Int. J. Advance Soft Compu. Appl, Vol. 8, No. 2, July 2016
ISSN 2074-8523

A Modified Bee Colony Optimization with Local Search Approach for Job Shop Scheduling Problems Relevant to Bottleneck Machines

Wai Mun Choo[1], Li-Pei Wong[1], Ahamad Tajudin Khader[1]

[1]School of Computer Sciences,
Universiti Sains Malaysia, Pulau Pinang, Malaysia
e-mail: cwm13_sk043@student.usm.my, lpwong@usm.my, tajudin@usm.my

Abstract

The job shop scheduling problem (JSSP) is regarded as one of the problems which is widely studied. This paper presents a bee colony optimization (BCO) algorithm with a local search named the modified two-enhancement scheme with neighbourhood N5 perturbation (BCO+mTESN5) to solve the JSSP. In previous research, the BCO algorithm with TESN5 local search (BCO+TESN5) is applied to solve the JSSP. The BCO+TESN5 algorithm uses a brute force strategy in performing the N5 neighbourhood local search. However, the brute force local search strategy incurs expensive overhead. To address the high overhead issue, the BCO+mTESN5 algorithm is proposed where the local search is performed on a targeted bottleneck machine within the N5 neighbourhood structure. The selection of the targeted bottleneck machine is done based on a list of bottleneck machines identified by the shifting bottleneck heuristic (SBH). Two selection strategies are tested to select the targeted bottleneck machine, namely: the greedy selection and the linear ranking selection. The proposed algorithms are examined using a set of benchmark problems obtained from the OR-library. The results show that the proposed BCO+mTESN5 with linear ranking selection successfully solves 54% of the 82 OR-library benchmark dataset to $\leq$ 1% from known optimum and it is comparable to the BCO+TESN5 algorithm. In terms of computational time to obtain the best makespan, the proposed BCO+mTESN5 with linear ranking selection outperforms the BCO+TESN5 algorithm by 25%.

Keywords: *Bee algorithm, Local search, Neighbourhood search, Selection strategy, Shifting bottleneck heuristic.*

1 Introduction

Rapid growth of global market makes the manufacturing industry a complex yet dynamic industry. In order to compete in a versatile environment, manufacturers are attempting very hard to increase productivity and minimize the cycle time of their products. One of the common issues among the manufacturer is to ensure that scarce resources are allocated effectively to a set of competing activities. This is crucial as it maximizes the utilization of the resources (e.g. machines, human work force) and thus the productivity can be improved.

Job shop scheduling problem (JSSP) is commonly found in manufacturing industry. Over the last 50 years, there is a considerable amount of literature on JSSP has been introduced [1]. For instance, exact methods that guarantee the optimum solution have been extensively applied to address the JSSP. These exact methods include branch-and-bound [2; 3], and linear programming [4]. However, exact methods are found to be expensive as they need exponential computational time for solving large scale scheduling problems. Another group of techniques used to solve the JSSP is approximate algorithms. An approximate algorithm employs heuristic and/or iterative improvements during the problem solving process. Priority dispatching rules are categorized as approximate algorithms. Priority dispatching rules are frequently applied to solve the JSSP because of their ease of implementation and low time complexity. The priority dispatching rules can be based on due dates or processing time such as first in first out (FIFO), earliest due date (EDD), and shortest processing time (SPT). Priority dispatching rules perform well in certain cases but there is no guarantee that the rules are able to generate the optimum solution for all problems [5].

The need for more robust methodologies which can produce reasonably good solutions within short period of time leads to the development of a new family of approximate algorithms named meta-heuristic. According to Osman and Laporte, a meta-heuristic is an iterative generation process which guides a subordinate heuristic by combining perceptively different concepts for exploring and exploiting the search space using learning strategies to structure information in order to find efficient near-optimal solutions [6]. Meta-heuristic are also considered as high level strategies for exploring search space such that the search is able to avoid from being trapped in local minima. [5; 7] introduce some meta-heuristics which are commonly used to solve job shop scheduling problems. These include genetic algorithm (GA), tabu search (TS), simulated annealing (SA), particle swarm intelligence (PSO), ant colony optimization (ACO), bee colony optimization (BCO).

The PSO, ACO, and BCO algorithms are nature inspired meta-heuristics which model the behavior of different swarms of animals and insects. These three algorithms (i.e. PSO, ACO, and BCO) were developed based on bird flocking, ant colony, and honey bees' behavior respectively. The nature inspired meta-heuristics have been studied by researcher and they have been applied to solve real-world problems in different domains including JSSP. In order to enhance the performance of the algorithms, these meta-heuristics are hybridized with other techniques (e.g. local search methods or other meta-heuristics). Table 1 shows some of the recent methods which are employed to solve JSSP. Many of the methods listed in Table 1 are hybridized with other approaches, to make the methods more robust.

Table 1: Methods used to solve job shop scheduling problem

Method	Year	Title
Genetic Algorithm (GA)	2012	A new hybrid genetic algorithm for job shop scheduling problem [8]
	2014	A new hybrid parallel genetic algorithm for the job-shop scheduling problem [9]
	2015	A new hybrid island model genetic algorithm for job shop scheduling problem [10]
	2015	A local search genetic algorithm for the job shop scheduling problem with intelligent agents [11]
Tabu Search (TS)	2012	A hybrid genetic tabu search algorithm for solving job shop scheduling problems: A case study [12]
	2015	A tabu search algorithm to minimize total weighted tardiness for the job shop scheduling problem [13]
	2015	A tabu search algorithm to minimize total weighted tardiness for the job shop scheduling problem [14]
Simulated Annealing (SA)	2011	A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardiness objective [15]
	2012	Improved simulated annealing algorithm used for job shop scheduling problems [16]
	2015	Neighbourhood generation mechanism applied in simulated annealing to job shop scheduling problems [17]
Particle Swarm Optimization (PSO)	2011	An efficient hybrid particle swarm optimization for the job shop scheduling problem [18]
	2012	A two-stage hybrid particle swarm optimization algorithm for the stochastic job shop scheduling problem [19]
	2013	A neighbourhood property for the job shop scheduling problem with application to hybrid particle swarm optimization [20]
Ant Colony	2006	A hybrid ant colony optimization technique for job-shop

Optimization (ACO)		scheduling problems [21]
	2008	On the pheromone update rules of ant colony optimization approaches for the job shop scheduling problem [22]
	2010	Ant colony optimisation with parameterised search space for the job shop scheduling problem [23]
	2013	An ant colony optimization algorithm for job shop scheduling problem [24]
Bee Colony Optimization (BCO)	2006	A bee colony optimization algorithm to job shop scheduling [25]
	2010	Bee colony optimisation algorithm with big valley landscape exploitation for job shop scheduling problems [26]
	2012	A generic bee colony optimization framework for combinatorial optimization problems [27]

The BCO algorithm is one of the potential algorithms to solve the JSSP. Chong et al. proposed a BCO algorithm specifically designed for the JSSP [25]. The BCO algorithm described in [25] was compared with the ACO algorithm, and the results shows that the proposed algorithm outperforms the ACO algorithm in terms of accuracy and speed. Wong proposed the BCO algorithm with TESN5 local search (BCO+TESN5) to solve the JSSP [27], and the results shows that the BCO+TESN5 algorithm is comparable with PSO algorithm. One of the limitations of the BCO+TESN5 algorithm is that it employs a brute force strategy in performing the N5 neighbourhood local search. The details of the N5 neighbourhood structure can be found in Section 4. This causes the TESN5 local search is expensive in terms of processing overhead. This paper intends to reduce the high overhead of the TESN5 by performing the local search on a targeted bottleneck machine only. It describes the work on applying the BCO algorithm with the modified two-enhancement scheme with neighbourhood N5 perturbation (mTESN5) to the JSSP.

The organization of this paper is as follows. Section 1 gives introductory information about this work. Section 2 describes the JSSP. Section 3 highlights several bee related algorithms. Section 4 presents the N5 neighbourhood structure. Section 5 presents the proposed work, i.e. the integration of the modified TESN5 in the BCO algorithm to solve the JSSP. Section 6 presents the experimental results. Section 7 concludes the paper.

2 Job Shop Scheduling Problem

In this section, the JSSP definition, its modelling, and optimization metrics are described. Let's consider a JSSP which consists of n jobs, $J = (1, 2, \ldots, n)$. These jobs will be processed on a set of *m* machines $M = (1, 2, \dots, m)$. Each job consists of several operations and each operation will have to be processed on different machines according to a pre-defined precedence constraint. The operations are dependent to precedence operation in each job. This set of operations is denoted by $O = \{0, O_{11}, \dots, O_{1m}, O_{n1}, \dots, O_{im}, *\}$ where O_{ij} denotes the *j*-th operation of job J_i. The cardinality of *O* is $(n \text{ x } m) + 2$ where "0" and "*" indicate the fabricated starting and last operations respectively. Once an operation is processed on a machine, it cannot be interrupted until it is finished. A JSSP schedule/solution illustrates the sequence of the operations on the machines.

