finite set of solutions (Choong, Wong & Lim, 2019). To effectively search for a solution, an algorithm could be designed for a combinatorial optimisation problem such as the FJSP.

1.3 Hyper-heuristic as a Solution for Production Scheduling

Hyper-heuristic is a high-level approach which explores a search space of heuristic components or known low-level heuristics to solve computationally difficult problems (Burke *et al.*, 2010; Drake *et al.*, 2020). Specifically, hyper-heuristic is a method that operates on a heuristic search space instead of a solution space (Burke *et al.*, 2013). In other words, the hyper-heuristic determines a low-level heuristic (LLH) at each point of decision to solve a problem iteratively, rather than searching for a solution which solved the problem directly. In the context of production scheduling, the LLHs may refer to a set of dispatching rules which support decision making in case of a scheduling conflict.

Hyper-heuristic can be classified into generation hyper-heuristic (i.e. heuristic to generate heuristic) and selection hyper-heuristic (i.e. heuristic to select heuristic) (Burke *et al.*, 2010; Drake *et al.*, 2020). A general taxonomy on the classes of hyper-heuristic is presented in Figure 1.1.

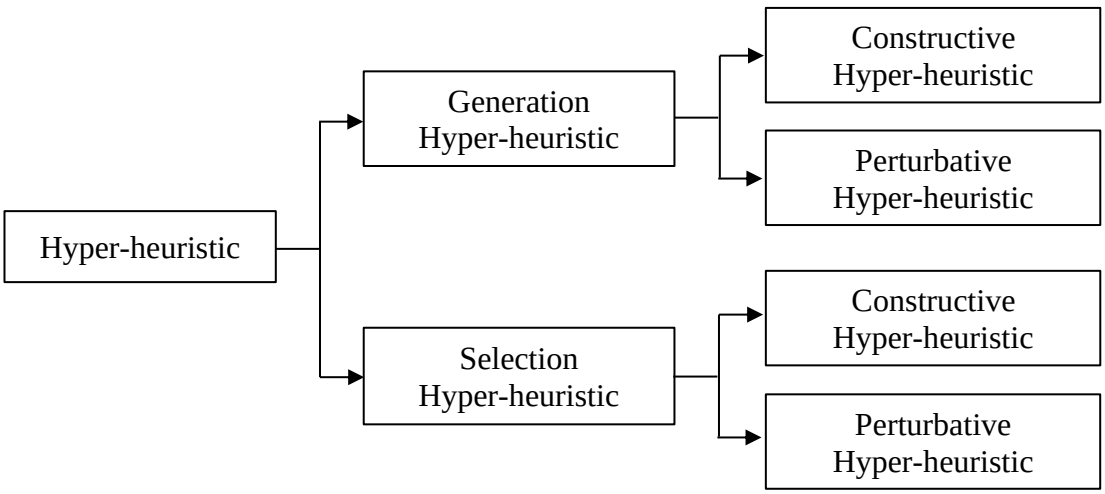

Figure 1.1 Taxonomy of hyper-heuristics

Generation hyper-heuristic derives new heuristics from a set of heuristic components. The set of heuristic components consists of the function set and the terminal set. The function set is usually made up of mathematical operators such as '+' (addition), '−' (subtraction), 'max' (maximum) and 'min' (minimum). In the context of the FJSP, the terminal set consists of the shop features such as processing time and the number of remaining operations (Zhou, Yang & Zheng, 2019a).

Selection hyper-heuristic involves a process of selecting a suitable LLH from a set of common LLHs. A generic framework for the selection hyper-heuristic is visualised in Figure 1.2. The low-level represents the problem domain which consists of the problem representation, evaluation function(s), an initial solution and a set of LLHs which forms the LLH search space. Meanwhile, the high-level represents the hyper-heuristic consists of two major processes, i.e. the LLH selection and the move acceptance. The LLH selection step consists of a selection methodology which selects a suitable LLH from the LLH search space. The selected LLH is applied to produce a solution to an instance of a problem. The move acceptance step will determine whether to accept the solution.

The low- and high-levels are separated from one another by a domain barrier layer. The domain barrier avoids problem-specific information from flowing into the hyper-heuristic (Kalender *et al.*, 2013), allowing the hyper-heuristic to be domain-independent so that the hyper-heuristic can be applied on problems from other domains without a major structural change (Kheiri & Keedwell, 2015).

