ACCEPTED ARTICLE

This is an Accepted Manuscript of an article published by Taylor & Francis in ENGINEERING OPTIMIZATION on 23 Aug 2022, available at: https://www.tandfonline.com/doi/10.1080/0305215X.2022.2106477.

Simulated Annealing-based Hyper-heuristic for Flexible Job Shop Scheduling

Kelvin Ching Wei Lim[a], Li-Pei Wong[a] and Jeng Feng Chin[b]

[a]School of Computer Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia; [b]School of Mechanical Engineering, Universiti Sains Malaysia, 14300 Nibong Tebal, Penang, Malaysia

ABSTRACT

The flexible job shop scheduling problem (FJSP) is common in high mix industries such as semiconductor manufacturing. An FJSP is initiated when an operation can be executed on a machine assigned from a set of alternative machines. Thus, an FJSP consists of the machine assignment and job sequencing sub-problems which can be resolved using a pair of problem-dependent machine assignment rules (MAR) and job sequencing rules (JSR). Selecting a MAR-JSR pair that performs efficiently is a challenge. This study proposes a simulated annealing-based hyper-heuristic (SA-HH) for assembling a heuristic scheme (HS) consisting of MAR-JSR pairs with a set of problem state features. Two variants of SA-HH, i.e. SA-HH based on HS with problem state features (SA-HH_{PSF}) and without problem state features (SA-$HH_{NO\text{-}PSF}$), are investigated. In terms of the best makespan, SA-HH_{PSF} outperforms or comparable to over 75% of the benchmark algorithms on 8 out of 10 instances in Brandimarte dataset.

KEYWORDS

manufacturing; production scheduling; machine scheduling; heuristic selection; heuristic scheme

1. Introduction

The flexible job shop scheduling problem (FJSP) has been studied extensively by researchers as an engineering optimisation problem in various industries, such as semiconductor manufacturing (Mönch et al., 2011) and aero-engine blade manufacturing (Zhou, Yang and Zheng, 2019b). The FJSP is made up of a batch of jobs $J = \{j_1, j_2, ..., j_x\}$ and a set of machines $M = \{m_1, m_2, ..., m_y\}$. Each job consists of a pre-defined sequence of operations such that for the i-th job, the set of operations is given as $O_i = \{o_{i,1}, o_{i,2}, ..., o_{i,z}\}$ and $o_{i,l}$ refers to the l-th operation of i-th job. With the presence of parallel machines, each operation can be processed on exactly one machine selected from a set of parallel machines $B(o_{i,l}) \subseteq M, |M| \geq 1$. This raises two sub-problems in FJSP: machine assignment and job sequencing (Brandimarte, 1993).

Solving a FJSP, which is a type of combinatorial optimisation problem, involves the search of an optimal solution from a finite set of feasible solutions (Choong, Wong

CONTACT Li-Pei Wong. Email: LPWONG@USM.MY

and Lim, 2019). There are two types of search algorithms: exact and approximation (Mohan, Lanka and Rao, 2019). Exact algorithms such as branch-and-bound (Soto et al., 2020) and mixed integer linear programming (Zhang et al., 2021) are used to solve FJSP to optimality. However, exact algorithms are only suitable for small-sized FJSP instances because the run time increases with problem size (Sharma and Jain, 2016).

Approximation algorithms do not guarantee optimality especially for large and complex FJSP instances. These algorithms can return sub-optimal solutions within a short period of time. This is particularly useful when computational resources are limited. Heuristic, metaheuristic and hyper-heuristic are examples of approximation algorithms. Heuristic such as dispatching rule is a quick and straightforward optimisation technique. However, its performance depends heavily on problem characteristics (Zhou and Yang, 2019). Metaheuristics such as artificial bee colony algorithm (Ferreira et al., 2020) and differential evolution (Cao, Shi and Chang, 2022) are search algorithms which iteratively explore and exploit the solution space for a solution. Nonetheless, parameter tuning for a metaheuristic can be time-consuming and limit its reusability (Qu et al., 2015).

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 (Drake et al., 2020). Hyper-heuristic can be categorised into the generation hyper-heuristic (i.e. heuristic to generate heuristic) and the selection hyper-heuristic (i.e. heuristic to select heuristic) (Burke et al., 2010, 2013).

The generation hyper-heuristic derives new heuristics from a set of heuristic components and are mostly developed on the basis of the genetic programming (GP) framework (Sabar et al., 2013). However, GP-based hyper-heuristics tend to generate sophisticated heuristics which eventually lead to flexibility and interpretability issues (Nguyen, Zhang and Tan, 2017). Meanwhile, a wide range of heuristics defined by domain experts are publicly available and can be implemented by using a simpler representation. With concerns that these pre-defined heuristics are problem dependent and less applicable in complex scheduling environments (Zhou, Yang and Zheng, 2019a), studies on selection hyper-heuristic are gaining attention.

The selection hyper-heuristic operates on a heuristic search space where a suitable heuristic is selected to solve a problem. Applying a fixed heuristic throughout could be disadvantageous, as the performance of heuristic are problem-dependent. Hence, instead of one, multiple heuristics could be selected (Garza-Santisteban et al., 2019a). The idea is to exploit the strengths of each heuristic by applying the heuristic consecutively whenever a scheduling decision is needed (Kheiri and Keedwell, 2015; Kheiri et al., 2016). Such approach is deemed less feasible on problem-dependent heuristics (e.g. dispatching rules). Another approach proposed by Garza-Santisteban et al. (2019a) includes the use of problem state features to facilitate the application of multiple heuristics, namely a heuristic scheme (HS). In this approach, the choice of heuristic is determined by a mathematical model based on a comparison between the current problem state and the pre-defined problem state in the HS. The choice of heuristic that corresponds to a particular problem state makes it more efficient in solving a problem. While Garza-Santisteban et al. (2019a) has experimented the HS formulation on the JSP, this research applies the design of the HS with adaptations on the formulation of problem state features so that it becomes applicable on the FJSP.

This study is attempted on a static FJSP environment where all job-related information is known in advance with all machines kept idle at the initial state. The following constraints are considered: (1) each machine can only execute one opera-

tion at a time without pre-emption, and (2) the execution of each job is subjected to the precedence constraint. This study considers makespan ($C_{\max}$), which refers to the completion time of the final completed job, as the sole optimisation objective.

To determine a good-performing HS for FJSP, a Simulated Annealing-based Hyper-heuristic (SA-HH) is proposed. The simulated annealing (SA) algorithm was first proposed by Kirkpatrick, Gelatt and Vecchi (1983). It emulates the slow cooling process of metals as the search process in discovering the global optimum of the objective function. By embedding the SA into the hyper-heuristic framework, the SA-HH is formed. The motivation of proposing the SA-HH to solve the FJSP is twofold. Firstly, the SA-HH is a single-point selection hyper-heuristic which maintains a single candidate solution from the heuristic search space. This solution eventually simplifies the initialisation process and allows easier control on the search process. Secondly, instead of 'accept all moves' and 'improvements only', the SA-HH uses an acceptance strategy which probabilistically accepts a poor move to allow the algorithm to escape from the local optimum.

The remainder of the article is organised as follows. Section 2 reviews the related work. Section 3 introduces and describes the proposed algorithm. Section 4 discusses the experimental results. Section 5 summarises the research findings and concludes the article.