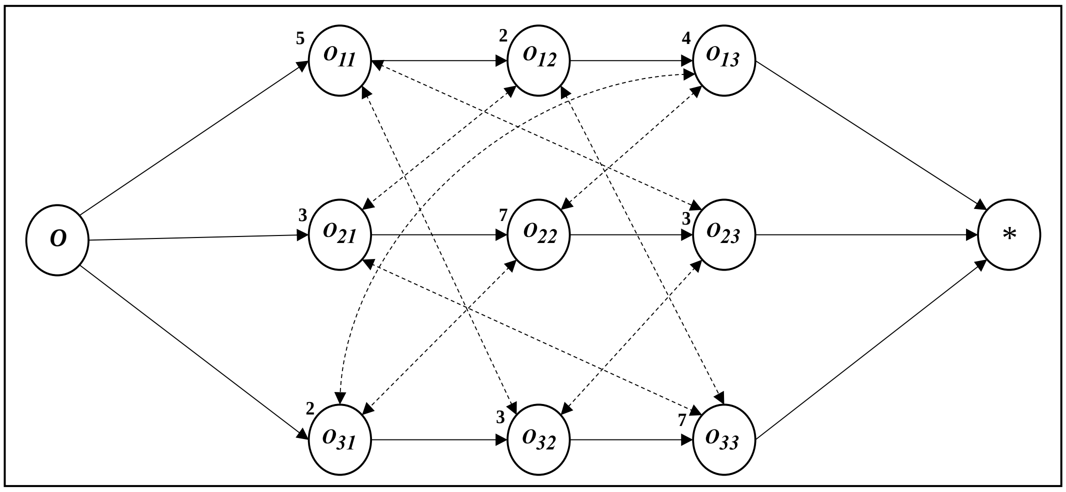
A JSSP can be modelled using a disjunctive graph $G = \{O, E_{Con} \cup E_{Dis}\}$ [28-30]. A set of directed conjunctive arcs which denotes the precedence constraints of each job is represented as E_{Con} and a set of bi-directional disjunctive arcs which denotes the capacity constraints is represented as E_{Dis}.

For example, given a three-job, three-machine JSSP as shown in Table 2, each operation of a particular job is denoted by $x(y)$, where *x* indicates the processing time and *y* indicates the operating machine. Each row is a pre-defined machine precedence order for each job. Let's take job J_1 as an example, the first operation is to be processed on machine 1 (m_1) for five units of time. This is followed by the second operation which is to be processed on machine 2 (m_2) for two units of time. Finally, the third operation of J_1 is to be processed on machine 3 (m_3) for four units of time. By considering jobs J_1, J_2, and J_3, all the operations of these three jobs can be grouped according to machines as follows: $m_1 = \{O_{11}, O_{23}, O_{32}\}$, $m_2 = \{O_{12}, O_{21}, O_{33}\}$, and $m_3 = \{O_{13}, O_{22}, O_{31}\}$ respectively.

The JSSP described in in Table 2 is illustrated as a disjunctive graph as shown in Fig. 1. The vertices in the disjunctive graph (Fig. 1) represent the operations while the arcs represent the given precedence between the operations. Solid directed arcs represent denote the precedence constraints of each job. For example, in order to complete job J_1, O_{11} must be processed first, followed by O_{12}, and finally O_{13}. The dashed bi-directional disjunctive arcs indicate the capacity constraints of each machine. For example, O_{12}, O_{21}, and O_{33} are linked by three bi-directional disjunctive arcs, which indicate these three operations are processed on a same machine, which is machine 2 (m_2).

Table 2: A 3-job x 3-machine JSSP

Job	Operations		
	O_1	O_2	O_3
J_1	5 (1)	2 (2)	4 (3)
J_2	3 (2)	7 (3)	3 (1)
J_3	2 (3)	3 (1)	7 (2)



Legends:

→ Directed conjunctive arc

⇠⇢ Bi-directional disjunctive arc

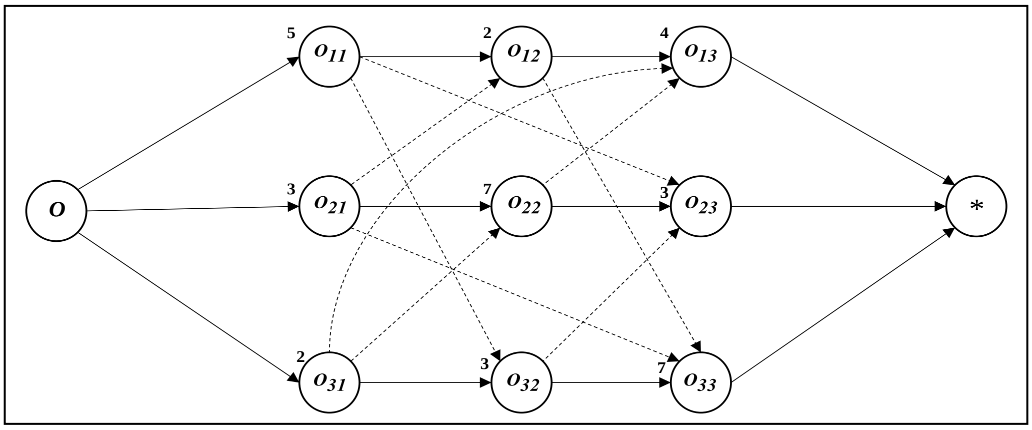
τ O_{ij}

τ – Processing time of O_{ij}

O_{ij} – j-th operation of job i

Fig. 1: The 3-job x 3-machine JSSP in Table 2 represented as a disjunctive graph

In the disjunctive graph model, a feasible JSSP schedule can be obtained by converting the bi-directional disjunctive arcs to become directed arcs such that no cycle exists in the graph. Fig. 2 shows a feasible schedule for the three-job, three-machine JSSP in Table 2, after the arcs conversion for each machine is performed. Take machine 2 as an example, the bi-directional disjunctive arcs are converted into a set of directed disjunctive arcs such that the operation sequencing within machine 2 is as follows: O_{21} is processed first, followed by O_{12}, and finally O_{33}. Note that the disjunctive graph (as shown in Fig. 2) contains no cycle. In another word, by sequencing the operation sequence (i.e. arc conversation from bi-directional disjunctive arc to become directional disjunctive arc) on each machine such that the operation precedence constraint is fulfilled, a feasible JSSP solution is obtained.

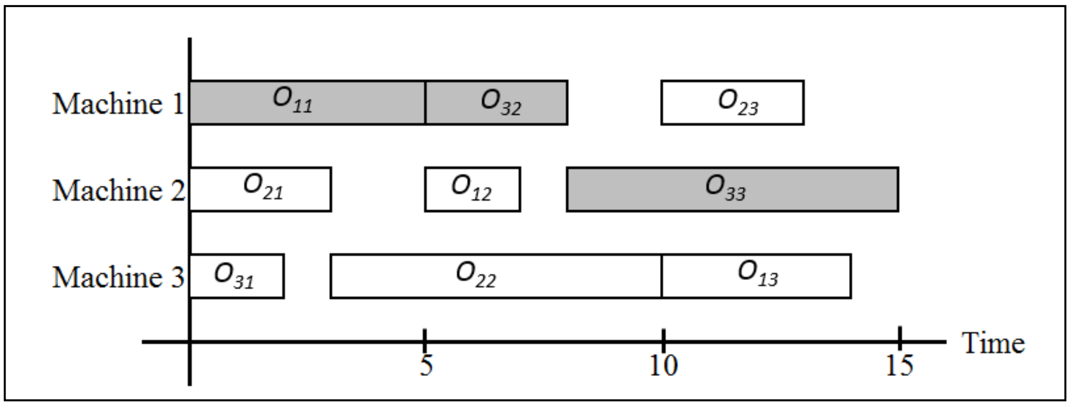


Legends:

⟶	Directed conjunctive arc	τ	τ – Processing time of O_{ij}
- - →	Directed disjunctive arc	O_{ij}	O_{ij} – j-th operation of job i

Fig. 2: A directed graph for the 3-job x 3-machine JSSP in Table 2

Critical path denotes the longest sequence of operations in a JSSP which must be completed on time such that the entire scheduling project to complete on due date (further description of the critical path method is in Section 5.1). The sum of processing time of the critical operations indicates the makespan of the whole schedule. The makespan of a JSSP schedule can be visualized using a Gantt chart. Gantt chart is a general way of graphically presenting a schedule of jobs on machines. The *x*-axis of the chart represents time whereas *y*-axis represents machine. Each job's start and finish time on a particular machine is shown in a Gantt chart. Fig. 3 is a Gantt chart which shows the makespan and the critical path for the directed graph in Fig. 2. From Fig. 3, the critical path is highlighted in grey colour and it can be identified as "$0 \rightarrow O_{11} \rightarrow O_{32} \rightarrow O_{33} \rightarrow *$". The total processing time for the critical path is 0 + 5 + 3 + 7 = 15. This indicates the makespan for this schedule is 15 units of time.



Legend:

O_{ij} – j-th operation of job i ▨ Critical Path

Fig. 3: Gantt chart for the directed graph (feasible schedule) in Fig. 2

A critical path can be decomposed into a set of blocks. Each block contains a set of well-ordered operation that processed on the same machine [31]. Two consecutive blocks must contain operations which are processed on different machines. If the Gantt chart (i.e. Fig. 3) is referred to, the critical path can be decomposed into two consecutive blocks. The first block consists of O_{11} and O_{32}. The second block consists of O_{33}. The block decomposition forms the central idea in the implementation of the N5 neighbourhood operators. A detailed description of the block decomposition and N5 neighbourhood structure will be explained in Section 4.

