Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

BEE COLONY OPTIMIZATION ALGORITHM WITH BIG VALLEY LANDSCAPE EXPLOITATION FOR JOB SHOP SCHEDULING PROBLEMS

Li-Pei Wong
Chi Yung Puan
Malcolm Yoke Hean Low

School of Computer Engineering
Nanyang Technological University
Nanyang Avenue, SINGAPORE 639798

Chin Soon Chong

Singapore Institute of Manufacturing Technology
71 Nanyang Drive
SINGAPORE 638075

ABSTRACT

Scheduling is a crucial activity in semiconductor manufacturing industry. Effective scheduling in its operations leads to improvement in the efficiency and utilization of its equipment. Job Shop Scheduling is an NP-hard problem which is closely related to some of the scheduling activities in this industry. This paper presents an improved Bee Colony Optimization algorithm with Big Valley landscape exploitation as a biologically inspired approach to solve the Job Shop Scheduling problem. Experimental results comparing our proposed algorithm with Shifting Bottleneck Heuristic, Tabu Search Algorithm and Bee Colony Algorithm with Neighborhood Search on Taillard JSSP benchmark show that it is comparable to these approaches.

1 INTRODUCTION

Semiconductor manufacturing industry is a complex yet dynamic business. Some major activities in the semiconductor production are wafer fabrication, wafer probe, product assembly and final testing. These activities are highly capital intensive and need to be performed in an unpredictable environment, as the activities are sensitive to disruption factors such as frequent facilities maintenance, rework, machine downtime etc. To compete in a versatile environment where the product life cycle is considerably short, semiconductor manufacturers are trying different methods to improve productivity and minimize the cycle time of their products. Solutions to such problems play an important role in ensuring that scarce resources are allocated effectively to competing activities, so as to maximize their utilization and efficiency.

Job Shop Scheduling Problem (JSSP) is closely related to activities in semiconductor manufacturing industry such as part routing, part processing operations and coordination of part handling as discussed by Gupta and Sivakumar (2006), and Cavalieri et al. (1999). It is NP-hard in nature (Lenstra et al. 1977). In a typical JSSP, a sequential job allocation on resources (machines) that optimizes a particular objective function is to be determined. While many algorithms exist to solve the JSSP (Blazewicz et al. 1996, Lee et al. 1997), the Bee Colony Optimization (BCO) algorithm has recently been adapted (Chong et al. 2006, Chong et al. 2007). The bee inspired algorithms are generalized from the foraging behaviors of bees where waggle dance is used as a communication medium to attract other bees to a food source. When the behaviors are applied algorithmically on complex and dynamic problems, the algorithm appears to be self-organized, flexible and robust in discovering solutions to the problems (Bonabeau and Meyer 2001).

The bee inspired algorithms have also been attempted in various areas including the dynamic server allocation in Internet hosting center (Nakrani and Tovey 2004), hex game playing program (Rijswijck 2007), Traveling Salesman Problem (Lucic and Teodorovic 2002, Lucic and Teodorovic 2003, Wong et al. 2008), and Telecommunication Network Routing (Wedde et al. 2004). A survey that discusses bee inspired algorithms and their applications to some generalized assignment problems can be found in Baykosoglu et al. (2007).

In this paper, a BCO algorithm with Big Valley landscape (BCBV) is presented. Besides the foraging behaviors, bees in the proposed algorithm are equipped with the ability to explore the search space which appears in a Big Valley structure as discussed in the works by Reeves (1999), Nowicki and Smutnicki (2005), and Boese et al. (2008). An effective search around the Big Valley structure will help in locating the best solution in the space. The BCBV algorithm is tested on Taillard JSSP benchmark and compared against the Shifting Bottleneck Heuristic (SBP), Tabu Search Algorithm (TSA), and Bee Colony Algorithm with Neighborhood Search (BCNS) (see Sections 5.1 and 5.2 for details).