2. Related Work

This section reviews different types of hyper-heuristics in tackling various combinatorial optimisation problems. Both generation and selection hyper-heuristics can be further categorised based on the nature of low-level heuristic, i.e. perturbative hyper-heuristic or constructive hyper-heuristic (Burke et al., 2013). Perturbative hyper-heuristic iteratively modifies an existing solution, whereas constructive hyper-heuristic iteratively builds a solution from scratch (Choong, Wong and Lim, 2018). Hence, there are four categories of hyper-heuristics, i.e. perturbative generation, constructive generation, perturbative selection and constructive selection.

Perturbative selection hyper-heuristic has been proposed to solve FJSP where the candidate solution of the heuristic search space is represented by using a sequence of heuristics. In the sequence, each perturbative heuristic is executed based on a predefined order to perturbate the FJSP schedule. Luo, Lin and Xu (2020) used a generic selection hyper-heuristic to manipulate perturbative heuristics into a sequence for the FJSP. At the same time, differential evolution-based hyper-heuristic (Lin et al., 2017) and backtracking search-based hyper-heuristic (Lin, 2019) have been proposed to create new sequences of perturbative heuristics for the FJSP with fuzzy processing time.

Perturbative selection hyper-heuristic has also been proven effective in other domain problems. Bonab et al. (2019) proposed the swarm-based simulated annealing-based hyper-heuristic to sequence perturbative heuristics for clustering. Kheiri and Keedwell (2015) developed a hidden Markov model-based hyper-heuristic to solve the travelling salesman and vehicle routing problems. The authors claimed that the proposed hyper-heuristic outperforms the current model with minimum parameter tuning. A similar hyper-heuristic by Ahmed, Mumford and Kheiri (2019) was applied to solve the urban transit route design problem with better results in less time.

Production scheduling problems like the FJSP are commonly solved using dispatching rules. Sequences of perturbative heuristic can improve robust solutions. However,

forming sequences of a problem-specific constructive heuristic is less feasible. Research on constructive selection hyper-heuristic is already ongoing at the level of the JSP. Garza-Santisteban et al. (2019a) proposed a simulated annealing-based hyper-heuristic (SA-HH) for the selection of multiple constructive heuristics in the form of HS to solve the JSP. In this research, a set of problem state features adapted from a study by Mirshekarian and Šormaz (2016) is introduced to manipulate the constructive heuristics in the HS. This set of features enables the accurate identification of constructive heuristic to be applied with respect to the current problem state. Given the positive outcomes of the SA-HH in solving the JSP, together with the presence of a set of pre-defined heuristics, the SA-HH may emerge as a potential approach in solving the FJSP.

3. Proposed Method

This study proposes the SA-HH to select heuristics into an HS to solve the FJSP. As shown in Algorithm 1, the SA-HH has four phases: initialisation, HS perturbation, HS acceptance and temperature update.

Algorithm 1 SA-HH algorithm

// Initialisation
Generate an initial HS
Evaluate HS by applying HS on FJSP instance
Compute the fitness $f(\text{HS})$
Set HS as the global best HS (denoted as HS*)
Initialise the initial temperature as θ and score matrix as S
while System has not reach max. no. of fitness evaluations **do**
// HS perturbation
    Perturbate HS to form a neighbour HS (denoted as HS$'$)
    Evaluate HS$'$ by applying HS$'$ on FJSP instance
    Compute the fitness $f(\text{HS}')$
// HS acceptance
    if $f(\text{HS}') > f(\text{HS})$ **then**
        HS = HS$'$
        if $f(\text{HS}) > f(\text{HS}^*)$ **then**
            HS*= HS
    else
        if $e^{\frac{\Delta f}{k\theta}} > \text{Uniform}(0,1)$ **then**
            HS = HS$'$
    Update S according to the outcomes of HS acceptance
// Temperature update
    Update θ according to the cooling schedule
    return HS*

3.1. Heuristic Scheme (HS)

The encoding of the hyper-heuristic's solution in the form of HS is first explained. HS is made up of an array of n_G heuristic blocks. Each heuristic block consists of a set of n_H problem state features (f) and a corresponding action (a). The representation of HS was originally presented by Garza-Santisteban et al. (2019a) for the JSP and

adapted in this research for the FJSP. Fig 1 shows an HS with n_G heuristic blocks where $f_{g,h}$ refers to the h-th problem state feature in g-th block, and a_g refers to the action at g-th block.

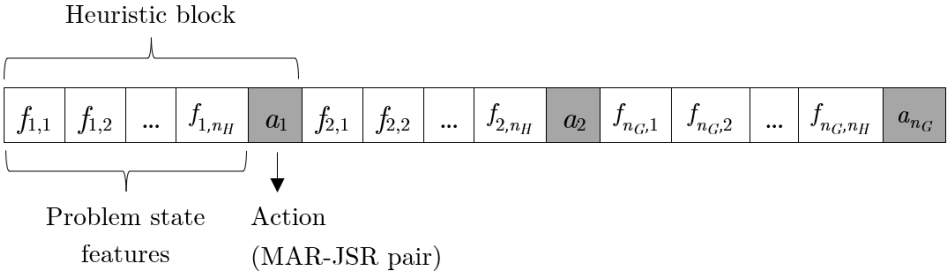


Figure 1. HS of n_G heuristic blocks (Adapted from Garza-Santisteban et al. (2019a))

To represent the state of the FJSP, a set of features is introduced. Each heuristic block of HS in this work incorporates seven problem state features as listed in Table 1. The problem state features include a set of commonly used job-related attributes (i.e. number of operations, number of jobs and processing time). Problem state features can be classified into solution-related features and problem-related features (Garza-Santisteban et al., 2019b). The solution-related features track the current progress, whereas problem-related features track the remaining progress. The following equations are used to define the problem state features:

(1) Total processing times of all scheduled operations,

$$p_U = \sum_{i=1}^{x} \sum_{l=1}^{n_{U_i}} p_{i,l,k'} \tag{1}$$

where:

$$\begin{aligned} U_i &= \text{set of scheduled operations of } j_i, \\ n_{U_i} &= \text{total number of scheduled operations of } j_i, \\ k' &= \text{index of machine assigned to } o_{i,l}, \text{ and} \\ p_{i,l,k'} &= \text{processing time of } o_{i,l} \text{ on } m_{k'}. \end{aligned}$$

(2) Total processing times of all pending operations,

$$p_V = \sum_{i=1}^{x} \sum_{l=1}^{n_{V_i}} \max\left(p_{i,l,1}, p_{i,l,2}, ..., p_{i,l,n_{B(o_{i,l})}}\right) \tag{2}$$

where:

$$\begin{aligned} V_i &= \text{set of pending operations of } j_i, \\ n_{V_i} &= \text{total number of pending operations of } j_i, \text{ and} \\ n_{B(o_{i,l})} &= \text{total number of alternative machines for } o_{i,l}. \end{aligned}$$

(3) Total processing times of all operations,

$$p_T = p_U + p_V \tag{3}$$

Prior to machine assignment, the actual processing time is unknown for a pending operation. Therefore, the maximum possible processing time is used in Equation (2).