In a JSSP, there are many different performance metrics which can be optimized. Some examples of the JSSP performance metrics which are commonly employed in optimization are listed in Table 3 [1; 32; 33]:

Table 3: Commonly used JSSP performance metrics for optimization

Objective function	Symbol	Interpretation
Makespan	C_{max}	The total amount of time required to completely process all the jobs.
Mean completion time	$\bar{C}$	The average time spent by a job in the schedule.
Mean flow time	$\bar{F}$	The average time spent by a job in the schedule and including the processing time, waiting time and transfer time.
Maximum lateness	L_{max}	It is defined as the difference between completion time and due date of the job.
Maximum tardiness	T_{max}	The maximum value of lateness of the jobs.
Number of tardy jobs	N_T	The number of jobs that complete after their due date.

3 Bee Colony Optimization

In this section, an introduction to bee behaviour in nature will be presented. The strong self-organization and division system of honey bee swarm behaviour has gained the interest of researchers. One of the behaviours is the foraging behaviour of honey bees. The exchange of information among bees in bee foraging bahaviour forms collective knowledge to be employed by the entire bee colony. Each hive has a dancing area which allows the foragers to inform and recruit the nest mates to the newly found or existing food sources. The communication between bees is carried out by performing waggle dance on the dancing floor. By observing the dances, a bee opts for a dance to follow such that in a long run, more profitable food sources are favored.

Various algorithms inspired by the behaviour of honey bee have been developed. Some known algorithms based on bee swarm intelligence are the bee system (BS) algorithm, the artificial bee colony (ABC) algorithm, the bees algorithm (BA), the marriage bee optimization (MBO) algorithm, and the bee colony optimization (BCO) algorithm.

Lučić and Teodorović proposed the BS algorithm for solving difficult combinatorial optimization problem. In their research, they used the BS algorithm to solve traveling salesman problem (TSP) [34]. The implementation is tested on eight benchmark TSPs and the proposed BS algorithm is enriched with 2-opt and 3-opt heuristics, which act as a local optimizer to further improve the solutions.

The BA was introduced by Pham et al. [35]. This algorithm is a population-based algorithm that mimics the foraging behaviour of honeybee. The BA population agents allocate a larger number of forager and a small number of scouts. The scouts randomly search the solution space and evaluate the fitness or profitability of food source while the foragers search the neighbourhood to look for further fitness improvement. The BA had been applied to train artificial neural networks [35], to schedule jobs for machine [36], and to solve a timetabling problem [37].

Honey bees exhibit many features that can be computationally realized as a problem solving model in an intelligent system. Besides foraging behaviours, other features such as mating, marriage and reproduction of bees are computationally realized as an algorithmic tool to solve real-world problems. Abbass presented an optimization algorithm based on the marriage in honey bees (MBO) [38]. A normal honey bee's colony consists of queen bee, drones and workers. Queens represent the main reproductive individuals in a colony by laying eggs. When this phenomenon is computationally realized to solve an optimization problem, the queen bee represents a solution, and when the queen bee laid egg, the eggs are produced with a series of crossover and mutation processes. If the hatchling is found to be better than the queen, then the hatchling replaces the current queen and removes its sibling's solutions. This process repeats until a pre-defined criterion is met. The MBO had been applied to solve data mining problem as satisfiability problem [39], and partitioning and scheduling problem in codesign [40].

The ABC algorithm is one of the popular swarm intelligence algorithms based on honey bee behaviour. The ABC algorithm was first proposed by Karaboga [41]. This algorithm simulates the foraging behaviour of honey bee. In the ABC algorithm, employed bees, onlooker bees and scouts bees are the three important constructs of the algorithm. In the ABC algorithm, position of food source represents a possible solution to the problem, and the nectar amount of the food sources corresponds to the quality or fitness of the solution. Initially, all food sources are discovered by scout bees, then onlooker bees in the hive select the food sources to exploit. After an onlooker bee selects a food source to go, it becomes employed bee to fly to the food source and returns to the hive with collected nectar. When employed bee is exploiting the exhausted food sources, it will become a scout bee to search for other food sources. The ABC algorithm had been applied to solve numerical problems [41] and to train neural networks [42].

The BCO algorithm is a population-based algorithm proposed by Teodorović and Dell'Orco [43]. The BCO algorithm consists of a population of artificial bees and each of the artificial bees generate one solution to the problem. There are two alternating phases in this algorithm, namely: the forward pass and backward pass. In the forward pass phase, every artificial bee explores the search space with predefined moves. The predefined moves allow the bees to construct or improve the solution to reproduce a new solution. After having a partial solution from the forward pass, the artificial bees will return to the hive and start the backward pass phase. While in the backward pass, artificial bees that return to the hive will perform a waggle dance in the dance floor to inform other bees about the food source information (solution) they gained. Every bee will make a decision with a certain probability either to abandon the created partial solution or to perform a dance such that other bees are recruited towards to the found food source [44; 45].

Wong proposed a generic BCO framework to address different combinatorial optimization problems such as the quadratic assignment problem, traveling salesman problem and JSSP [27]. It computationally realizes the bee foraging behaviour by first initializing a swarm of artificial bees. These bees will be searching for food sources aided by a fragmentation transition rule (i.e. as its path/solution construction mechanism). The fragmentation transition rule is made up of two elements: arc fitness and heuristic distance. Before a bee starts foraging, it will probabilistically observe a waggle dance to follow. This dance becomes a preferred path of the foraging bee. When the bee constructs the path/solution, the node which appeared in preferred path is with higher arc fitness value and therefore it has higher chance to be selected by the bee as the next visiting node. On the other hand, under the influence of the heuristic distance, a bee tends to select the next nearest node from the current node. When this BCO algorithm is used to solve JSSP to optimize the makespan (i.e. fitness function), the two-enhancement scheme with neighbourhood N5 perturbation is also applied to improve the search precision. The proposed algorithm by Wong [27] is able to solve 30 JSSP benchmark instances to optimum, out of the 82 benchmark problems in the OR-Library.

The BCO algorithm that are described in [26; 27; 46] to solve the JSSP have a few distinctions with the BCO algorithm as described in [44; 45]. The most significant difference is that in the BCO algorithm described in [27], the bees are required to explore the search space and only return to the hive after completing a set of path. This allows the bees exchange the information of a set feasible solution rather than partial solution via waggle dance.

In this paper, unless it is stated otherwise, the abbreviation of "BCO algorithm" refers to the BCO algorithm in [27]. Fig. 4 shows the BCO flow proposed in [27].

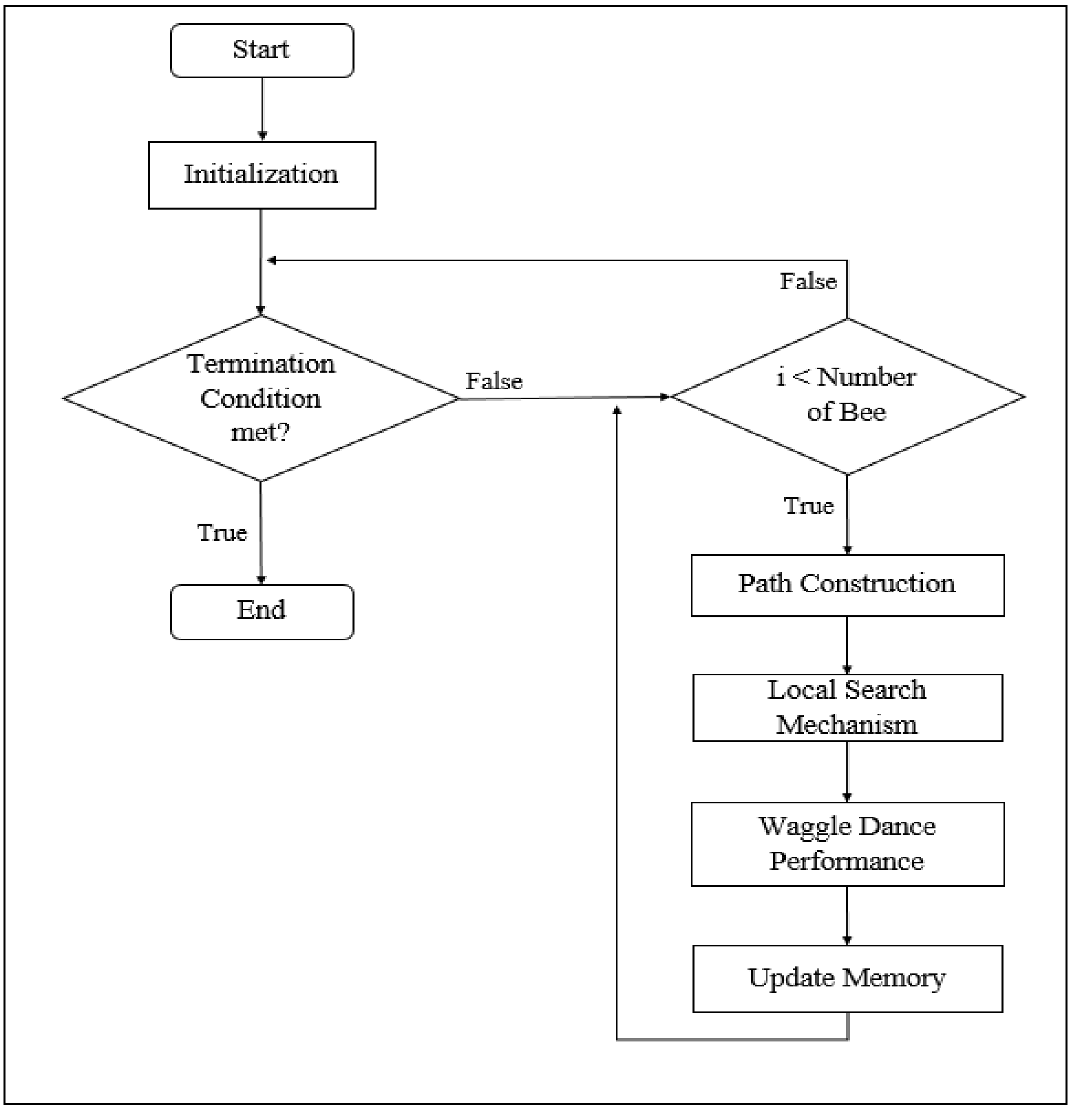


Fig. 4: The flow chart of the BCO algorithm proposed in [27]

4 N5 Neighbourhood Structure

Local search can be integrated with a meta-heuristic method for solving an optimization problem. The local search starts from an initial solution and iteratively improves the solution with a better one in a defined set of neighbourhood solutions by performing a series of neighbourhood moves. If the new solution is with a better quality, it will replace the old solution. These steps continue until no further improvement can be done and at this stage, this solution is named as the local optimum solution within the defined neighbourhood [5].