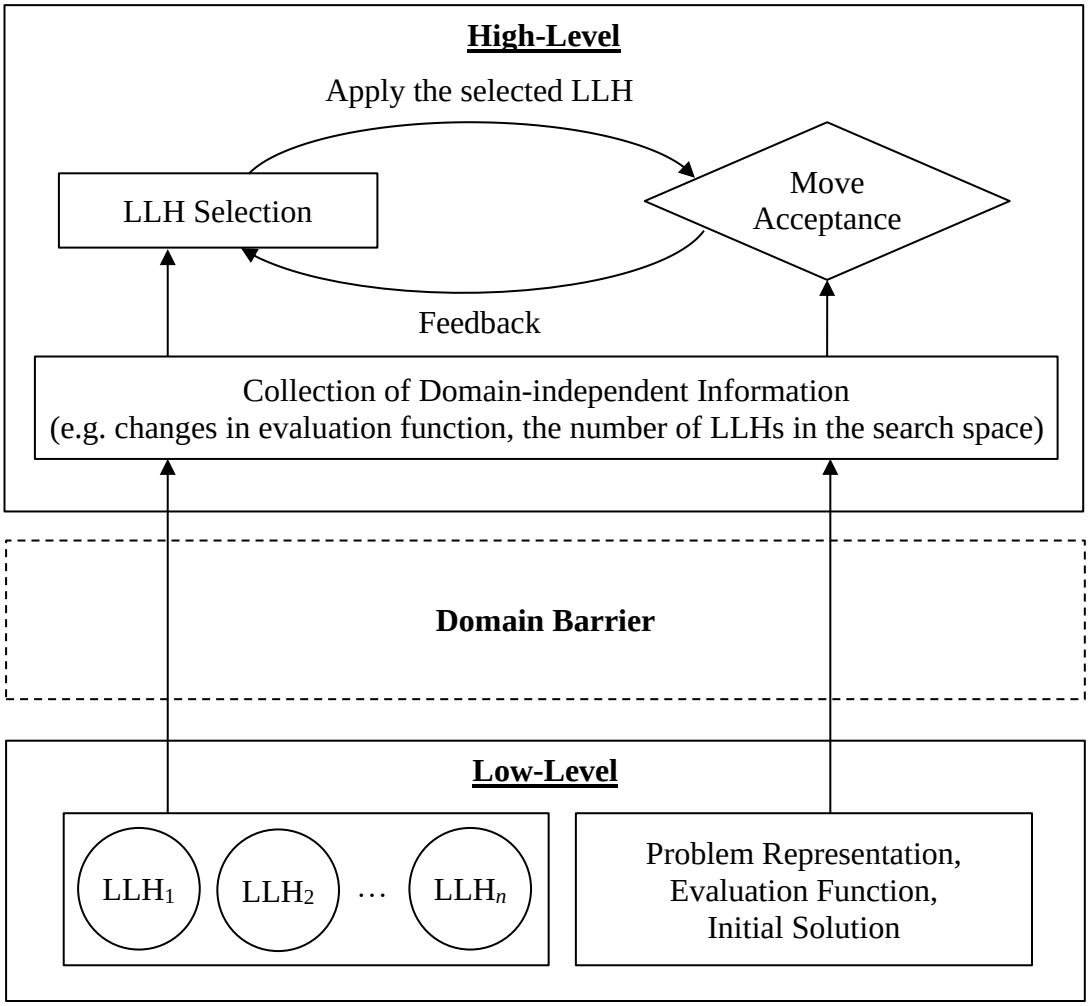

Figure 1.2 Generic selection hyper-heuristic framework (Burke *et al.*, 2010)

Given the availability of a wide range of LLHs in the production scheduling domain, hyper-heuristic could be a potential solution to FJSP. The commonly available LLHs, such as the "shortest processing time" rule and "first come first serve" rule, can be applied directly to perform the scheduling tasks in FJSP due to its simplicity and ability to provide a swift response. However, the performance of these LLHs is domain dependent. Therefore, there is no single universal heuristic which can be applied effectively across all problem instances (Nguyen *et al.*, 2015). In other words, heuristics that perform well in an instance may not necessarily perform well in another instance (Garza-Santisteban, Sanchez-Pamanes, *et al.*, 2019). To overcome the drawback, hyper-heuristic serves as a promising solution in the search of a good-

performing LLH that corresponds to the problem characteristic. This motivates the study of a hyper-heuristic in addressing the FJSP.