Table 1. Problem state features considered

Class	Feature	Description	Equation
Solution-related	Average Processing Time (APT)	Ratio between p_U to p_T.	$\frac{p_U}{p_T}$
	Completed Jobs (PC_JOBS)	Ratio between the number of completed jobs to the total number of jobs, x.	$\frac{\sum_{i=1}^{x}\left[z_i=n_{U_i}\right]}{x}$
	Completed Operations (PC_OPS)	Ratio between the number of completed operations to the total number of operations.	$\frac{\sum_{i=1}^{x}\left[n_{U_i}\right]}{\sum_{i_1}^{x} z_i}$
	Dispersion of Processing Time Index for Scheduled Operations (DPT)	Ratio between the standard deviation of p_U, $\sigma(\mathrm{p}_U)$ to the mean of p_U, $\mu(\mathrm{p}_U)$.	$\frac{\sigma(p_U)}{\mu(p_U)}$
Problem-related	Average Remaining Processing Time (ARPT)	Ratio between p_V to p_T.	$\frac{p_V}{p_T}$
	Average Pending Processing Times per Job (NJT)	Ratio between p_V to the total number of pending operations.	$\frac{p_V}{\sum_{i=1}^{x} n_{V_i}}$
	Dispersion of Processing Time Index for Pending Operations (DNPT)	Ratio between standard deviation of p_V, $\sigma(\mathrm{p}_V)$ and the mean of p_V, $\mu(\mathrm{p}_V)$.	$\frac{\sigma(p_V)}{\mu(p_V)}$

Action in each heuristic block is dictated by a pair of machine assignment rule (MAR) and job sequencing rule (JSR), or collectively known as the MAR-JSR pair. By referring to Figure 1, the HS is created with the following rules:

(1) All heuristic blocks in the HS should contain the same set of problem state features in the same order (e.g. $f_{1,1}$ and $f_{2,1}$ should refer to the same problem state feature).
(2) Each heuristic block should have a unique MAR-JSR pair.

3.2. Initialisation

The SA-HH starts with an initialisation phase where an HS of two heuristic blocks is formed. Each heuristic block is created by selecting a MAR-JSR pair at random and a random value [0, 1] is given to each problem state feature.

The initial HS is evaluated by applying it on an FJSP instance. Solving FJSP is an iterative process where a small batch of operations is being loaded into the schedule at a time, beginning with the first operation of each job. The current problem state is computed and mapped to the problem state defined in each heuristic block by computing its Euclidean distance using Equation (4).

$$\text{Euclidean distance}, d = \sqrt{\sum_{h=1}^{n_H=7} (q_h - r_h)^2} \tag{4}$$

where:

$q_h = h$-th problem state feature value in the g-th block of HS, and
$r_h = h$-th feature value of the current problem state.

The corresponding MAR-JSR pair of the heuristic block with the shortest Euclidean distance is applied to schedule the first batch of operations. A partial schedule is obtained at the end of each iteration. The schedule generation procedure continues to execute until a complete FJSP schedule is obtained with all the operations loaded into the schedule.

Upon obtaining a complete FJSP schedule, the fitness of the HS is computed by using Equation (5). For a minimisation problem, the fitness function is defined as the reciprocal of C_max where a higher fitness value implies a better solution.

$$f\,(\text{HS}) = \frac{1}{C_{\max}} \tag{5}$$

After the evaluation, the execution moves on to the next phase, i.e. HS perturbation.

3.3. HS Perturbation

Starting with an initial temperature θ, the SA-HH iteratively modifies the HS through three perturbative strategies, i.e. S1: Perturbate a problem state feature value; S2: Add a new block; and S3: Remove a block.

The HS perturbation results in HS$'$ which may grow to a maximum size determined via parameter tuning (see Section 4.1). Upon reaching the maximum size, S2 is avoided to prevent HS from growing indefinitely. Meanwhile, when the size of the HS is 2, S3 is avoided because the HS is not meaningful when there is only one heuristic block.

S1 perturbates the HS by modifying a randomly selected problem state feature value in a heuristic block. For example, the value of $f_{1,3}$ is modified from 0.39 to 0.58 in Figure 2.

S2 adds an entirely new heuristic block to HS. The new heuristic block is created with random values being assigned to each problem state feature and a MAR-JSR pair

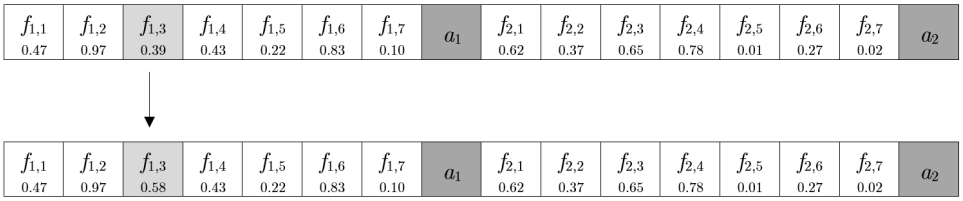


Figure 2. Perturbate a problem state feature value

is selected. Meanwhile, S3 removes an existing heuristic block from HS. The selection of MAR-JSR pairs (S2) and a heuristic block (S1 and S3) for perturbation are based on the scores of the MAR-JSR pairs defined in a score matrix, S, which is given as follows:

$$S = \begin{bmatrix} s_{1,1} & s_{2,1} & \cdots & s_{1,R} \\ s_{2,1} & s_{2,2} & \cdots & s_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ s_{Q,1} & s_{Q,2} & \cdots & s_{Q,R} \end{bmatrix}$$

where Q and R refer to the number of MARs and JSRs considered, respectively, and the index notation $s_{q,r}$, q and r refer to MAR and JSR indices, respectively. The scores in the score matrix are then converted into the probability of heuristic selection using Equation (6).

$$P\left(s_{q,r}\right) = \frac{s_{q,r}}{\sum_{q=1}^{Q}\sum_{r=1}^{R} s_{q,r}} \tag{6}$$

On the basis of the probability distribution, the MAR-JSR pairs with higher score are more likely to be selected and added to HS$'$ via S2. In Figure 3, a new heuristic block is added to the HS.

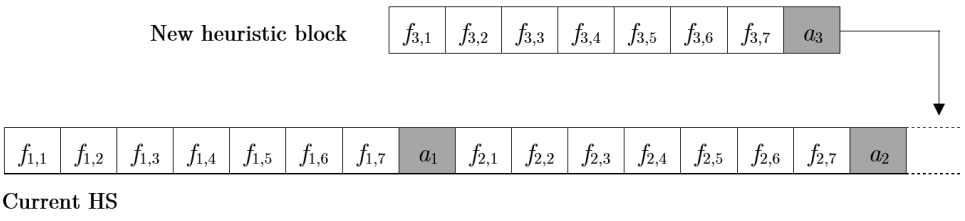


Figure 3. Add a new heuristic block

The heuristic block with MAR-JSR pairs of lower scores are more likely to be selected for perturbation in S1 and elimination in S3. In Figure 4, the second heuristic block is eliminated from HS considering a_2 has the least score.

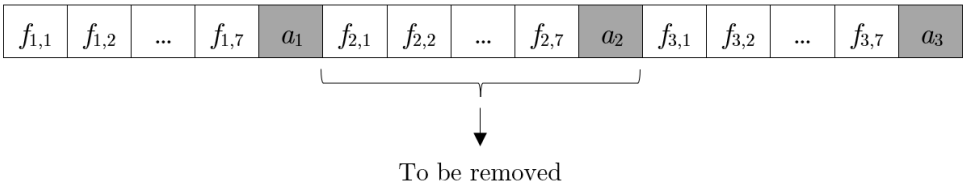


Figure 4. Remove an existing heuristic block

HS′ is evaluated as in Section 3.2. Subsequently, the algorithm enters an HS acceptance phase.