In the JSSP domain, several neighbourhood structures are defined such as N1, N2, N3, N4, N5, and N6 [47]. Based on a particular neighbourhood structure, local search operator (i.e. the improvement heuristic) can be effectively designed. In this section, the N5 neighbourhood is described as it is employed as the local search approach. The block decomposition described in Section 2 forms the central idea in the implementation of the N5 neighbourhood operators.

Let's consider a feasible JSSP solution which the critical path (i.e. highlighted in grey) is decomposed into four blocks as shown in Fig. 5. The N5 neighbourhood is defined as the interchange of two successive operations in a block structure that belongs to a critical path. For examples, the N5 neighbourhood swapping is performed on the first two operations of the last block or the last two operations of the first block. For intermediate blocks, the swapping can be done by swapping the first two operations or the last two operations of the block.

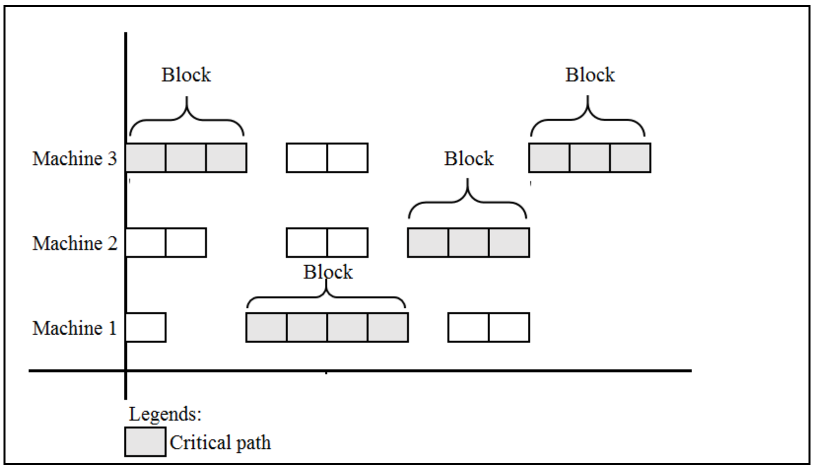


Fig. 5: An example solution illustrated as Gantt chart

Fig. 6 illustrates the possible swapping moves which can be performed according to the example in Fig. 5. There are six possible swapping operations which can be done within the blocks in critical path as shown in Fig. 6. Two possible swapping operations for first and last block of critical path positioned at machine 3, two possible swapping operations on intermediate blocks of critical path positioned at machine 2 and machine 1 respectively. The N5 neighbourhood structure searches all of the neighbourhood space in order to achieve a better solution and it has been successfully applied on solving the scheduling problem with excellent results [31]. However, this local search method is expensive as the number of possible swapping operation increases when the problem size increased.

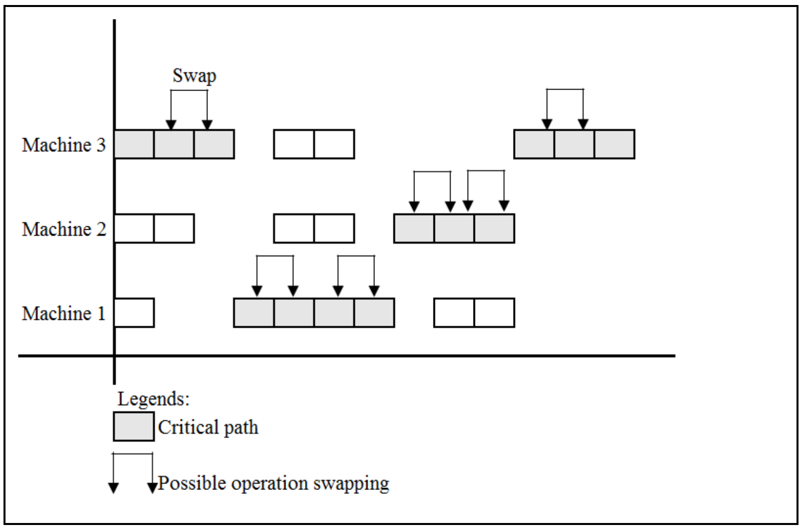


Fig. 6: Possible swapping procedure using N5 neighbourhood structure

5 The Modified Two-Enhancement Scheme with Neighbourhood N5 Perturbation (mTESN5)

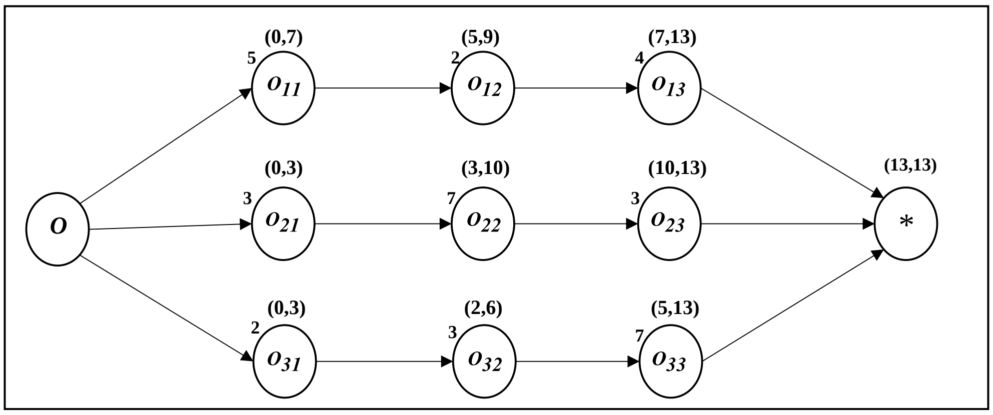
In [27], the BCO algorithm which is integrated with the two-enhancement scheme with neighbourhood N5 perturbation (BCO+TESN5) is applied in solving the JSSP. The TESN5 local search consists of two phases of local search. The first phase involves swapping and insertion operations which are based on the simulated annealing algorithm and the second phase is the minor perturbation based on N5 neighbourhood structure. In the second phase, the N5 neighbourhood structure will attempt to swap all the possible operations of the blocks that are in the critical path according to a brute force manner. This brute force strategy is very time consuming especially when the problem size increases. Instead of performing the perturbation on the entire neighbourhood space, the bottleneck machine is identified such that perturbation in performed on a bottleneck machine in order to reduce the expensive overhead of the TESN5 local search mechanism.

For example, based on Fig. 6, if the brute force strategy is applied, all the six possible swapping moves will be performed before determining which move leads to a better solution. However, if only one bottleneck machine is selected to perform the swapping procedure, only two moves will be performed. This decreases 67% of the required swapping moves in the brute force strategy. The bottleneck machine is identified based on the adaptation of the shifting bottleneck heuristic (SBH). The descriptions of the bottleneck machine concept and SBH are presented in Section 5.1. This local search which focuses on bottleneck machine is named as the modified two-enhancement scheme with neighbourhood N5 Perturbation (mTESN5).

5.1 Bottleneck machines identification

In general, the SBH sequences the machines one at a time, consecutively, by considering the machine identified as a bottleneck among the machines not yet sequenced. Every time after a new machine is sequenced, local re-optimization is performed on all previously established sequences. The SBH decomposes a JSSP into several one-machine scheduling problems. Both the bottleneck identification and the local re-optimization procedures are based on repeatedly solving certain one-machine scheduling problems [1; 48].

This section describes how a list of machines is identified and ranked according to its maximum lateness using the bottleneck identification steps of the SBH. Let's consider the JSSP example shown in Table 2. At the initial state, the operation sequence for each machine is not determined (i.e. only the operation sequences of jobs are considered). The critical path method (CPM) [32] is applied to find the earliest start time and the latest finish time of each operation as shown in Fig. 7.