This paper starts with a discussion on JSSP in Section 2. Section 3 explains the Big Valley landscape structure. A discussion on the BCBV algorithm is presented in Section

978-1-4244-2708-6/08/$25.00 ©2008 IEEE

4. Experiments and results are presented in Section 5. Finally, this paper ends with a conclusion.

2 JOB SHOP SCHEDULING PROBLEM (JSSP)

As presented by Adams et al. (1988), JSSP is defined by a set J of n jobs, $J = \{1, 2, \ldots, n\}$. These jobs are to be processed on a set of m machines, $M = \{1, 2, \ldots, m\}$. O is a set of operations, $O = \{0, O_{11}, \ldots, O_{1m}, O_{n1}, \ldots, O_{nm}, z\}$ where O_{ij} denotes the j-th operation of job J_i. 0 and z denotes two fabricated operations which represent the "first" and "ultimate" operation. Thus, $|O| = (n*m)+2$. Each operation O_{ij} is associated with t_{ij} and τ_{ij} which denote its earliest start time and processing time respectively. Apart from these, the following constraints have to be fulfilled:

- Each job J_i in set J is composed of a set A_i which consists of ordered pairs of operations, constrained by the precedence relations in (1).

$$t_{i(j+1)} - t_{ij} \geq \tau_{ij}, \quad \forall (O_{ij}, O_{i(j+1)}) \in A_i \tag{1}$$

- Each machine M_r in set M is composed of a set E_r which describes the set of all pairs of operations to be performed on machine r. Each operation O_{ij} will be processed for τ_{ij} without interruption and each machine can handle at most one operation at a time. These constraints are shown in (2).

$$\left.\begin{array}{c} t_{ij} - t_{kl} \geq \tau_{kl} \\ \oplus \\ t_{kl} - t_{ij} \geq \tau_{ij} \end{array}\right\} \forall (O_{ij}, O_{kl}) \in E_r, \forall r \in M \tag{2}$$

- For every operation in O, t_{ij} must be greater than or equal to 0. This constraint guarantees the completion of all jobs as shown in (3).

$$t_{ij} \geq 0, \quad \forall O_{ij} \in O \tag{3}$$

Although there are many metrics that can be considered as the objective function of JSSP, makespan (C_{max}) is the most common and will be the focus in this paper. C_{max} is defined as the longest duration for which all operations of all jobs are completed.

2.1 Disjunctive Graph Representation, Critical Path and Its Block Decomposition

A common representation for JSSP is the disjunctive graph. A disjunctive graph is a collection of nodes (vertices) and edges (arcs). Nodes are linked by the edges of the graph. Hence, a disjunctive graph G is defined as

$$G = \left(O, E_{Conj} \cup E_{Disj}\right) \tag{4}$$

where O is the set of operations as defined in Section 2. E_{Conj} is a set of directed conjunctive edges representing the precedence constraints of each job as described in (1). E_{Disj} is a set of bi-directional disjunctive edges representing the capacity constraints of each machine as described in (2). These disjunctive edges link all the operations that need to be handled by a particular machine. An example of the disjunctive graph based on a 3-job x 3-machine JSSP in Table 1 is shown in Figure 1. Each row of the table represents a pre-defined machine precedence order for each job with the processing time in parentheses. A solution can be obtained by converting bi-directional disjunctive edges to become directed edges. To make sure that a feasible solution is obtained, the conversion is performed such that no cycle exists in the graph. This will eventually turn the disjunctive graph to a directed graph as shown in Figure 2.

Table 1. An Example of 3-job x 3-machine JSSP.

Job	Machine (Processing time)		
1	2 (3)	1 (13)	3 (6)
2	1 (8)	2 (4)	3 (12)
3	3 (10)	2 (5)	1 (5)

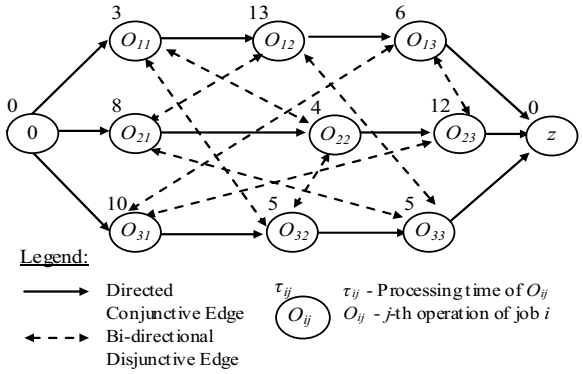

Figure 1: A Disjunctive Graph for 3-job x 3-machine JSSP Instance in Table 1.