3.4. HS Acceptance

The SA-HH keeps track of the best performing HS (HS*) based on its fitness. The logics on HS acceptance are described in lines 10 to 16 of Algorithm 1. HS′ can replace the HS even if it is less fit by comparing its acceptance probability with a random number. The acceptance probability is computed using Equation (7).

$$\text{Acceptance probability} = e^{\frac{\Delta f}{k\theta}} \tag{7}$$

where:

$$\begin{aligned} \Delta f &= \text{change of fitness function,} \\ k &= \text{constant } (k \text{ is fixed at } 0.05), \text{ and} \\ \theta &= \text{current temperature.} \end{aligned}$$

The scores of each MAR-JSR pair in the HS is updated in S based on the rewards and penalties listed in Table 2. Each MAR-JSR pair receives a score of 1 at the initial state, giving them an equal chance of being selected. To speed up the algorithm's rate of convergence, a reward of 10 is given to increase the impact on the probability distribution so that the selection is biased towards good-performing MAR-JSR pairs. While SA-HH tends to accept poor HS via the acceptance probability, these MAR-JSR pairs will still be rewarded but only half as much. Non-performing MAR-JSR pairs could have their scores reduced to a minimum value of 1 to avoid the MAR-JSR pairs from being neglected in the remaining iterations.

Table 2. Rewarding criteria for MAR-JSR pairs

Acceptance Outcome	Value of Reward
Initial Score	1
Accepted as HS	+10
Accepted as HS*	+10
Accepted as HS via acceptance probability	+5
Rejected	−10
Minimum score	1

3.5. Temperature Update

At the end of each iteration, the temperature of the system (θ) is updated according to a geometric cooling schedule, as stated in Equation (8).

$$\theta_{r+1} = \beta \cdot \theta_r \tag{8}$$

where:

$$\beta = \text{cooling rate, and}$$
$$r = \text{number of iterations.}$$

HS perturbation, HS acceptance and temperature update phases will continue to execute until the maximum number of iterations is reached with the HS* as the final output.

3.6. SA-HH Based on HS Without Problem State Features

To evaluate the performance of the problem state features, an additional variant of the SA-HH is introduced by eliminating the use of problem state features in the HS. To facilitate communication, SA-HH_{PSF} denotes the SA-HH based on the HS with problem state features, whereas the one without will be denoted as SA-$HH_{NO\text{-}PSF}$.

In the SA-$HH_{NO\text{-}PSF}$, HS is created without problem state features. Repetition of heuristic blocks is allowed and the HS can grow indefinitely. The n_G MAR-JSR pairs in the HS are called in a n_G-periodic sequence so that a complete schedule is obtained by repeating the same series. For example, an HS with two heuristic blocks will be executed in a 2-periodic sequence, i.e. $a_1, a_2, a_1, a_2, \cdots$. If a complete schedule is obtained in less iterations than n_G, the unused heuristic blocks are removed. For example, if a complete FJSP schedule is obtained within five iterations using an HS with six heuristic blocks, the sixth heuristic block is removed.

The entire design of the SA-HH_{PSF} (Algorithm 1) remains the same in the SA-$HH_{NO\text{-}PSF}$ except perturbation strategy S1, i.e. perturbating a problem state feature value (as described in Section 3.3), which will be replaced with swapping the position of two randomly selected heuristic blocks.

4. Experiment, Result and Discussion

Experiments are conducted to assess the SA-$HH_{NO\text{-}PSF}$ and the SA-HH_{PSF}. The experimental settings are described first, followed by parameter tuning and finally, results of the comparison studies.

4.1. Experimental Settings

The performance of the SA-$HH_{NO\text{-}PSF}$ and the SA-HH_{PSF} are evaluated using the dataset by Kacem, Hammadi and Borne (2002a,b) which consists of six instances, i.e. I_1 to I_6 and the dataset by Brandimarte (1993) which consists of 10 instances, i.e. MK01 to MK10 with varying number of jobs (x), and the number of machines (y).

The complexity of each instance (given as $x \times y$) is classified as low (0 to 99), medium (100 to 199) or high (200 and above) in Table 3. As no arrival information is provided, all the jobs are assumed to arrive and start at time unit 0 (Chen et al., 2020).

Table 3. Complexity of instances in the benchmark dataset

Instance	x	y	Level of Complexity
I_1	4	5	Low
I_2	10	7	Low
I_3	10	10	Medium
I_4	15	10	Medium
I_5	8	8	Low
I_6	10	6	Low
MK01	10	6	Low
MK02	10	6	Low
MK03	15	8	Medium
MK04	15	8	Medium
MK05	15	4	Low
MK06	10	15	Medium
MK07	20	5	Medium
MK08	20	10	High
MK09	20	10	High
MK10	20	15	High

The SA-$\text{HH}_{\text{NO-PSF}}$ and the SA-HH_{PSF} are implemented using MATLAB 2021a. Two sets of experiments were designed to evaluate the algorithm's performance. Table 4 lists the details of each experiment.

Table 4. Design of experiments

Experiment	Description	Hardware Used	Performance Indicator
I	Performance comparison between SA-$\text{HH}_{\text{NO-PSF}}$ and SA-HH_{PSF}	Windows server with Intel Xeon CPU E3-1220 v5 3.00 GHz processors and 24 GB of memory	Average $C_{\max}$ of 30 runs
II	Performance comparison among SA-$\text{HH}_{\text{NO-PSF}}$, SA-$\text{HH}_{\text{PSF}}$ and various benchmark algorithms	Ubuntu server with Intel Core i7-3930K 3.20 GHz processors and 32 GB of memory	Best $C_{\max}$ within 30 runs

4.2. Parameter Tuning

The initial temperature, cooling rate, number of fitness evaluations and the maximum size of HS are tuned experimentally. Table 5 lists the three levels of values for parameter

tuning, i.e. low, medium and high. As such, $3^4 = 81$ parameter combinations are generated for the SA-HH$_{\text{PSF}}$, whereas $3^3 = 27$ parameter combinations are generated for the SA-HH$_{\text{NO-PSF}}$.

For each parameter configuration, the algorithms are executed 30 times on a representative instance, i.e. MK06 with medium complexity. The configuration that enables the algorithm to achieve the lowest average $C_{\max}$ is identified as the reference configuration which is then compared with the remaining configurations using the two-sample T-test. Configurations with equal performance as the reference configuration are grouped together and eventually, the final configuration is determined by identifying the mode of each parameter. Table 5 presents the outcome of parameter tuning which will be applied in the remaining experiments.

Table 5. Values before and after parameter tuning for SA-HH$_{\text{NO-PSF}}$ and SA-HH$_{\text{PSF}}$

Parameter	Parameter Values Before Tuning			Parameters Values After Tuning	
	High	Medium	Low	SA-HH$_{\text{NO-PSF}}$	SA-HH$_{\text{PSF}}$
Initial temperature	50,000	30,000	10,000	30,000	30,000
Cooling rate	0.900	0.500	0.005	0.900	0.900
Number of fitness evaluations	3,000	2,500	2,000	3,000	3,000
Maximum number of heuristic blocks	30	20	10	*Not applicable*	20

4.3. Experimental Results

Experiment I compares the SA-HH$_{\text{NO-PSF}}$ and the SA-HH$_{\text{PSF}}$ to assess the effect of problem state features on the algorithm's performance. For each instance, both variants of the SA-HH are executed for 30 times and the average $C_{\max}$ are computed. A sign test (Derrac et al., 2011) is performed on the average $C_{\max}$ of each algorithm at the significance level of 0.05. The paired treatment values for the sign test are tabulated in Table 6.