Legends:

$\longrightarrow$	Directed conjunctive arc
(r_j, d_j)	O_{ij} – j-th operation of job i
τ O_{ij}	τ – Processing time of O_{ij}
	r_j – Earliest start time of O_{ij}
	d_j – Latest finish time of O_{ij}

Fig. 7: Critical path method network graph

Next, each machine is treated as a one-machine scheduling problem. These one-machine problems are solved as $1|\ r_j \ |\ L_{max}$ in order to minimize the maximum lateness, using the earliest due date (EDD) dispatching rule. Based on the CPM network graph shown in Fig. 7, all the three machines (i.e. $m_1 = \{O_{11}, O_{23}, O_{32}\}$, $m_2 = \{O_{12}, O_{21}, O_{33}\}$, and $m_3 = \{O_{13}, O_{22}, O_{31}\}$) will be solved as three different $1|\ r_j \ |\ L_{max}$ problems.

Taking machine 1 as an example, when the EDD dispatching rule is applied, the following sequence is obtained: $O_{11} \rightarrow O_{32} \rightarrow O_{23}$. This sequence is obtained due to the EDD dispatching rule with the ready times is applied. Therefore, the maximum lateness on machine 1 is equal to 2 (as shown in Table 4). If the EDD is applied without considering the ready times, the following sequence is obtained: $O_{32} \rightarrow O_{11} \rightarrow O_{23}$. The maximum lateness for this sequence is equal to 3 (as shown in Table 5). Note that the EDD dispatching rule with the ready times is able to produce a sequence with lower maximum lateness. Thus, it is preferable to apply the EDD dispatching rule with the ready times. Table 4 and Table 5 show the 1| r_j | L_{max} problem and schedule for minimization of L_{max} for machine 1, with the application of different EDD dispatching rules. r_j and d_j are the earliest start time and the latest finish time. S_j is the start time of the operation and C_j is the completion time of the operation according to the job sequence determined by the EDD dispatching rule. L_{max} measures the difference between C_j and d_j (i.e. $L_{max} = C_j - d_j$). The highest L_{max} among the operations is considered as the L_{max} of a particular machine.

Table 4: The 1| r_j | L_{max} problem and schedule for minimization of L_{max} for machine 1 using EDD dispatching rule with ready time consideration

Job (J)	1	3	2
Operation	O_{11}	O_{32}	O_{23}
Processing time, τ	5	3	3
Earliest start time, r_j	0	2	10
Latest finish time, d_j	7	6	13
Start time, S_j	0	5	10
Completion time, C_j	5	8	13
Lateness, L_{max}	-2	2	0

Table 5: The 1| r_j | L_{max} problem and schedule for minimization of L_{max} for machine 1 using EDD dispatching rule without ready time consideration

Job (J)	3	1	2
Operation	O_{32}	O_{11}	O_{23}
Processing time, τ	3	5	3
Release time, r_j	2	0	10
Due date, d_j	6	7	13
Start time, S_j	2	5	10
Completion time, C_j	5	10	13
Lateness, L_{max}	-1	3	0

Once the three 1| r_j | L_{max} problems are solved, each of these is with a maximum lateness (i.e. machines 1, 2, and 3 are with the maximum lateness (L_{max}) of 2, 0, and 0 respectively). These machines are ranked according to their L_{max} value as a tardy machine list. Machine 1 has the highest maximum lateness value, thus machine 1 is ranked at the top position in the tardy machine list. There is a tie between machines 2 and 3. To break the tie, randomly pick a machine (i.e. between machines 2 and 3) and rank it at the second position in the tardy machine list.

5.2 Selection strategy

In the proposed mTESN5 local search algorithm, only one machine will be selected using a selection strategy such that the machine will undergo the local perturbation based on N5 neighbourhood structure. The machine will be selected from a tardy machine list, which the generation of such tardy machine list is described in Section 5.1. Two selection strategies are implemented and tested, namely: greedy selection strategy and linear ranking selection strategy. Thus, this forms two different BCO with mTESN5 algorithms.

If a greedy selection strategy is applied, only the tardiest machine (i.e. the first position) in the tardy machine list is selected to perform the perturbation based on N5 neighbourhood structure. Hence, the local search will only be focused on the tardiest machine, which is the bottleneck machine with the highest maximum lateness value. This algorithm is denoted by “BCO with GS-TESN5 algorithm”.

If a linear ranking selection is applied, each machine in the tardy machine list is linearly assigned with a selection probability according to their rank using a linear function as shown in Equation 1. Hence, the machine with the higher selection probability (i.e. higher maximum lateness value) tends to be selected to undergo the local perturbation. At the same time, other machines in the tardy machine list are having some chance to be selected to undergo the local perturbation. This algorithm is denoted by “BCO with LRS-TESN5 algorithm”.

$$\mathrm{P}(i)=\frac{1}{N}\times\left(SP\text{-}2(SP\text{-}1)\times\frac{i\text{-}1}{N\text{-}1}\right) \tag{1}$$

Equation 1 is a linear function where $i \in \{1, \ldots, N\}$ represents a machine’s rank in this selection strategy. All machines are ranked according to their L_{max} value, i.e., rank 1 is assigned to the machine with highest L_{max} and rank N is assigned to the machine with lowest L_{max} value. Parameter SP (i.e. $1 \leq SP \leq 2$) is used to control the gradient of the linear selection function. A larger SP value results in a higher selection pressure for selecting the solution with the highest rank [49].

A parameter tuning experiment is performed to find a suitable SP value for the linear ranking selection strategy. Four SP values are examined in this parameter tuning experiment, namely: 1.0, 1.1, 1.5, and 2.0. SP = 1.0 represents uniform selection pressure, while SP= 1.1, SP = 1.5 and SP = 2.0 represent low, medium and high selection pressure respectively. 82 JSSP benchmark problems with five replications are used in the parameter tuning experiment. Table 6 shows the results of the linear ranking selection with different SP values in the parameter tuning experiment. M denotes deviation percentage from the best known makespan and μ_T denotes the average computational time to obtain the best makepsan, μ_T.

Table 6: Results of linear ranking selection with different selection pressures

SP Values	1.0	1.1	1.5	2.0
Average M (%)	3.18	3.21	3.21	3.19
Average μ_T (s)	343.85	301.88	298.50	254.92

From Table 6, the average deviation percentage from the best known makespan is similar, for all the four SP values. However, the computational time, μ_T of experiment with SP = 2.0 is the shortest. Therefore, linear function with high selection pressure (i.e. SP = 2.0) is employed in the BCO with LRS-TESN5 algorithm.

6 Experimental Results

The results of the proposed algorithms (i.e. the BCO with GS-TESN5 algorithm and the BCO with LRS-TESN5 algorithm) are presented in this section. The proposed algorithms are implemented using JAVA with NetBean IDE 8.0 as the development tool. The experiments were conducted on a Windows OS cluster with Intel (R) Core™ i7-4700HQ Processor, and 16GB RAM.

The results based on the BCO model in [27] is also included to compare the effectiveness of the proposed algorithm, and it is denoted by BCO+TESN5. The parameter setting used in [27] is adapted in the experiments conducted in this research as follows: β = 10, λ = 0.9, K = 100, ϖ = 25, N_{Bee} = 25, and BC_{Max} = 10000. The BCO with LRS-TESN5 algorithm uses a linear function which a parameter named SP has to be empirically tuned. Via a different parameter tuning experiment as presented in Section 5.2, the SP value is set at 2.0, in order to impose higher selection pressure towards the tardiest machine such that the perturbation is performed on it.

The proposed algorithms are tested on the 82 benchmark instances obtained from the OR-library. These 82 benchmark problems are categorized into six different series (i.e. ABZ, ORB, FT/MT, LA, SWV, and YN). The dimension of these 82 benchmark problems range from 6-job x 6-machine up to 50-job x 10-machine. Five replications of experiment are conducted for each benchmark instance to obtain average results.

Table 7 shows the results of the BCO+TESN5 algorithm, the BCO with GS-TESN5 algorithm, and the BCO with LRS-TESN5 algorithm respectively, in terms of the best makespan and its deviation percentage from the known best optimum, for the 82 benchmark instances obtained from the OR-library. The first, second, and the third columns denote the problem instance name, problem dimension (n-job x m-machine), and the best known optimum. The rest of the columns denote the best makespan and its deviation percentage from the known best optimum, based on five replications of algorithm execution.

Table 7: Performance of the BCO+TESN5, BCO with GS-TESN5, and BCO with LRS-TESN5 algorithms (in terms of the best makespan and its deviation percentage from the known best optimum)