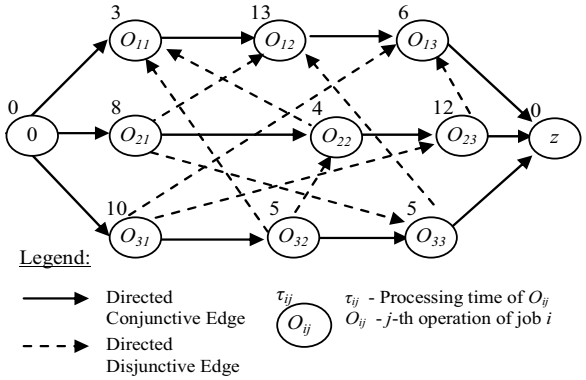

Figure 2: A Directed Graph (Feasible Solution) for 3-job x 3-machine JSSP Instance in Table 1.

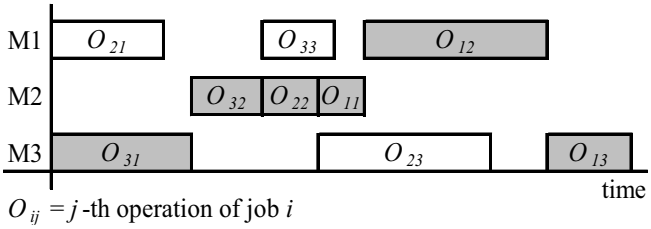

Figure 3: Gantt Chart for the Directed Graph in Figure 2.

The longest path (or critical path) in the directed graph provides the makespan for the JSSP. The critical path in Figure 2 is given by: $0 \rightarrow O_{31} \rightarrow O_{32} \rightarrow O_{22} \rightarrow O_{11} \rightarrow O_{12} \rightarrow O_{13} \rightarrow z$. The sum of all the processing times is 0 + 10 + 5 + 4 + 3 + 13 + 6 + 0 = 41. To better illustrate the critical path found in Figure 2, a Gantt chart is shown in Figure 3 where the operations in the critical path are shaded.

A critical path consists of a set of operations which cannot be delayed so that all the jobs will be completed on time. The critical path can be decomposed into a set of r blocks, $B = \{b_1, b_2, \ldots b_r\}$ as described by Grabowski et al. (1986), and Nowicki and Smutnicki (1996). Each block contains an order set of operations/nodes which are processed on the same machine $b_k = \{d_{k1}, d_{k2}, \ldots, d_{ki}\}$ where $d \in O$. At the same time, two consecutive blocks must contain operations/nodes which are processed on different machines. The following example illustrates the block decomposition of the critical path found in Figure 3.

As stated earlier, one of the critical paths for the 3-job x 3-machine JSSP instance in Table 1 is given by $\{O_{31}, O_{32}, O_{22}, O_{11}, O_{12}, O_{13}\}$. By performing block decomposition, four blocks are identified where $r = 4$, $B = \{b_1, b_2, b_3, b_4\}$. The content of each block is as follows: $b_1 = \{O_{31},\}$, $b_2 = \{O_{32}, O_{22}, O_{11}\}$, $b_3 = \{O_{12}\}$, $b_4 = \{O_{13}\}$. This block decomposition is the central idea in the implementation of the neighborhood operator which will be discussed in Section 4.3.

In the subsequent section, the Big Valley landscape structure will be discussed. The discussion will include how the landscape looks and its characteristics.

3 THE BIG VALLEY LANDSCAPE STRUCTURE

A landscape can be described as a structure of the neighborhood generated by a heuristic operator used to traverse the search space of the problem in view of the objective function. Reeves (1999) suggested that when a heuristic search approach is applied to a combinatorial optimization problem that defines a unique search space, a "landscape" will be created. In addition, it is found that different landscapes will be created by different search operators used in the search space exploration. A landscape consists of many local optima or false peaks which will be changing with respect to the heuristic search operators. The existence of these local optima or false peaks often obstruct the search from locating global optimum as illustrated in Figure 4.

However, these landscape structures can assist the search of global optimum instead of obstructing it. One such landscape structure is the "Big Valley" structure observed by Boese et al. (2008) in the 2-opt operator for Traveling Salesman Problem (TSP). Similar landscape structure has since been extended to flow shop scheduling problem (FSP) by Reeves (1999), and also in JSSP by Nowicki and Smutnicki (2005).

In the Big Valley landscape structure, local optima tend to exist close to one another in clusters, with each cluster centered on the global optimum forming a valley structure as illustrated in Figure 4. The formation of the Big Valley landscape has strong implication for how heuristic search should be performed. The Big Valley structure suggests that the determination of new start points for search should be based on previous local optimum rather than based on a random point in the search space. This is because good candidate solutions are often found to be close to other good solutions. By exploring and exploiting the areas near to these local optima effectively, the search will be directed towards the global optimum eventually.