A null hypothesis is defined such that no difference between T_1 and T_2, whereas the alternative hypothesis is that a difference exists between T_1 and T_2. Based on the result in Table 6, there are 13 positive signs, two negative signs and a tied match. Hence, the p-value is 0.00451. Given that the p-value < 0.05, the null hypothesis is rejected. A conclusion is drawn such that a difference exists on the median of the signed differences. This indicates that SA-HH$_{\text{PSF}}$ significantly outperforms the SA-HH$_{\text{NO-PSF}}$.

Table 6. Paired treatment values for sign test

Instance	Average $C_{\max}$ of 30 runs (units of time)		$T_1 - T_2$	Sign
	SA-HH$_{\text{NO-PSF}}$ (T_1)	SA-HH$_{\text{PSF}}$ (T_2)		
I_1	11.13	11.27	−0.14	−
I_2	11.63	11.33	0.30	+
I_3	8.20	8.00	0.20	+
I_4	13.17	12.20	0.97	+
I_5	16.23	15.73	0.50	+
I_6	7.07	7.30	−0.23	−
MK01	42.17	42.00	0.17	+
MK02	30.70	28.73	1.97	+
MK03	204.00	204.00	0.00	#
MK04	70.07	69.60	0.47	+
MK05	182.07	178.07	1.40	+
MK06	75.23	73.83	3.10	+
MK07	157.97	154.87	4.00	+
MK08	525.93	523.87	2.06	+
MK09	327.33	326.90	0.43	+
MK10	239.40	235.10	4.30	+

Tied match.

The results suggest that the problem state features in the SA-HH$_{\text{PSF}}$ are important in manipulating the application of MAR-JSR pairs with respect to the actual problem state. The choice of MAR-JSR pair in the SA-HH$_{\text{PSF}}$ is determined by calculating the problem state features are measuring the Euclidean distances between the current problem state and each heuristic block of the HS. As the performance of MARs and JSRs are problem sensitive, the selected MAR-JSR pairs made by the SA-HH$_{\text{PSF}}$ are more accurate over the SA-HH$_{\text{NO-PSF}}$ which applies the MAR-JSR pairs in a periodic sequence without referring to the problem state.

The performance of the schedule can be visualised using a Gantt chart. Therefore, an example of FJSP schedule obtained by SA-HH$_{\text{PSF}}$ for MK06 is presented in Figure 5. The makespan of the schedule is 69 units of time.

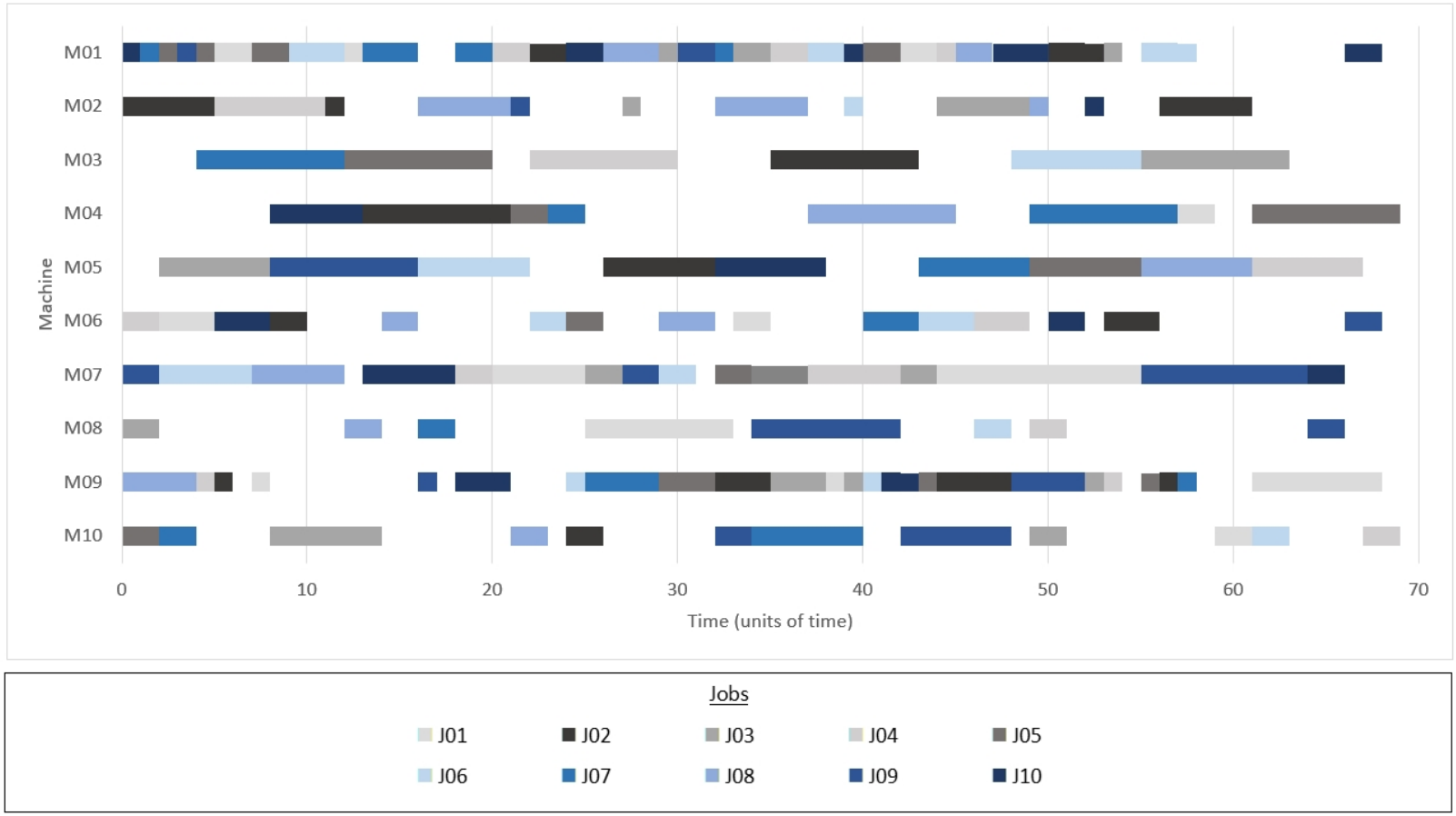


Figure 5. An example of FJSP schedule for MK06.

Table 7 presents the average computational time of SA-HH$_{\text{NO-PSF}}$ and the SA-HH$_{\text{PSF}}$ of 30 executions. The data shows that SA-HH$_{\text{PSF}}$ records shorter average computational time than SA-HH$_{\text{NO-PSF}}$.