1.4 Problem Statements and Research Questions

Heuristic is commonly applied to solve the FJSP following its simplicity and low computational burden. However, it is known that the performance of heuristics varies from one instance to another. To accurately determine a good performing heuristic from a variety of heuristics without adequate domain knowledge is not an easy task. This exerts a greater challenge while solving the FJSP as it can be further modelled into the machine assignment and the job sequencing sub-problems. The sub-problems are addressed by the machine assignment rules (MAR) and the job sequencing rules (JSR) respectively (Zhou, Yang & Zheng, 2019a). A MAR allocates a machine from a set of alternative machines to an operation, whereas a JSR performs arrangement on the order of execution for the operations allocated to a given machine. Both MAR and JSR should exist in pairs as they will be applied one after another when the FJSP is being solved. They are collectively known as a MAR-JSR pair.

Given a variety of MARs and JSRs, there are various formations of problem-dependent MAR-JSR pairs. Selecting a MAR-JSR pair randomly from the list to solve a FJSP instance is less practical because there are chances that a less suitable MAR-JSR pair is being selected, resulting in a poor schedule. Moreover, while solving the same instance, the problem characteristic changes as the schedule is being generated. Based on the common practice, a fixed heuristic is determined and applied throughout the scheduling process. For instance, a MAR-JSR pair is selected at the beginning and the same MAR-JSR pair is used throughout the scheduling process.

However, the performance of the selected MAR-JSR pair may be affected by the change in problem characteristic over time. Considering the problem-specific nature of heuristics, it is proposed that several MAR-JSR pairs be identified and compiled into a heuristic scheme. This allows the selected MAR-JSR pairs to co-operate so that a better schedule can be achieved. Nevertheless, the identification of good-performing MAR-JSR pairs for a specific FJSP instance without a good grasp on the problem remains as an issue, especially when a variety of MARs and JSRs is available for selection. This initiates the first research question, i.e.:

How to create a heuristic scheme which made up combinations of machine assignment rule (MAR) and job sequencing rule (JSR), i.e. MAR-JSR pairs?

Since a heuristic scheme consists of several MAR-JSR pairs, a MAR-JSR pair from the heuristic scheme must be identified to perform scheduling tasks at each decision point. Determining a MAR-JSR pair randomly from the heuristic scheme might still result in a poor FJSP schedule because the choice is not made based on proper reasoning. As the performances of the MARs and the JSRs are problem-specific, the choice of MAR-JSR pair should correspond to the FJSP state. The problem states can be represented by a set of features, which could refer to the progress of the schedule (e.g. the number of completed jobs) or the progress of the remaining schedule (e.g. the remaining of processing times) (Garza-Santisteban, Cruz-Duarte, *et al.*, 2019). A representation on the heuristic scheme alongside a set of problem state features were presented by Garza-Santisteban, Sanchez-Pamanes, *et al.* (2019) in addressing the JSP. Unlike the JSP where the operations are allocated to a fixed machine, the FJSP which involves an additional machine assignment sub-problem raises uncertainties to the problem states because the processing time of a particular operation remains unknown before the machine assignment sub-problem is solved.

This exerts greater challenge in determining the FJSP state and identifying a suitable MAR-JSR pair to perform the scheduling tasks. This leads to the second research question, i.e.:

How can the MAR-JSR pairs in the heuristic scheme be applied at the right state of FJSP?

1.5 Research Objectives

Based on the problem statements and research questions, the following research objectives are defined:

1. To develop a hyper-heuristic in creating a heuristic scheme made up of pairs of machine assignment rule (MAR) and job sequencing rule (JSR) for the flexible job shop scheduling problem (FJSP).
2. To adapt a set of problem state features to categorise states of a problem to facilitate the application of machine assignment rule (MAR) and job sequencing rule (JSR) for the flexible job shop scheduling problem (FJSP).