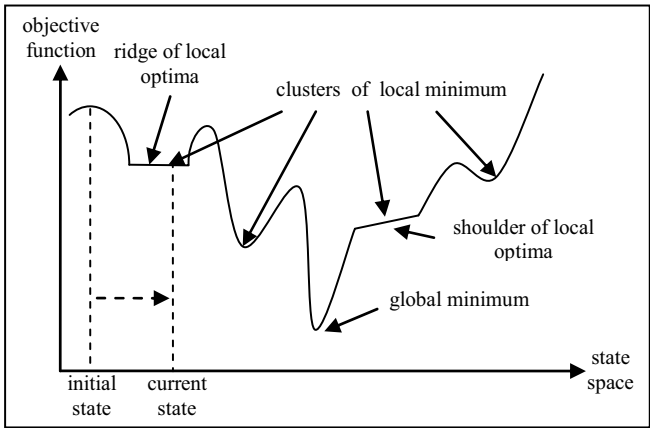

Figure 4: A Landscape with the Big Valley Feature.

3.1 Big Valley Landscape Analysis

To investigate the Big Valley landscape structure, a plot from a sample of 1500 solutions collected through a run of BCBV algorithm on the ta_01 (15-job x 15-machine) problem is generated as shown in Figure 5. ta_01 is one of the datasets in Taillard JSSP benchmark which will be discussed in Section 5.1. The solutions are plotted using the percentage makespan difference, θ (see Section 5.1) versus their heuristic distance (see Section 4.4.1) from a fixed local optimum.

In Figure 5, solutions are concentrated in four separate clusters at different distances from a fixed local optimum. The clusters are formed by the neighborhood operator (see Section 4.3.2) which is integrated in BCBV. Figure 5 also shows that the clusters are some heuristic distance away from one another. Search should be performed intensely within clusters to find new local optima with better makespan since they are most probable to occur close to one another. The clustering approach by neighborhood operator and intensive searching around different clusters which are certain heuristic distance apart is implemented in BCBV algorithm. Details about the analysis on the Big Valley landscape can be obtained in Reeves (1999), and Nowicki and Smutnicki (1996).

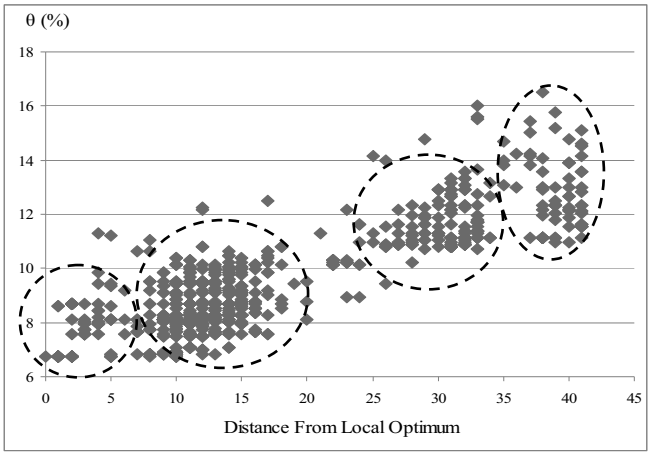

Figure 5: θ versus Distance from a Local Optimum.

4 BCO WITH BIG VALLEY LANDSCAPE

This section first describes the natural foraging model of a typical bee colony. Next, an algorithmic framework of BCBV is presented. It is followed by an overview of the foraging model used in constructing feasible solutions for JSSP, and the waggle dance model used in finding good solutions. Finally, a discussion on the Big Valley landscape exploitation in BCO is provided.

4.1 Bee Colony

The foraging behavior in a bee colony remains mysterious for many years until von Frisch (1974) translated the language embedded in bee waggle dances. Waggle dance operates as a communication tool among bees. Through the waggle dance, bees could describe the distance, direction, and description of the food source to other bees. Distance is conveyed by the type and duration of the waggle dance.

Suppose a bee found a rich food source, a figure-eight pattern is shown in the dance. This figure-eight dance consists of a straight waggle run followed by a turn to the right back to the starting point, and then another straight waggle run followed by a turn to the left and back to the starting point again. Via these informative dances, the bee has actually informed its hive mates about the direction and distance of the food source. von Frisch (1974) also suggested that bees could describe the type of flower which is richest to forage through the use of pollen. Once a bee has associated itself with the particular scent of pollen from a waggle dance, it will ignore flowers with other scents at the indicated location. Such a mechanism achieves a kind of short-term memory to differentiate and identify food sources and will be explored through the addition of a Taboo list in the BCO model in this paper. Further discussions about waggle dance can be found in Dyer (2002), and Biesmeijer and Seeley (2005).