Problem	Problem Dimension	Known Optimum	BCO+TESN5 algorithm		BCO with GS-TESN5 algorithm		BCO with LRS-TESN5 algorithm	
			Best Makespan	*B* (%)	Best Makespan	*B* (%)	Best Makespan	*B* (%)
ABZ5	10 x 10	1234	1234	0.00	1234	0.00	1234	0.16
ABZ6	10 x 10	943	943	0.00	943	0.00	943	0.00
ABZ7	20 x 15	656	694	5.79	695	5.95	690	5.18
ABZ8	20 x 15	665	710	6.77	710	6.77	715	7.52
ABZ9	20 x 15	679	723	6.48	733	7.95	729	7.36
FTP06	6 x 6	55	55	0.00	55	0.00	55	0.00
FTP10	10 x 10	930	937	0.75	937	0.75	937	0.75
FTP20	20 x 5	1165	1165	0.00	1173	0.69	1173	0.69
ORB01	10 x 10	1059	1059	0.00	1059	0.00	1059	0.00
ORB02	10 x 10	888	889	0.11	889	0.11	889	0.11
ORB03	10 x 10	1005	1008	0.30	1005	0.00	1021	1.59
ORB04	10 x 10	1005	1011	0.60	1005	0.00	1011	0.60
ORB05	10 x 10	887	889	0.23	889	0.23	889	0.23
ORB06	10 x 10	1010	1012	0.20	1013	0.30	1019	0.89
ORB07	10 x 10	397	397	0.00	397	0.00	397	0.00
ORB08	10 x 10	899	899	0.00	899	0.00	908	1.00
ORB09	10 x 10	934	934	0.00	934	0.00	934	0.00
ORB10	10 x 10	944	944	0.00	944	0.00	944	0.00
LA01	10 x 5	666	666	0.00	666	0.00	666	0.00
LA02	10 x 5	655	655	0.00	655	0.00	655	0.00
LA03	10 x 5	597	597	0.00	597	0.00	597	0.00
LA04	10 x 5	590	590	0.00	590	0.00	590	0.00
LA05	10 x 5	593	593	0.00	593	0.00	593	0.00
LA06	15 x 5	926	926	0.00	926	0.00	926	0.00
LA07	15 x 5	890	890	0.00	890	0.00	890	0.00
LA08	15 x 5	863	863	0.00	863	0.00	863	0.00
LA09	15 x 5	951	951	0.00	951	0.00	951	0.00
LA10	15 x 5	958	958	0.00	958	0.00	958	0.00
LA11	20 x 5	1222	1222	0.00	1222	0.00	1222	0.00
LA12	20 x 5	1039	1039	0.00	1039	0.00	1039	0.00
LA13	20 x 5	1150	1150	0.00	1150	0.00	1150	0.00
LA14	20 x 5	1292	1292	0.00	1292	0.00	1292	0.00
LA15	20 x 5	1207	1207	0.00	1207	0.00	1207	0.00
LA16	10 x 10	945	945	0.00	945	0.00	945	0.00
LA17	10 x 10	784	784	0.00	784	0.00	784	0.00
LA18	10 x 10	848	848	0.00	848	0.00	848	0.00

Table 7 (continued)

Problem	Problem Dimension	Known Optimum	BCO+TESN5 algorithm		BCO with GS-TESN5 algorithm		BCO with LRS-TESN5 algorithm	
			Best Makespan	*B* (%)	Best Makespan	*B* (%)	Best Makespan	*B* (%)
LA19	10 x 10	842	842	0.00	849	0.83	842	0.00
LA20	10 x 10	902	902	0.00	907	0.55	907	0.55
LA21	15 x 10	1046	1067	2.01	1055	0.86	1059	1.24
LA22	15 x 10	927	930	0.32	935	0.86	935	0.86
LA23	15 x 10	1032	1032	0.00	1032	0.00	1032	0.00
LA24	15 x 10	935	949	1.50	956	2.25	949	1.50
LA25	15 x 10	977	993	1.64	991	1.43	986	0.92
LA26	20 x 10	1218	1218	0.00	1218	0.00	1218	0.00
LA27	20 x 10	1235	1264	2.35	1269	2.75	1269	2.75
LA28	20 x 10	1216	1226	0.82	1241	2.06	1234	1.48
LA29	20 x 10	1152	1208	4.86	1207	4.77	1214	5.38
LA30	20 x 10	1355	1355	0.00	1355	0.00	1355	0.00
LA31	30 x 10	1784	1784	0.00	1784	0.00	1784	0.00
LA32	30 x 10	1850	1850	0.00	1850	0.00	1850	0.00
LA33	30 x 10	1719	1719	0.00	1719	0.00	1719	0.00
LA34	30 x 10	1721	1721	0.00	1721	0.00	1721	0.00
LA35	30 x 10	1888	1888	0.00	1888	0.00	1888	0.00
LA36	15 x 15	1268	1297	2.29	1291	1.81	1297	2.29
LA37	15 x 15	1397	1418	1.50	1426	2.08	1425	2.00
LA38	15 x 15	1196	1244	4.01	1254	4.85	1230	2.84
LA39	15 x 15	1233	1251	1.46	1264	2.51	1259	2.11
LA40	15 x 15	1222	1244	1.80	1254	2.62	1246	1.96
SWV01	20 x 10	1407	1497	6.40	1502	6.75	1502	6.75
SWV02	20 x 10	1475	1562	5.90	1558	5.63	1565	6.10
SWV03	20 x 10	1398	1537	9.94	1505	7.65	1515	8.37
SWV04	20 x 10	1470	1597	8.64	1609	9.46	1606	9.25
SWV05	20 x 10	1424	1540	8.15	1586	11.38	1577	10.74
SWV06	20 x 15	1678	1832	9.18	1842	9.77	1843	9.83
SWV07	20 x 15	1600	1722	7.63	1739	8.69	1735	8.44
SWV08	20 x 15	1756	1946	10.82	1955	11.33	1970	12.19
SWV09	20 x 15	1661	1818	9.45	1860	11.98	1866	12.34
SWV10	20 x 15	1754	1904	8.55	1937	10.43	1922	9.58
SWV11	50 x 10	2991	3375	12.84	3399	13.64	3371	12.70
SWV12	50 x 10	3003	3333	10.99	3356	11.75	3375	12.39
SWV13	50 x 10	3104	3463	11.57	3491	12.47	3481	12.15
SWV14	50 x 10	2968	3257	9.74	3271	10.21	3259	9.80
SWV15	50 x 10	2904	3208	10.47	3222	10.95	3216	10.74
SWV16	50 x 10	2924	2924	0.00	2924	0.00	2924	0.00
SWV17	50 x 10	2794	2794	0.00	2794	0.00	2794	0.00

Table 7 (continued)

Problem	Problem Dimension	Known Optimum	BCO+TESN5 algorithm		BCO with GS-TESN5 algorithm		BCO with LRS-TESN5 algorithm	
			Best Makespan	*B* (%)	Best Makespan	*B* (%)	Best Makespan	*B* (%)
SWV18	50 x 10	2852	2852	0.00	2852	0.00	2852	0.00
SWV19	50 x 10	2843	2843	0.00	2843	0.00	2843	0.00
SWV20	50 x 10	2823	2823	0.00	2823	0.00	2823	0.00
YN1	20 x 20	885	930	5.08	934	5.54	934	5.54
YN2	20 x 20	909	946	4.07	965	6.16	968	6.49
YN3	20 x 20	892	941	5.49	951	6.61	942	5.61
YN4	20 x 20	968	1054	8.88	1034	6.82	1047	8.16

Table 8 shows the results on the relative performance of makespan in terms of percentage for the algorithms. The results show the average, best, and worst percentage difference from the best known makespan and the average computational time to obtain the best result. They are denoted by *M*, *B*, *W*, and μ_T respectively.

Table 8: Overall performance of BCO+TESN5, BCO with GS-TESN5, and BCO with LRS-TESN5 algorithms

	BCO+TESN5 algorithm	BCO+mTESN5 algorithms	
		BCO with GS-TESN5 algorithm	BCO with LRS-TESN5 algorithm
Average *B* (%)	2.56	2.81	2.79
Average *M* (%)	3.03	3.24	3.19
Average *W* (%)	3.46	3.64	3.56
Average μ_T (s)	340.77	307.37	257.38
Number of instances that are solved to optimum	30	29	30
Number of instances that are solved to ≤ 1% from known optimum/ upper bound	46	45	44

The BCO+TESN5 algorithm is able to solves 30 problem instances to optimum [27]. This is equivalent to 37% of the 82 problem instances are solved to optimum. About 56% of the 82 problem instances, which is equivalent to 46 problem instances, are solved to <=1% deviation from known optimum. The averages of B, M, and W for the BCO+TESN5 algorithm are 2.56%, 3.03%, and 3.46% respectively (as listed in Table 8).

The BCO with GS-TESN5 algorithm solves 29 problem instances to optimum. This is equivalent to 35% of the 82 problem instances are solved to optimum. About 56% of the 82 problem instances, which is equivalent to 46 problem instances, are solved to <=1% deviation from known optimum. On the other hand, the BCO with LRS-TESN5 solves 30 problem instances to optimum. This is equivalent to 37% of the 82 problem instances are solved to optimum. About 54%

of the 82 problem instances, which is equivalent to 44 problem instances, are solved to <=1% deviation from known optimum.

From Table 8, it is noted that the average *M* of the BCO with GS-TESN5 and BCO with LRS-TESN5 is close to the BCO+TESN5 algorithm proposed in [27] with 0.21% and 0.05% difference respectively. On the other hand, the average μ_T for BCO with GS-TESN5 algorithm is 307.37s. The BCO with GS-TESN5 algorithm reduces the average μ_T for roughly 33.4s, which is approximately 10% compared with the BCO+TESN5 algorithm. For the BCO with LRS-TESN5 algorithm, the average μ_T is 257.38s, which is 83.39s different from the BCO+TESN5 algorithm. The BCO with LRS-TESN5 algorithm successfully reduces approximately 25% of the μ_T compared with the BCO+TESN5 algorithm.

Next, the results are further analyzed according to six different problem series (i.e. ABZ, ORB, FT/MT, LA, SWV, and YN). In general, problem instances in FTP, ORB, and LA series are considered as easy problems whereas problem instances in ABZ, SWV, and YN series are considered as hard problems. Table 9 compares the performance in term of solution accuracy and μ_T of the three algorithms according to this series.