Among the motivations of this research is to study the potential of a hyper-heuristic in identifying good-performing MAR-JSR pairs to be included in a heuristic scheme to solve FJSP. While the MARs and the JSRs are manually defined heuristics which are applicable to FJSP directly, a selection hyper-heuristic can be employed to perform selection on the MARs and JSRs. For this purpose, this research aims to develop a selection hyper-heuristic, i.e. simulated annealing-based hyper-heuristic (SA-HH) in creating a heuristic scheme made up of MAR-JSR pairs for FJSP.

The motivation of proposing the SA-HH to solve FJSP is twofold. Firstly, the simplicity of the SA-HH as a single-point selection hyper-heuristic which maintains

only a single candidate solution throughout the execution simplifies the initialisation process and eases manipulation on the search process. Secondly, the SA-HH adopts an acceptance strategy which probabilistically accepts a poor move to allow the algorithm to escape from the local optimum.

The second objective aims to adapt a set of problem state features introduced by Garza-Santisteban, Sanchez-Pamanes, *et al.* (2019) and Garza-Santisteban, Cruz-Duarte, *et al.* (2019) to manipulate the MAR-JSR pairs in a heuristic scheme. Instead of randomly selecting a MAR-JSR pair from the heuristic scheme, the problem state features could serve as a guide on the selection. This allows the MAR-JSR pairs to be applied with respect to the problem state which is deemed more accurate as the performance of MAR-JSR pairs is problem-specific. Therefore, leveraging the problem state features in the heuristic scheme serve as an advantage in improving the performance of SA-HH.

1.6 Research Scope

Production scheduling covers a wide range of scheduling problems ranging from single machine scheduling, flow shop scheduling to job shop scheduling. The main focus of this research is to propose an algorithm to solve the FJSP. The FJSP can be further categorised in terms of the flexibility of the flexible job shop, i.e. the FJSP with partial flexibility (P-FJSP) and the FJSP with total flexibility (T-FJSP) (Nouiri *et al.*, 2018). In the P-FJSP, each operation of the job can only be executed on a machine assigned from a subset of machines from the production facility. Meanwhile, in the T-FJSP, each operation of the job can be assigned to any machine in the production facility. This research evaluates the proposed algorithm based on a dataset published

by Brandimarte (1993) which consists of only P-FJSP instances and thus, the research is limited to the P-FJSP.

From the perspective of information evolution, the formulation of the FJSP considered in this research is a static FJSP whereby all the job and machine-related information received at the initial state of scheduling are not subjected to dynamic changes. From the perspective of information quality, both deterministic and stochastic FJSP are being investigated. In the deterministic FJSP, all information is known in advance, whereas information in the stochastic FJSP is subjected to uncertainties. The scope of this research is limited to static and deterministic FJSP via the benchmark dataset by Brandimarte (1993). Meanwhile, the research also focuses on static FJSP considering stochastic job arrival times with slight modifications made to the benchmark dataset to include stochastic job arrival times for each job. Dynamic problems are not considered in this research.

The FJSP considered in this research is a single objective optimisation problem. Optimisation objectives such as the minimisation of flow time, tardiness, machine workload have been investigated in the literature of FJSP. Nevertheless, the most commonly considered objective function in the literature of FJSP is the minimisation of makespan (Chaudhry & Khan, 2016). This refers to the minimisation of the completion time of the final completed job. As such, the minimisation of makespan is considered as the sole optimisation objective in this research.

Among the categories of hyper-heuristic, i.e. generation hyper-heuristic and selection hyper-heuristic, this research focuses on the exploration of a selection hyper-heuristic. The LLHs evolved by genetic programming-based generation hyper-heuristics tend to be more competitive the manually defined LLHs in more complex

scheduling problems (Zhou, Yang & Zheng, 2019a). However, due to the nature of genetic programming, the evolved LLHs are in a form of complex tree structure. As a result, evaluating a LLH with a complex structure can be computationally expensive (Nguyen & Zhang, 2017). Therefore, the commonly defined LLHs which involves a simpler implementation are reconsidered. However, the performance of LLHs is sensitive to problem states. Selection hyper-heuristic which is able to select a suitable LLH for a particular instance emerges as a potential solution to this end.