4.2 BCBV Algorithm

The BCBV algorithm uses a similar model based on bees' foraging behaviors as shown in Figure 6. The BCBV starts with a set of feasible schedules generated by dispatching rules, which will be discussed in Section 4.3.1. A combination of foraging and performing waggle dance constitutes one cycle (or iteration). The foraging model will be discussed in Sections 4.3.2 and 4.3.3. The waggle dance model, which includes how a bee observes, selects and performs a waggle dance, is implemented using a linked list *WL* and will be discussed in Section 4.4.

The BCBV algorithm is executed for N_{max} iterations and the best solution found during the searching process will be presented as the final schedule at the end of a run.

```
procedure BCBV
    C_max_best ← ∞
    N_iter ← 0
    GenerateInitialSolution( ) [see section 4.3.1]
    while N_iter <> N_max do
      for each forager bee f_i do
        if WL<>{ }
          f_i.ObserveNSelectDance( ) [see section 4.4.2 & 4.4.3]
        end if
        f_i.C_max ← f_i.Forage( ) [see section 4.3.2 & 4.3.3]
        if f_i.C_max < C_max_best
          C_max_best ← f_i.C_max
          f_i.PerformWaggleDance( ) [see section 4.4.1]
        end if
      end for
      N_iter ← N_iter+1
    end while
end procedure BCBV
```

Figure 6: Algorithmic Framework of BCBV.

4.3 Foraging Model with Neighborhood Search

The foraging model starts with a group of bees constructing a set of feasible solutions by using a set of dispatching rules for JSSP with the use of disjunctive graph representation as mentioned in Section 2.

4.3.1 Generate Initial Feasible Solution

The dispatching rules that are applied in the BCBV algorithm include Shortest Processing Time, Longest Processing Time, Most Work Remaining, Least Work Remaining, Work in Next Queue, Last In First Out, First In First out, Shortest Processing Time+Work in Next Queue, Shortest Processing Time+Most Work Remaining and Random Dispatching Rules. In the Random Dispatching Rules, bees are allowed to randomly pick one of the listed rules to construct a feasible solution. These rules are applied in a round robin fashion for all the bees. Implementation details of the listed rules can be found in the work by Yamada (2003).

The set of feasible solutions generated by the dispatching rules will be used to generate other feasible solutions. Other than this, bees may also generate new solution from their own solution found in previous iteration or based on the dance (solution) that they followed. These three sets of feasible solutions will serve as a foundation to produce other solutions for the rest of the algorithm via a neighborhood operator. The mechanism of the operator will be explained in the subsequent section.

4.3.2 Neighborhood Operator

Each feasible solution, S_x, has it own critical path which can be identified via the disjunctive graph. To produce a set of moves based on S_x, a neighborhood operator $C(S_x)$, which is based on the block structure described in Section 2, is defined in (5):

$$C(S_x) = \bigcup_{j=1}^{r} C_j(b_j),\ \forall b_j \in B \tag{5}$$

where C_j is defined as in (6), (7), and (8):

$$C_1(b_1) = \begin{cases} \{(d_{1(i-1)}, d_{1i})\}, |b_1| > 1 \text{ and } |B| > 1 \\ \varnothing \quad , \text{otherwise} \end{cases} \tag{6}$$

$$C_j(b_j) = \begin{cases} \{(d_{j1}, d_{j2}), (d_{j(i-1)}, d_{ji})\}, |b_j| > 3 \text{ and } |B| > j \\ \{(d_{j1}, d_{j2})\} \quad , |b_j| = 2 \text{ and } |B| > j \\ \varnothing \quad , \text{otherwise} \end{cases}$$

$$\text{for j=2, …, } r\text{-1} \tag{7}$$

$$C_r(b_r) = \begin{cases} \{(d_{r1}, d_{r2})\}, |b_r| > 1 \text{ and } |B| > 1 \\ \varnothing \quad , \text{otherwise} \end{cases} \tag{8}$$

The generation of $C(S_x)$ is adapted from the work by Nowicki and Smutnicki (1996). (6) suggests that swapping is allowed on the last two operations in the first block. (7) suggests that swapping is allowed on the first two (and last two) in every intermediate blocks (b_2, …, b_{r-1}). (8) suggests that the swapping is allowed on the first two operations in the last block. Once the neighborhood operation ends, $C(S_x)$ contains a set of moves that can be used to generate new feasible solutions. Of all the moves in $C(S_x)$, moves that are able to generate better C_{max} when compared to the current C_{max} (improving moves) are placed into the set M_I, and moves that have been recently visited (Taboo moves) are placed in the set M_T. A bee will then decide which move to select based on the strategies that will be discussed in Section 4.3.3.