Table 9: Performance of BCO+TESN5, BCO with GS-TESN5, and BCO with LRS-TESN5 algorithms by series

Series	Instances	BCO+TESN5 algorithm		BCO with GS-TESN5 algorithm		BCO with LRS-TESN5 algorithm	
		Average of		Average of		Average of	
		M (%)	μ_T (s)	*M* (%)	μ_T (s)	*M* (%)	μ_T (s)
ABZ	5	4.58	411.52	4.85	308.65	4.66	262.48
FTP	3	0.61	125.69	0.84	140.22	1.02	64.71
ORB	10	0.56	112.45	0.59	117.51	0.69	76.25
LA	40	0.84	128.30	0.96	126.64	0.88	102.19
SWV	20	7.81	782.07	8.31	711.58	8.20	609.26
YN	4	7.16	902.66	7.19	691.96	7.34	590.46

In terms of the solution accuracy, the averages *M* for all the three algorithms are within the same range. In terms of computational time, all the three algorithms require lesser μ_T to solve the problems from the FTP, ORB, and LA series. In contrary, ABZ, SWV, and YN series require longer μ_T to obtain the best solution.

Table 9 shows that the BCO+TESN5 algorithm uses 782.07s on average to solve the SWV problems, while BCO with GS-TESN5 and BCO with LRS-TESN5 uses approximate 9.01% and 22.1% lesser time to solve the SWV series problem respectively. For YN problem series, the BCO+TESN5 algorithm uses 902.66s on average to solve the problems. On the other hand, BCO with GS-TESN5 and BCO with LRS-TESN5 use approximate 23.34% and 34.59% lesser time to solve the problems of YN series respectively. All these show that the BCO with GS-TESN5 and BCO with LRS-TESN5 algorithms are able to shorten the average μ_T to solve the hard benchmark problem series (i.e. ABZ, SWV and YN problem series).

A comparison study which involves the BCO with GS-TESN5, BCO with LRS-TESN5, ACO algorithm [24] and Tabu Search [31] is presented in Table 10.

Based on the 40 problems in the LA series, the comparison results show that both of the BCO with GS-TESN5, BCO with LRS-TESN5 algorithms outperform the ACO algorithm but underperform the Tabu Search algorithm. On average, the BCO with GS-TESN5 and BCO with LRS-TESN5 algorithms solve the 40 instances to 0.76% and 0.61% from the known optimum respectively. The ACO and Tabu Search algorithms solve the 40 instances to an average of 3.96% and 0.43% from the known optimum respectively.

Table 10: Comparison study which involves the algorithms of BCO with GS-TESN5, BCO with LRS-TESN5, ACO [24] and Tabu Search [31]

Problem	Known Optimum	ACO [24]		Tabu Search [31]		BCO with GS-TESN5		BCO with LRS-TESN5	
		Best Makespan	*B (%)*	*Best Makespan*	*B (%)*	*Best Makespan*	*B (%)*	*Best Makespan*	*B (%)*
LA01	666	666	0.00	666	0.00	666	0.00	666	0.00
LA02	655	669	2.13	655	0.00	655	0.00	655	0.00
LA03	597	623	4.36	597	0.00	597	0.00	597	0.00
LA04	590	611	3.56	593	0.51	590	0.00	590	0.00
LA05	593	593	0.00	593	0.00	593	0.00	593	0.00
LA06	926	926	0.00	926	0.00	926	0.00	926	0.00
LA07	890	890	0.00	892	0.22	890	0.00	890	0.00
LA08	863	863	0.00	863	0.00	863	0.00	863	0.00
LA09	951	951	0.00	951	0.00	951	0.00	951	0.00
LA10	958	958	0.00	958	0.00	958	0.00	958	0.00
LA11	1222	1222	0.00	1222	0.00	1222	0.00	1222	0.00
LA12	1039	1039	0.00	1039	0.00	1039	0.00	1039	0.00
LA13	1150	1150	0.00	1150	0.00	1150	0.00	1150	0.00
LA14	1292	1292	0.00	1292	0.00	1292	0.00	1292	0.00
LA15	1207	1212	0.41	1207	0.00	1207	0.00	1207	0.00
LA16	945	1005	6.35	946	0.11	945	0.00	945	0.00
LA17	784	812	3.57	785	0.13	784	0.00	784	0.00
LA18	848	885	4.36	861	1.53	848	0.00	848	0.00
LA19	842	875	3.92	848	0.71	849	0.83	842	0.00
LA20	902	912	1.11	902	0.00	907	0.55	907	0.55
LA21	1046	1107	5.38	1055	0.86	1055	0.86	1059	1.24
LA22	927	1018	9.82	954	2.91	935	0.86	935	0.86
LA23	1032	1051	1.84	1032	0.00	1032	0.00	1032	0.00
LA24	935	1011	8.13	948	1.39	956	2.25	949	1.50
LA25	977	1062	8.70	988	1.13	991	1.43	986	0.92
LA26	1218	1296	6.40	1218	0.00	1218	0.00	1218	0.00
LA27	1235	1362	10.28	1259	1.94	1269	2.75	1269	2.75
LA28	1216	1330	9.38	1216	0.00	1241	2.06	1234	1.48
LA29	1152	1339	15.73	1164	1.04	1207	4.77	1214	5.38
LA30	1355	1410	4.06	1355	0.00	1355	0.00	1355	0.00
LA31	1784	1798	0.78	1784	0.00	1784	0.00	1784	0.00
LA32	1850	1868	0.97	1850	0.00	1850	0.00	1850	0.00
LA33	1719	1731	0.70	1719	0.00	1719	0.00	1719	0.00
LA34	1721	1788	3.89	1721	0.00	1721	0.00	1721	0.00
LA35	1888	1913	1.32	1888	0.00	1888	0.00	1888	0.00
LA36	1268	1396	10.09	1275	0.55	1291	1.81	1297	2.29
LA37	1397	1517	8.59	1422	1.79	1426	2.08	1425	2.00
LA38	1196	1315	9.95	1209	1.09	1254	4.85	1230	2.84
LA39	1233	1304	5.76	1235	0.16	1264	2.51	1259	2.11

Problem	Known Optimum	ACO [24]		Tabu Search [31]		BCO with GS-TESN5		BCO with LRS-TESN5	
		Best Makespan	*B (%)*	*Best Makespan*	*B (%)*	*Best Makespan*	*B (%)*	*Best Makespan*	*B (%)*
LA40	1222	1307	6.96	1234	0.98	1254	2.62	1246	1.96
Average of 40 instances			3.96		0.43		0.76		0.65

7 Conclusion

A BCO algorithm with modified two-enhancement scheme with neighbourhood N5 perturbation (BCO+mTESN5) local search is proposed to solve the job shop scheduling problem (JSSP). The mTESN5 local search is designed such that the perturbation is only performed on a targeted bottleneck machine. A list of machines is identified and ranked according to its maximum lateness using the bottleneck identification steps of the shifting bottleneck heuristic (SBH). Two selection strategies (i.e. greedy and linear ranking selection strategies) are implemented and examined to select a targeted bottleneck machine from the list. Thus, two different algorithms are obtained based on the selection strategy as follows: the BCO+mTESN5 algorithm with greedy selection (denoted by BCO with GS-TESN5) and the BCO+mTESN5 algorithm with linear ranking selection (denoted by BCO with LRS-TESN5). The proposed algorithms are tested on a set of 82 JSSP benchmark problems. The results show that the BCO with GS-TESN5 algorithm reduces the computational time by 10% compared with the BCO+TESN5 algorithm. The BCO with LRS-TESN5 algorithm reduces the computational time by 25% compared with BCO+TESN5. In tackling hard problem such as ABZ, SWV and YN series of the benchmark problem, the BCO with LRS-TESN5 algorithm is able to shorten the average computational time to obtain the best result compared with the BCO+TESN5 algorithm.

As for the future work, the BCO algorithm can be applied in SBH algorithm during re-scheduling process. During the re-scheduling process in the classic SBH, algorithm such as the EDD dispatching rule or branch-and-bound algorithm are used. The BCO algorithm can be used to replace the EDD dispatching rule or branch-and-bound algorithm, in solving the one-machine problems.

ACKNOWLEDGEMENT
This work was supported by the Collaborative Research in Engineering, Science and Technology (CREST) R&D Grant (Grant No: 304/PKOMP/650652/C121).

References

[1] Pinedo, M. L. 2012. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media.

[2] Brucker, P., Jurisch, B. and Sievers, B. 1994. A branch and bound algorithm for the job-shop scheduling problem. *Discrete applied mathematics*, Vol. 49, No. 1, pp. 107-127.

[3] Land, A. H. and Doig, A. G. 2010. An automatic method for solving discrete programming problems. In: *50 Years of Integer Programming 1958-2008*. Springer, pp. 105-132.

[4] Dantzig, G. B. 2002. Linear programming. *Operations Research*, Vol. 50, No. 1, pp. 42-47.

[5] Zobolas, G. I., Tarantilis, C. D. and Ioannou, G. 2008. Exact, heuristic and meta-heuristic algorithms for solving shop scheduling problems. In: Xhafa, F. and Abraham, A. eds. *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*. Vol. 128 2008. Springer, pp. 1-40.

[6] Osman, I. H. and Laporte, G. 1996. Metaheuristics: A bibliography. *Annals of Operations research*, Vol. 63, No. 5, pp. 511-623.

[7] Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G. and Węglarz, J. 2007. Scheduling in Job Shops. In: *Handbook on Scheduling: From Theory to Applications*. pp. 345-396.

[8] Ren, Q.-D.-E.-J. and Wang, Y. 2012. A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, Vol. 39, No. 10, pp. 2291-2299.