Based on Figure 1.1, selection hyper-heuristic can be further classified based on the nature of the LLHs, i.e. constructive selection hyper-heuristic and perturbative selection hyper-heuristic (Burke *et al.*, 2010; Drake *et al.*, 2020). Constructive hyper-heuristic begins with an empty solution and iteratively extends the partial solution until a complete solution is obtained, whereas perturbative hyper-heuristic improves a complete solution by performing modification on one or more components of the solution until a termination criterion is satisfied. In the context of the FJSP, the heuristics refer to the MARs and JSRs which are examples of constructive hyper-heuristic. Therefore, the scope of this research is limited to constructive selection hyper-heuristic.

1.7 Research Contributions

To achieve the first research objective, the SA-HH is proposed to identify good MAR-JSR pairs. From the literature, the SA-HH introduced by Garza-Santisteban, Sanchez-Pamanes, *et al*. (2019) has been identified as a base reference for this research. The proposed SA-HH by Garza-Santisteban, Sanchez-Pamanes, *et al*. (2019) has been experimented on the JSP, and yet to be experimented on the FJSP. The first

contribution of this research focuses on extending the ability of the SA-HH to solve the FJSP which is of a greater complexity than the JSP. To address the FJSP, both the MAR and the JSR are needed to solve the machine assignment sub-problem and the job sequencing sub-problem respectively, resulting in a MAR-JSR pair. At each point of scheduling, the MAR-JSR pair is applied one after another, i.e. the application of MAR followed by the application of JSR. This differs from the JSP where only the JSR is involved under the absence of the machine assignment sub-problem. The modified SA-HH is expected to include the selection of MAR-JSR pairs instead of individual JSRs. Moreover, the inclusion of MARs increases the size of the search space as MAR-JSR pairs are formed. An additional component, i.e. the scoring mechanism is added to guide the search towards good performing MAR-JSR pairs by rewarding them based on the acceptance outcome. As a result, MAR-JSR pairs with a higher score will have a higher probability of being selected. The enhanced SA-HH will be compared with the benchmark algorithms based on the benchmark dataset by Brandimarte (1993) for the static FJSP.

The second objective of this research emphasises on the need of a representation of the FJSP state to guide the application of MAR-JSR pairs in the heuristic scheme. In a study conducted by Garza-Santisteban, Sanchez-Pamanes, *et al.* (2019) and Garza-Santisteban, Cruz-Duarte, *et al.* (2019), a set of problem state features have been introduced to solve JSP instances. This second contribution of this research adapts the problem state features to handle the additional machine assignment sub-problem in FJSP. Unlike the JSP where the processing time for each operation is known, the presence of parallel machines in FJSP complicates the problem by asserting an uncertainty on each operation's actual processing time. This is because the processing time of an operation varies from one machine to another. Therefore, the

actual processing time for the specific operation remains uncertain until the machine assignment sub-problem is solved. Hence, the problem state features which involve the processing time feature are adapted by taking the uncertain processing time into consideration. This research intends to assess the role of problem state features in the heuristic scheme. Two variants of SA-HH are created for the comparison study, i.e. the SA-HH based on the heuristic scheme with problem state features ($SA\text{-}HH_{PSF}$) and the other without ($SA\text{-}HH_{NO\text{-}PSF}$). The performance of both variants of SA-HH are compared based on the benchmark dataset by Brandimarte (1993) for the static FJSP. The research also intends to evaluate the SA-HH on the static FJSP considering stochastic job arrivals. To simulate the stochastic event, the benchmark dataset by Brandimarte (1993) is modified to include the arrival time information for each job.

1.8 Thesis Outline

This thesis consists of six chapters. It begins with this introduction chapter (i.e. Chapter 1) that presents an overview of the entire research. It highlights the research background, problem statements, research questions, research objectives, research scope and research contributions.

Chapter 2 features a review of literatures related to the study. The review includes a discussion on various formulations of production scheduling problem and an analysis on various applications of exact and approximation algorithms.

Chapter 3 introduces the research methodology. This research is conducted according to a four-phase methodological procedure, i.e. problem analysis, model design & development and model evaluation.