Table 7. Average computation time of SA-HH$_{\text{NO-PSF}}$ and SA-HH$_{\text{PSF}}$

Instance	Average computational time (seconds)	
	SA-HH$_{\text{NO-PSF}}$	SA-HH$_{\text{PSF}}$
I_1	58	56
I_2	159	148
I_3	173	154
I_4	425	358
I_5	128	132
I_6	176	168
MK01	220	201
MK02	320	267
MK03	952	776
MK04	376	336
MK05	530	417
MK06	937	705
MK07	720	646
MK08	916	924
MK09	1381	1395
MK10	1452	1432
Average	558	507

4.4. Competitiveness of SA-HH$_{NO\text{-}PSF}$ and SA-HH$_{PSF}$

Experiment II compares the SA-HH$_{\text{NO-PSF}}$, the SA-HH$_{\text{PSF}}$ to several published algorithms as follow:

(1) Tabu Search (TS) (Brandimarte, 1993)
(2) Heuristic Algorithm (HA) (Ziaee, 2014)
(3) Genetic Algorithm (GA) (Chang, Tsai and Liu, 2014)
(4) Multi-agent model based on Chemical Reaction Optimization with Greedy Algorithm (MACROG) (Marzouki, Belkahla Driss and Ghédira, 2017)
(5) Grey Wolf Optimization (GWO) (Jiang and Zhang, 2018)
(6) Self-learning Genetic Algorithm based on Reinforcement Learning (SLGA) (Chen et al., 2020)
(7) Artificial Bee Colony Algorithm (Ferreira et al., 2020)

The SA-HH$_{\text{NO-PSF}}$ and the SA-HH$_{\text{PSF}}$ are executed using a number of fitness evaluation which is less than all the benchmark algorithms. Table 8 details the algorithmic configurations of MACROG, GWO, SLGA and ABC. GWO, SLGA and ABC are also evaluated using adaptive parameters whose values depend on the number of jobs (x), and number of machines (y), defined in the instance.

Table 8. Configurations used by benchmark algorithms

Algorithm	Population Size	No. of Iterations	No. of Fitness Evaluations	No. of Replications
MACROG	1,000	1,000	1,000,000	*Not stated*
GWO	200	$10 \times x \times y$	$200 \times 10 \times x \times y$	10
SLGA	$5 \times x \times y$	$10 \times x \times y$	Population size × No. of iterations	20
ABC	$(3 \times x) + (11 \times x)$	$2 \times x \times y$	Population size × No. of iterations	20

Table 9 compares the best $C_{\max}$ achieved by SA-$HH_{NO\text{-}PSF}$ and SA-HH_{PSF} to the best $C_{\max}$ achieved by the benchmark algorithms evaluated against the dataset by Kacem, Hammadi and Borne (2002a,b). The result shows that the performances of SA-$HH_{NO\text{-}PSF}$ and SA-HH_{PSF} are on par with most of the benchmark algorithms except I_3 and I_4. In I_3, both SA-HH variants are one time unit off the best $C_{\max}$ recorded by HA and GWO, whereas in I_4, both SA-HH variants outperform HA and GWO. I_6 is excluded from this comparison study as the data was not reported in the mentioned papers.

Table 9. Performance comparison in terms of best $C_{\max}$ on the dataset by Kacem, Hammadi and Borne (2002a,b)

Instance	Best $C_{\max}$ (units of time)					
	HA	MACROG	GWO	SLGA	SA-$HH_{NO\text{-}PSF}$	SA-HH_{PSF}
I_1	11	11	11	11	11	11
I_2	13	20	11	11	11	11
I_3	7	19	7	#	8	8
I_4	12	#	13	#	11	11
I_5	15	14	14	14	14	14

Not reported in the literature.

Meanwhile, Table 10 compares the best $C_{\max}$ achieved by SA-$HH_{NO\text{-}PSF}$ and SA-HH_{PSF} to the best $C_{\max}$ achieved by the benchmark algorithms on the dataset by Brandimarte (1993). The result shows that the SA-$HH_{NO\text{-}PSF}$ and the SA-HH_{PSF} outperform TS, HA, GA and MACROG in most instances. The SA-$HH_{NO\text{-}PSF}$ and the SA-HH_{PSF} are also less competitive than GWO, SLGA and ABC due to the relatively high number of fitness evaluation being used in the respective experiments. However, further investigations could be initiated to verify whether both SA-HH variants could achieve similar performances if the same number of fitness evaluation is used during the experiments.

Table 10. Performance comparison in terms of best $C_{\max}$ on the dataset by Brandimarte (1993)

Instance	Best $C_{\max}$ (units of time)								
	TS	HA	GA	MACROG	GWO	SLGA	ABC	SA-$HH_{NO\text{-}PSF}$	SA-HH_{PSF}
MK01	42	42	41	40	40	40	39	40	41
MK02	32	28	28	32	29	27	26	27	27
MK03	211	204	204	204	204	204	204	204	204
MK04	81	75	66	64	64	60	60	67	66
MK05	186	179	178	179	175	172	169	174	174
MK06	86	69	73	85	69	69	58	68	67
MK07	157	149	150	172	147	144	144	147	147
MK08	523	555	523	552	523	523	523	523	523
MK09	369	342	327	421	322	320	308	312	311
MK10	296	242	257	358	249	254	#	222	217

Not reported in the literature.

On the basis of the values of $1/C_{\max}$, the percentile rank of each algorithm is calculated and visualised in Figures 6, 7 and 8 for instances with low, medium and high complexities. I_1, I_5, MK03 and MK08 are excluded from the visuals due to similar performances achieved by almost all the algorithms, whereas I_3 and I_4 are excluded due to limited data.

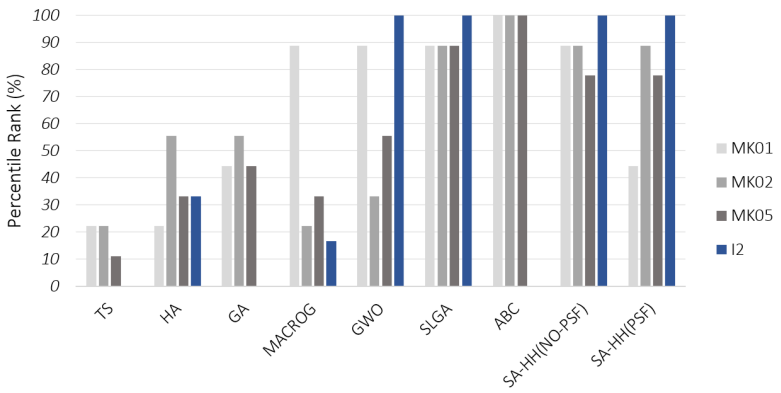


Figure 6. Algorithms' percentile rank on benchmark instances with low complexity

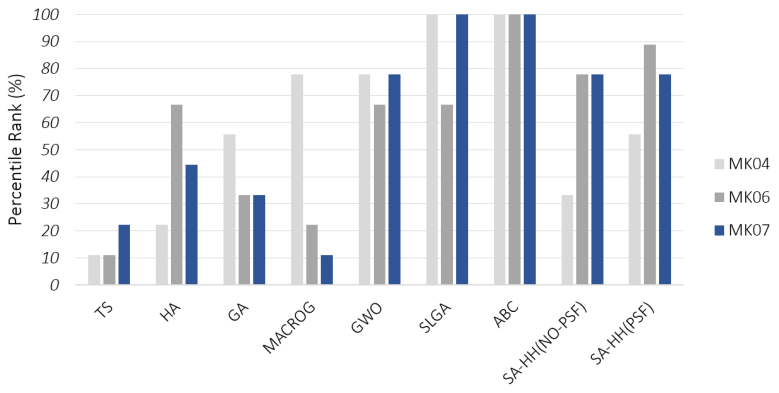
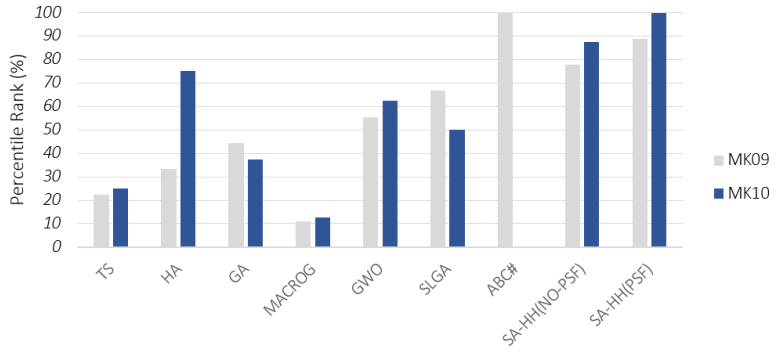


Figure 7. Algorithms' percentile rank on benchmark instances with medium complexity



Data on MK10 for ABC is not reported in the literature.