[9] Spanos, A. C., Ponis, S. T., Tatsiopoulos, I. P., Christou, I. T. and Rokou, E. 2014. A new hybrid parallel genetic algorithm for the job-shop scheduling problem. *International Transactions in Operational Research*, Vol. 21, No. 3, pp. 479-499.

[10]Kurdi, M. 2015. A new hybrid island model genetic algorithm for job shop scheduling problem. *Computers & Industrial Engineering*, Vol. 88, No., pp. 273-283.

[11]Asadzadeh, L. 2015. A local search genetic algorithm for the job shop scheduling problem with intelligent agents. *Computers & Industrial Engineering*, Vol. 85, No., pp. 376-383.

[12]Meeran, S. and Morshed, M. S. 2012. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: A case study. *Journal of Intelligent Manufacturing*, Vol. 23, No. 4, pp. 1063-1078.

[13]Lin, Y. K. and Chong, C. S. 2015. A tabu search algorithm to minimize total weighted tardiness for the job shop scheduling problem. *Journal of Industrial and Management Optimization*, Vol. 12, No. 2, pp. 703-717.

[14]Peng, B., Lu, Z. and Cheng, T. C. E. 2015. A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, Vol. 53, No., pp. 154-164.

[15]Zhang, R. and Wu, C. 2011. A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardiness objective. *Computers & Operations Research*, Vol. 38, No. 5, pp. 854-867.

[16]Song, S.-Z., Ren, J.-J. and Fan, J.-X. 2012. Improved simulated annealing algorithm used for job shop scheduling problems. In: Xie, A. and Huang, X. eds. *Advances in Electrical Engineering and Automation*. Vol. 139. pp. 17-25.

[17]Antonio Cruz-Chavez, M. 2015. Neighbourhood generation mechanism applied in simulated annealing to job shop scheduling problems. *International Journal of Systems Science*, Vol. 46, No. 15, pp. 2673-2685.

[18]Zhang, X.-F., Koshimura, M., Fujita, H. and Hasegawa, R. 2011. An efficient hybrid particle swarm optimization for the job shop scheduling problem. In: *Proceedings of the 2011 IEEE International Conference on Fuzzy Systems.* pp. 622-626.

[19]Zhang, R., Song, S. and Wu, C. 2012. A two-stage hybrid particle swarm optimization algorithm for the stochastic job shop scheduling problem. *Knowledge-Based Systems*, Vol. 27, No., pp. 393-406.

[20]Zhang, R. and Wu, C. 2013. A neighbourhood property for the job shop scheduling problem with application to hybrid particle swarm optimization. *IMA Journal of Management Mathematics*, Vol. 24, No. 1, pp. 111-134.

[21]Yoshikawa, M. and Terai, H. 2006. A hybrid ant colony optimization technique for job-shop scheduling problems. In: Baik, D.K. et al. eds. *Proceedings of Fourth International Conference on Software Engineering Research, Management and Applications.* pp. 95-100.

[22]Duc, D. D., Dinh, H. Q. and Xuan, H. H. 2008. On the pheromone update rules of ant colony optimization approaches for the job shop scheduling problem. In: Bui, T.D. et al. eds. *Proceedings of Intelligent Agents and Multi-Agent Systems*. Vol. 5357. pp. 153-160.

[23]Seo, M. and Kim, D. 2010. Ant colony optimisation with parameterised search space for the job shop scheduling problem. *International Journal of Production Research*, Vol. 48, No. 4, pp. 1143-1154.

[24]Flórez, E., Gómez, W. and Bautista, L. 2013. An ant colony optimization algorithm for job shop scheduling problem. *International Journal of Artificial Intelligence & Applications*, Vol. 4, No. 4, pp. 53-66.

[25]Chong, C. S., Low, M. Y. H., Sivakumar, A. I. and Gay, K. L. 2006. A bee colony optimization algorithm to job shop scheduling. In: *Proceedings of the 2006 Winter Simulation Conference, Vols 1-5.* IEEE, pp. 1954-1961.

[26]Wong, L.-P., Puan, C. Y., Low, M. Y. H., Wong, Y. W. and Chong, C. S. 2010. Bee colony optimisation algorithm with big valley landscape exploitation for job shop scheduling problems. *International Journal of Bio-Inspired Computation*, Vol. 2, No. 2, pp. 85-99.

[27]Wong, L.-P. 2012. *A generic bee colony optimization framework for combinatorial optimization problems.* PhD Thesis, School of Computer Engineering, Nanyang Technological University, Singapore.

[28]Roy, B. and Sussmann, B. 1964. *Les problemes d'ordonnancement avec contraintes disjonctives*. Technical Report 9, SEMA, Note D.S., Paris.

[29]Balas, E. 1969. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations Research*, Vol. 17, No. 6, pp. 941-957.

[30]Abdelmaguid, T. F. 2009. Permutation-induced acyclic networks for the job shop scheduling problem. *Applied Mathematical Modelling*, Vol. 33, No. 3, pp. 1560-1572.

[31]Nowicki, E. and Smutnicki, C. 1996. A fast taboo search algorithm for the job shop problem. *Management Science*, Vol. 42, No. 6, pp. 797-813.

[32]Alharkan, I. M. 2005. Job shop scheduling. In: *Algorithms for Sequencing and Scheduling*. Industrial Engineering Department, King Saud University, Riyadh, Saudi Arabia.

[33]Hart, E., Ross, P. and Corne, D. 2005. Evolutionary scheduling: A review. *Genetic Programming and Evolvable Machines*, Vol. 6, No. 2, pp. 191-220.

[34]Lučić, P. and Teodorović, D. 2002. Transportation modeling: An artificial life approach. In: *Proceedings of the 14th IEEE International Conference onTools with Artificial Intelligence, 2002.(ICTAI 2002).* IEEE, pp. 216-223.

[35]Pham, D. T., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S. and Zaidi, M. 2006. The bees Algorithm – A novel tool for complex optimisation. In: *Proceedings of 2006 Virtual International Conference on Intelligent Production Machines and Systems (IPROMS 2006).* pp. 454-461.

[36]Pham, D. T., Koc, E., Lee, J. Y. and Phrueksanant, J. 2007. Using the bees algorithm to schedule jobs for a machine. In: *Proceedings of the 8th International Conference on Laser Metrology, CMM and Machine Tool Performance, (LAMDAMAP)* Euspen, UK, Cardiff. pp. 430-439.

[37]Lara, C., Flores, J. J. and Calderón, F. 2008. Solving a school timetabling problem using a bee algorithm. In: *Proceedings of the 2008 Advances in Artificial Intelligence (MICAI 2008).* Springer, pp. 664-674.

[38]Abbass, H. 2001. MBO: Marriage in honey bees optimization - A haplometrosis polygynous swarming approach. In: *Proceedings of the 2001 Congress on Evolutionary Computation* IEEE, pp. 207-214.

[39]Benatchba, K., Admane, L. and Koudil, M. 2005. Using bees to solve a data-mining problem expressed as a max-sat one. In: Mira, J. and Álvarez, J.R. eds. *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*. Vol. 3562. Springer, pp. 212-220.

[40]Koudil, M., Benatchba, K., Tarabet, A. and Sahraoui, E. B. 2007. Using artificial bees to solve partitioning and scheduling problems in codesign. *Applied Mathematics and Computation*, Vol. 186, No. 2, pp. 1710-1722.

[41]Karaboga, D. 2005. *An idea based on honey bee swarm for numerical optimization*. Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department.

[42]Karaboga, D., Akay, B. and Ozturk, C. 2007. Artificial bee colony (ABC) optimization algorithm for training feed-forward neural networks. In: *Modeling Decisions for Artificial Intelligence*. Springer, pp. 318-329.

[43]Teodorović, D. and Dell’Orco, M. 2005. Bee colony optimization - A cooperative learning approach to complex transportation problems. In: *Proceedings of 16th Mini–EURO Conference and 10th Meeting of EWGT (Advanced OR and AI Methods in Transportation).* Poznan, Poland. pp. 51-60.

[44]Teodorović, D. 2009. Bee colony optimization (BCO). In: Lim, C.P. et al. eds. *Innovations in Swarm Intelligence*. Springer, pp. 39-60.

[45]Teodorović, D., Lučić, P., Marković, G. and Dell'Orco, M. 2006. Bee colony optimization: Principles and applications. In: *Proceedings of 8th Seminar on Neural Network Applications in Electrical Engineering (NEUREL 2006).* Belgrade, Serbia & Montenegro. IEEE, pp. 151-156.

[46]Wong, L.-P., Puan, C. Y., Low, M. Y. H. and Chong, C. S. 2008. Bee colony optimization algorithm with big valley landscape exploitation for job shop scheduling problems. In: *Proceedings of 2008 Winter Simulation Conference (WSC 2008).* IEEE, pp. 2050-2058.

[47]Błażewicz, J., Domschke, W. and Pesch, E. 1996. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, Vol. 93, No. 1, pp. 1-33.

[48]Adams, J., Balas, E. and Zawack, D. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science*, Vol. 34, No. 3, pp. 391-401.

[49]Alijla, B. O., Wong, L.-P., Lim, C. P., Khader, A. T. and Al-Betar, M. A. 2014. A modified intelligent water drops algorithm and its application to optimization problems. *Expert Systems with Applications*, Vol. 41, No. 15, pp. 6555-6569.