Chapter 4 describes the proposed method, i.e. Simulated Annealing-based Hyper-heuristic (SA-HH) in solving FJSP. The chapter begins with an introduction followed by a detailed explanation on each component of the proposed algorithm. Firstly, the heuristic scheme which is made of problem state features and MAR-JSR pairs is introduced. Subsequently, the major phases of the SA-HH are elaborated, i.e. initialisation, HS perturbation, HS acceptance and temperature update. To assess the role of problem state features in the SA-HH, two variants of SA-HH are introduced, i.e. SA-HH based on HS with problem state features (SA-HH_{PSF}) and SA-HH based on HS without problem state features (SA-$HH_{NO\text{-}PSF}$).

Chapter 5 presents the experimental results. Firstly, results from the tuning of SA-HH parameters is presented and discussed. Subsequently, experimental results on the intra-comparison study between SA-$HH_{NO\text{-}PSF}$ and SA-HH_{PSF} on static and deterministic FJSP are presented. Subsequently, an inter-comparison study with benchmark algorithms on static and deterministic FJSP was conducted and the respective experimental results are presented. This is followed by the presentation and discussion of experimental results from an intra-comparison study between SA-$HH_{NO\text{-}PSF}$ and SA-HH_{PSF} on static and stochastic FJSP.

The thesis is summarised in Chapter 6 with a summary of key findings obtained from this research. Subsequently, the limitation of this research is highlighted. Finally, several research directions which could possibly extended from the findings of this research are suggested.

CHAPTER 2

LITERATURE REVIEW

This chapter presents a review of related work to this research. It begins with Section 2.1 where a review on the flexible job shop scheduling problem (FJSP) is presented. From the literature, it is observed that FJSP can be solved in either the hierarchical approach or the integrated approach. At the same time, following the growing requirements in the industry, various formulations of FJSP have been introduced from the perspective of optimisation objectives and constraints. These approaches and formulations will be reviewed in this section.

Over the years, production scheduling problems have been studied extensively by researchers. In the literature review, a total of 41 papers published between the year 2008 and the year 2022 have been selected from the SCOPUS and the Web of Science database using the keyword “flexible job shop.” The papers are categorised according to the type of approach in solving the FJSP, i.e. exact and approximation algorithms which are reviewed in Section 2.2 and Section 2.3 respectively.

2.1 Flexible Job Shop Scheduling Problem (FJSP)

The JSP is among the most researched combinatorial optimisation problem following its wide application in the manufacturing domain (Hart & Sim, 2016). The JSP is raised when a schedule is needed for a batch for jobs to be executed on the machines in the manufacturing environment. The uniqueness of JSP is that each job follows a unique processing route, and the operations in a single job should be executed in a predefined order.

Following the introduction of the high-mix low-volume production in the industry, parallel machines are employed to handle production lines involving a large product mix. This extends the JSP into the FJSP. Unlike the JSP where each operation is assigned to a fixed machine, the presence of parallel machines provides a set of alternative machines to each operation. This requires the operation to be executed in either one of the machines within the set of alternative machines, creating the machine assignment sub-problem. Therefore, alongside the existing job sequencing sub-problem, the FJSP can be further modelled into two sub-problems, i.e. the machine assignment sub-problem and the job sequencing sub-problem (Zhou, Yang & Huang, 2020). The machine assignment sub-problem involves the assignment of operations to a suitable machine, whereas the job sequencing sub-problem occurs at the machine queue of each machine with the identification of a job to be prioritised for execution.

2.1.1 Approaches in Handling the FJSP

From the literature, the machine assignment and job sequencing sub-problems of FJSP can be handled using either the hierarchical approach or the integrated approach (Fattahi, Jolai & Arkat, 2009). Specifically, the hierarchical approach solves the two sub-problems one after another, whereas the integrated approach solves the two sub-problems simultaneously.

An example of solving the FJSP in a hierarchical approach was presented by Zhang, Mei & Zhang (2019b). The authors solved the FJSP using a routing rule and a sequencing rule. The routing rule is first applied to assign a machine to an operation. If the queue in front of the machine is not empty, the sequencing rule is then applied to select an operation to be executed on the machine. A similar approach was presented

by Zhou, Yang & Zheng (2019a) where both rules are now phrased as the machine assignment rule (MAR) and the job sequencing rule (JSR). Each rule is applied at the relevant decision points to make the necessary decisions, i.e. the MAR for the machine assignment sub-problem and the JSR for the job sequencing sub-problem.