Figure 8. Algorithms' percentile rank on benchmark instances with high complexity

The results show that the SA-HH$_{\text{NO-PSF}}$ and the SA-HH$_{\text{PSF}}$ outperform or on par with over 50% of the benchmark algorithms except MK04 for the SA-HH$_{\text{NO-PSF}}$ and MK01 for the SA-HH$_{\text{PSF}}$. Figures 6 and 7 demonstrate that majority of the algorithms perform equally, as the instances are of low and medium complexities. As shown in Figure 8, the SA-HH$_{\text{NO-PSF}}$ and the SA-HH$_{\text{PSF}}$ perform better on more complex instances. This shows that the MARs and JSRs considered in this study can handle complex instances while still performing well in simpler instances.

5. Conclusion

The research proposed two variants of SA-HH in selecting an HS (MAR-JSR pairs) for the FJSP. These variants, SA-HH based on the HS with problem state features (SA-HH$_{\text{PSF}}$) and without problem state features (SA-HH$_{\text{NO-PSF}}$), were evaluated using the dataset by Brandimarte (1993) and Kacem, Hammadi and Borne (2002a,b). Results of the sign test show that at the significance level of 0.05, a significant difference is observed on the median of the signed differences on the average $C_{\max}$ achieved by the SA-HH$_{\text{PSF}}$ and the SA-HH$_{\text{NO-PSF}}$ in 30 runs. This observation indicates that the SA-HH$_{\text{PSF}}$ significantly outperforms the SA-HH$_{\text{NO-PSF}}$. Meanwhile, the SA-HH$_{\text{PSF}}$ and the SA-HH$_{\text{NO-PSF}}$ outperform TS, HA, GA and MACROG in more than 50% of the

benchmark dataset in terms of the best $C_{\max}$ achieved in 30 runs. Based on the percentile rank, the SA-HH$_{\text{PSF}}$ is able to outperform or achieve similar performances with more than 75% of the benchmark algorithms on 8 out of 10 instances by Brandimarte (1993) in terms of the best $C_{\max}$. Furthermore, the performance of the SA-HH$_{\text{PSF}}$ and the SA-HH$_{\text{NO-PSF}}$ is better when they are applied on instances with high complexity.

The contributions of this research are threefold. Firstly, the SA-HH by Garza-Santisteban et al. (2019a) which is originally proposed to solve JSP has been successfully extended to solve the FJSP through the incorporation of MARs in HS to solve the additional machine assignment sub-problem in FJSP. Secondly, a refined computation of problem state features has been proposed to tackle the machine assignment sub-problem and proven statistically significant. Thirdly, the success of SA-HH in solving FJSP in this research has indicated its potential of solving similar optimisation problems in the industry. Thus, this research presents an explanatory example for such real-world applications. Future research will focus on three aspects. Firstly, the perturbation strategies could be enhanced to speed up convergence. Secondly, the rewarding system for the heuristic selection could be supported by a mathematical model. Thirdly, further investigations on the algorithm's performance could be made in a dynamic and stochastic flexible job shop environment.

Acknowledgement

The work was supported by the Research University Grant awarded by Universiti Sains Malaysia under Grant No. 1001/PMEKANIK/8014069.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author, L.-P. Wong, upon reasonable request.

References

Ahmed, Leena, Christine Mumford, and Ahmed Kheiri. 2019. "Solving Urban Transit Route Design Problem Using Selection Hyper-heuristics." *European Journal of Operational Research* 274 (2): 545–559.

Basán, Natalia P., Mariana E. Cóccola, Alejandro García del Valle, and Carlos A. Méndez. 2019. "An Effective MILP-Based Decomposition Algorithm for the Scheduling and Redesign of Flexible Job-Shop Plants." *Chemical Engineering Transactions* 74: 613–618.

Bonab, Mohammad Babrdel, Siti Zaiton Mohd Hashim, Tay Yong Haur, and Goh Yong Kheng. 2019. "A New Swarm-Based Simulated Annealing Hyper-Heuristic Algorithm for Clustering Problem." In *Procedia Computer Science* 163, Proceedings of the 16th International Learning and Technology Conference: 228–236.

Brandimarte, Paolo. 1993. "Routing and Scheduling in a Flexible Job Shop by Tabu Search." *Annals of Operations Research* 41 (3): 157–183.

Burke, Edmund K., Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward. 2010. "A Classification of Hyper-heuristic Approaches." In *Handbook of Metaheuristics. International Series in Operations Research & Management Science* 146, 449–468.

Burke, Edmund K., Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, En-

der Özcan, and Rong Qu. 2013. "Hyper-heuristics: A Survey of the State of the Art." *Journal of the Operational Research Society* 64 (12): 1695–1724.

Cao, Yang, Haibo Shi, and DaLiang Chang. 2022. "Differential Evolution Algorithm with Dynamic Multi-population Applied to Flexible Job Shop Schedule." *Engineering Optimization* 54 (3): 387–408.

Chang, Hao-Chin, Hung-Te Tsai, and Tung-Kuan Liu. 2014. "Application of Genetic Algorithm to Optimize Unrelated Parallel Machines of Flexible Job-shop Scheduling Problem." In *Proceedings of the 11th IEEE International Conference on Control and Automation*, 596–599.

Chen, Ronghua, Bo Yang, Shi Li, and Shilong Wang. 2020. "A Self-learning Genetic Algorithm Based on Reinforcement Learning for Flexible Job-shop Scheduling Problem." *Computers and Industrial Engineering* 149.

Choong, Shin Siang, Li-Pei Wong, and Chee Peng Lim. 2018. "Automatic Design of Hyper-heuristic Based on Reinforcement Learning." *Information Sciences* 436-437: 89–107.

Choong, Shin Siang, Li-Pei Wong, and Chee Peng Lim. 2019. "An Artificial Bee Colony Algorithm With a Modified Choice Function for the Traveling Salesman Problem." *Swarm and Evolutionary Computation* 44: 622–635.

Derrac, Joaquín, Salvador García, Daniel Molina, and Francisco Herrera. 2011. "A Practical Tutorial on the Use of Nonparametric Statistical Tests as a Methodology for Comparing Evolutionary and Swarm Intelligence Algorithms." *Swarm and Evolutionary Computation* 1 (1): 3–18.

Drake, John H., Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. 2020. "Recent Advances in Selection Hyper-heuristics." *European Journal of Operational Research* 285 (2): 405–428.