Nouiri *et al.* (2018) presented an integrated approach to solve the FJSP. Instead of employing a rule for each sub-problem, the particle swarm optimisation algorithm is used to solve both machine assignment and job sequencing sub-problems simultaneously. Likewise, Buddala & Mahapatra (2019) also introduced an integrated approach to solve the FJSP using the teaching-learning-based optimisation method. From the examples, the two sub-problems of FJSP are solved as a single problem instead of solving them one after another.

The integrated approach is beneficial because less computational resources are needed since both sub-problems of FJSP can be solved simultaneously (Buddala & Mahapatra, 2019). On the other hand, the hierarchical approach is inspired by the spirit of divide and conquer where the problem complexity can be reduced (Zhou, Yang & Huang, 2020). This results in a simpler implementation as opposed to the integrated approach. While each approach exhibits its own advantage, both approaches are said to be equally feasible in solving FJSP.

2.1.2 Optimisation Objectives

FJSP is a combinatorial optimisation problem where various types of optimisation objectives could be studied. Table 2.1 presents an overview on the optimisation objectives considered in the existing FJSP literature.

Table 2.1 Optimisation objectives considered in existing FJSP literature

Author & Reference	Proposed Method	Optimisation Objective(s) Considered				
		Makespan	**Flow Time**	**Tardiness**	**Machine Workload**	**Others**
Basán *et al.* (2019)	MILP	✔				No. of processing units at each workstation
Bekkar, Belalem & Beldjilali (2019)	IGIT, IGIRT	✔				
Chang, Tsai & Liu, (2014)	Improved GA	✔				
Chang *et al.* (2015)	HTGA	✔				
Chen *et al.* (2020)	SLGA	✔				
Defersha, Obimuyiwa & Yimer (2022)	SA	✔				
F. Zhang *et al.* (2020a)	MUGP, IMGP, IM^2GP		✔			
F. Zhang *et al.* (2020b)	$CCGP^c$, $CCGP^m$, $CCGP^{cm}$		✔			
Ferreira *et al.* (2020)	ABC	✔				
Gao *et al.* (2015)	Heuristic ensembles	✔*		✔*	✔*	
G. Zhang *et al.* (2020)	IMA	✔				
Jiang & Zhang (2018)	GWO	✔				
Kress & Müller (2019)	CP	✔				
Li, Wang & Peng (2022)	HSDE					Delivery delay
Lin *et al.* (2017)	DEHH	✔				
Lin (2019)	BS-HH	✔				
Luo, Lin & Xu (2020)	Selection HH	✔				
Mahmud *et al.* (2022)	$SA(MO)_2HH$					Supply chain cost & environmental sustainability reward

Table 2.1 Optimisation objectives considered in existing FJSP literature (ctd.)

Author & Reference	Proposed Method	Optimisation Objective(s) Considered				
		Makespan	Flow Time	Tardiness	Machine Workload	Others
Marzouki, Belkahla Driss & Ghédira (2017)	MACROG	✔				
Meng *et al.* (2019)	MILP					Energy consumption
Meng *et al.* (2020)	CP	✔				
Nguyen, Zhang & Tan (2018)	ACGP			✔		
Nie *et al.* (2013)	GEP	✔*	✔*	✔*		
Ozturk, Bahadir & Teymourifar (2019)	GEP	✔	✔	✔		
Serna *et al.* (2021)	GLNSA	✔				
Shahgholi Zadeh, Katebi & Doniavi (2019)	ABC-based Heuristic	✔				
Sobeyko & Mönch (2016)	SBH-LS, SBH-VNS			✔		
Soto *et al.* (2020)	MBB	✔			✔	
Tay & Ho (2008)	GP		✔	✔		
Teymourifar *et al.* (2020)	GEP	✔			✔	
Xiang & Liu (2019)	Branch-and-bound		✔			
Yska, Mei & Zhang (2018a)	CCGP-FC		✔			
Yska, Mei & Zhang (2018b)	CCGP	✔*	✔*	✔*		
Zakaria, BahaaElDin & Hadhoud (2019)	NiSuFC		✔*	✔*		
Zhang, Mei & Zhang (2018a)	MTGP		✔			
Zhang, Mei & Zhang (2018b)	ASGP, GSGP		✔			

Table 2.1 Optimisation objectives considered in existing FJSP literature (ctd.)