Ferreira, Inês C., Bernardoa Firme, Miguel S. E. Martins, Tiago Coito, Joaquim Viegas, João Figueiredo, Susana M. Vieira, and João M. C. Sousa. 2020. "Artificial Bee Colony Algorithm Applied to Dynamic Flexible Job Shop Problems." *Communications in Computer and Information Science* 1237 CCIS: 241–254.

Garza-Santisteban, Fernando, Jorge M. Cruz-Duarte, Ivan Amaya, José Carlos Ortiz-Bayliss, Santiago Enrique Conant-Pablos, and Hugo Terashima-Marín. 2019a. "Influence of Instance Size on Selection Hyper-Heuristics for Job Shop Scheduling Problems." In *Proceedings of the 2019 IEEE Symposium Series on Computational Intelligence*, 1708–1715.

Garza-Santisteban, Fernando, Roberto Sánchez-Pámanes, Luis Antonio Puente-Rodríguez, Ivan Amaya, José Carlos Ortiz-Bayliss, Santiago Conant-Pablos, and Hugo Terashima-Marin. 2019b. "A Simulated Annealing Hyper-heuristic for Job Shop Scheduling Problems." In *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, 57–64.

Jiang, Tianhua, and Chao Zhang. 2018. "Application of Grey Wolf Optimization for Solving Combinatorial Problems: Job Shop and Flexible Job Shop Scheduling Cases." *IEEE Access* 6: 26231–26240.

Kacem, Imed, Slim Hammadi, and Pierre Borne. 2002. "Pareto-optimality Approach for Flexible Job-shop Scheduling Problems: Hybridization of Evolutionary Algorithms and Fuzzy Logic." *Mathematics and Computers in Simulation* 60(3-5): 245–276.

Kacem, Imed, Slim Hammadi, and Pierre Borne. 2002. "Approach by Localization and Multiobjective Evolutionary Optimization for Flexible Job-shop Scheduling Problems." *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews* 32(1): 1–13.

Kheiri, Ahmed, and Ed Keedwell. 2015. "A Sequence-based Selection Hyper-heuristic Utilising a Hidden Markov Model." In *Proceedings of the 2015 Genetic and Evolutionary Computation Conference*, 417–424.

Kheiri, Ahmed, Ender Özcan, Rhyd Lewis, and Jonathan Thompson. 2016. "A Sequence-based Selection Hyper-heuristic: A Case Study in Nurse Rostering." In *Proceedings of the 11th International Conference on the Practice and Theory of Automated Timetabling*, 503–505.

Kirkpatrick, S., C. D. Gelatt Jr., and M. P. Vecchi. 1983. "Optimization by Simulated Annealing." *Science* 220 (4598): 671–680.

Lin, Jian, Dike Luo, Xiaodong Li, Kaizhou Gao, and Yanan Liu. 2017. "Differential Evolution

based Hyper-heuristic for the Flexible Job-shop Scheduling Problem with Fuzzy Processing Time." In *Lecture Notes in Computer Science* 10593 LNCS, Proceedings of the 11th International Conference on Simulated Evolution and Learning: 75–86.

Lin, Jian. 2019. "Backtracking Search based Hyper-heuristic for the Flexible Job-shop Scheduling Problem with Fuzzy Processing Time." *Engineering Applications of Artificial Intelligence* 77: 186–196.

Luo, Min, Jian Lin, and Li Xu. 2020. "Solving Flexible Job-shop Problem with Sequence-dependent Setup Times by Using Selection Hyper-heuristics." In *ACM International Conference Proceeding Series*, Proceedings of the 2nd International Conference on Artificial Intelligence and Advanced Manufacture, 428–433.

Marzouki, Bilel, Olfa Belkahla Driss, and Khaled Ghédira. 2017. "Multi Agent Model Based on Chemical Reaction Optimization with Greedy algorithm for Flexible Job shop Scheduling Problem." In *Procedia Computer Science* 112, Proceedings of the 21st International Conference on Knowledge - Based and Intelligent Information and Engineering Systems: 81–90.

Mirshekarian, Sadegh, and Dušan N. Šormaz. 2016. "Correlation of Job-shop Scheduling Problem Features with Scheduling Efficiency." *Expert Systems with Applications* 62: 131–147.

Mohan, Jatoth, Krishnanand Lanka, and A. Neelakanteswara Rao. 2011. "A Review of Dynamic Job Shop Scheduling Techniques." In *Procedia Manufacturing* 30, Proceedings of the 14th Global Congress on Manufacturing and Management: 34–39.

Mönch, Lars, John W. Fowler, Stéphane Dauzère-Pérès, Scott J. Mason, and Oliver Rose. 2011. "A Survey of Problems, Solution Techniques, and Future Challenges in Scheduling Semiconductor Manufacturing Operations." *Journal of Scheduling* 14 (6): 583–599.

Nguyen, Su, Mengjie Zhang, and Kay Chen Tan. 2017. "Surrogate-Assisted Genetic Programming with Simplified Models for Automated Design of Dispatching Rules." *IEEE Transactions on Cybernetics* 47 (9): 2951–2965.

Qu, Rong, Nam Pham, Ruibin Bai, and Graham Kendall. 2015. "Hybridising Heuristics Within an Estimation Distribution Algorithm for Examination Timetabling." *Applied Intelligence* 42 (4): 679–693.

Sabar, Nasser R., Masri Ayob, Graham Kendall, and Rong Qu. 2013. "Grammatical Evolution Hyper-heuristic for Combinatorial Optimization Problems." *IEEE Transactions on Evolutionary Computation* 17 (6): 840–861.

Sharma, Pankaj, and Ajain Jain. 2016. "A Review on Job Shop Scheduling with Setup Times." *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 230 (3): 517–533.

Soto, Carlos, Bernabé Dorronsoro, Héctor Fraire, Laura Cruz-Reyes, Claudia Gomez-Santillan, and Nelson Rangel. 2020. "Solving the Multi-objective Flexible Job Shop Scheduling Problem with a Novel Parallel Branch and Bound Algorithm." *Swarm and Evolutionary Computation* 53: 1–16.

Zhang, Hongliang, Gongjie Xu, Ruilin Pan and Haijiang Ge. 2021. "A Novel Heuristic Method for the Energy-efficient Flexible Job-shop Scheduling Problem with Sequence-dependent Set-up and Transportation Time." *Engineering Optimization* 1–22.

Zhou, Yong, and Jian-Jun Yang. 2019. "Automatic Design of Scheduling Policies for Dynamic Flexible Job Shop Scheduling by Multi-objective Genetic Programming Based Hyper-heuristic." In *Procedia CIRP* 79, Proceedings of the 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering: 439–444.

Zhou, Yong, Jian-Jun Yang, and Lian-Yu Zheng. 2019a. "Hyper-Heuristic Coevolution of Machine Assignment and Job Sequencing Rules for Multi-Objective Dynamic Flexible Job Shop Scheduling." *IEEE Access* 7: 68–88.

Zhou, Yong, Jian-Jun Yang, and Lian-Yu Zheng. 2019b. "Multi-agent Based Hyper-heuristics for Multi-objective Flexible Job Shop Scheduling: A Case Study in an Aero-engine Blade Manufacturing Plant." *IEEE Access* 7: 21147–21176.

Ziaee, Mohsen. 2014. "A Heuristic Algorithm for Solving Flexible Job Shop Scheduling Problem." *International Journal of Advanced Manufacturing Technology* 71 (1-4): 519–528.