Author & Reference	Proposed Method	Optimisation Objective(s) Considered				
		Makespan	Flow Time	Tardiness	Machine Workload	Others
Zhang, Mei & Zhang (2019a)	Two stage GPHH		✔			
Zhou & Yang (2019)	CCGP		✔	✔		
Zhou, Yang & Zheng (2019a)	GEP		✔*	✔*		
Zhou, Yang & Huang (2020)	CCGP-SM		✔	✔		
Ziaee (2014)	HA	✔				

Note: Works marked with an asterisk (*) denotes that despite multiple optimisation objectives are being considered in the research, the optimisation objectives are optimised independently.

From the literature, makespan is the most considered optimisation objective (Chaudhry & Khan, 2016). This is in line with compilation of data shown in Table 2.1 where out of the 41 papers reviewed, 24 of them considered makespan as the optimisation objective. Based on Table 2.1, most researchers have studied FJSP as a single objective optimisation problem. Besides that, FJSP could be studied as a multi-objective optimisation problem. There are two ways to handle a multi-objective optimisation problem, i.e. by combining the objectives into a single objective using a weighted sum or by the Pareto-based approach where a set of solutions is obtained (Zhang, Mei & Zhang, 2019b).

2.1.3 Constraints

From another aspect, feasible solutions of an optimisation problem are subjected to a set of constraints. According to Chaudhry & Khan (2016) and Xie *et al.* (2019), a general formulation of FJSP consists of the following constraints:

(i) All machines are idle at time unit 0.

(ii) All jobs are only available after the release date.

(iii) Each operation can only be executed on exactly one machine at any one time.

(iv) Each machine can only execute exactly one operation at any one time.

(v) No pre-emption is allowed.

(vi) Precedence constraint is only applicable among operations of the same job where for each job, the sequence of which the operations are executed is predefined.

Following the real-world requirements, additional constraints can be included in the formulation of FJSP. Bekkar, Belalem & Beldjilali (2019) incorporated the transportation time constraint into the formulation of FJSP. In this research, the transportation time of jobs between machines is considered as a separate parameter. This is as opposed to the general formulation of FJSP where the transportation time is assumed to be included in the processing time (Chaudhry & Khan, 2016).

On the other hand, Teymourifar *et al.* (2020) introduced the limited buffer constraint into the formulation of FJSP. In the general formulation of FJSP, the machine buffers are assumed to be infinite. In other words, a partially finished good may remain in the buffer space of the machine after the execution is completed, whereas the machine continues to execute the subsequent operation in the queue. However, due to the limited buffer spaces and inadequate transportation capacity, a

partially finished good may be required to remain on the machine after the execution is completed (Teymourifar *et al.*, 2020). Since the machine is occupied, the execution of the subsequent operation is delayed.

Li, Wang & Peng (2022) considered a FJSP with the job priority and outsourcing operations constraint. Under the job priority constraint, each job is assigned with a priority index where the execution priority is given to the job with the highest priority. The outsourcing operations constraint is raised when several operations of the job require execution assistance from an external production facility. Therefore, the constraint specifies the available time slots of the external production facility so that both schedules of the main production facility and the outsourcing production facility can be synchronised to produce a feasible schedule. In summary, the consideration of these additional constraints increases the complexity of the FJSP.

2.1.4 Quality of Information

A scheduling problem can be formulated based on assumptions on the quality of information provided. The quality of information, i.e. deterministic or stochastic indicates the degree of uncertainty of the given data (Pillac *et al.*, 2013). The problem is said to be deterministic if no random variables are considered, whereas the problem is said to be stochastic if at least one problem component is associated with a random variable. Although the taxonomy introduced by Pillac *et al.* (2013) was applied in the classification of vehicle routing problems, Gnanavelbabu, Caldeira & Vaidyanathan (2021) formulated the FJSP in a similar fashion. The deterministic variant of the FJSP is described with all the information are known with certainty. The stochastic variant