4.3.3 Move Selection Strategies

Recall that bees could remember scent and avoid searching patches of flowers with different scents through the pollen identification ability (refer to Section 4.1). To aid these bees in making a better decision in selecting a move, a similar short-term memory is given to the artificial bees in the model. This short-term memory is developed via the use of a Taboo list. Each time a move is selected, its inverse move will be added to the Taboo list. Should the inverse move be encountered in the next search, its selection is avoided if possible. The aim of the Taboo list is to direct the search process away from recently visited solutions so that more of the unexplored search space can be reached. Hence, more solutions can be evaluated. Besides Taboo list, another strategy is introduced to aid the bees to select a better move. Using this strategy, a bee will select at random a move from M_I should one exist, else it will select at random a non-improving move.

The combination of the neighborhood operator, Taboo list and move selection strategy form the foraging model shown in Figure 7. It corresponds to the Forage() method in Figure 6.

Step 1: Find a set of moves from a feasible solution, S_x by using operator $C(S_x)$.
Step 2: Identify M_I and M_T from $C(S_x)$.
Step 3: If $M_I - M_T \neq \varnothing$
  Randomly pick a move from $M_I - M_T$.
  Go to Step 7.
Step 4: If $M_I \cap M_T \neq \varnothing$
  Randomly pick a move from $M_I \cap M_T$.
  Go to Step 7.
Step 5: If $\overline{M}_T \neq \varnothing$
  Randomly pick a move from $\overline{M}_T$.
  Go to Step 7.
Step 6: Random pick a move from $C(S_x)$.
Step 7: Add the inverse of selected move to M_T
Step 8: Return the selected move.

Figure 7: Algorithm for Forage().

4.4 Waggle Dance with Exploitation of Big Valley Landscape

Upon returning to the hive after foraging, a bee will need to perform waggle dance in order to convey information about its discovery to other hive mates. At any one time, a bee will perform waggle dance if its C_{max} is smaller than the current best C_{max} among all the bees. Note that in our implementation, the solutions representing the dances by the bees are accumulated in an unbounded list, *WL*. Accumulation of these dances (solutions) over time will create a landscape which consists of multiple local optima as discussed in Section 3. To manage the landscape effectively, approaches stated in Sections 4.4.1 to 4.4.3 are proposed.

4.4.1 Dance Accumulation Strategy

Since *WL* is unbounded, it might accumulate too many dances such that a unique landscape could not be identified. To maintain a finite set of unique peaks (local optima), a replacement approach is introduced. This approach compares the similarity of the newly generated solution

(denoted as S_p) with each solution in *WL* (denoted by S_q where $S_q \in WL$) using a distance metric *D*.

The role of *D* is to compare the similarity between two solutions. It measures the number of bit difference between the directed disjunctive arcs representation of two solutions as defined in (9). The E_{Disj} of both S_p and S_q are represented as a total-ordering bitmap which consists of a string of booleans.

$$D = \frac{\left| diff\left\{E_{Disj}(S_p), E_{Disj}(S_q)\right\}\right|}{\left|E_{Disj}(S_p)\right|} \tag{9}$$

The solution S_q that is within a distance of replacement threshold D_R from S_p will be replaced, if one exists. If there are two or more solutions in the list that meet the threshold, all of them will be replaced accordingly. Otherwise, S_p will be appended to *WL*.

```
Step 1: If Sp.Cmax < Cmax_best
            Search through WL and accumulate Sp in WL by re-
                placing solutions which are within the replace-
                ment threshold, DR.
            Sp.CtElite ← 0.
Step 2: For each entry Sq in WL
            If Sq.CtElite > NAttempt
                WL.remove(Sq)
```

Figure 8: Algorithm for PerformWaggleDance().

To make sure that every solution in *WL* is searched intensively by bees, a counter, Ct_{Elite}, is associated to each solution. Ct_{Elite} is incremented whenever the associated solution (dance) is used (followed) by another bee. When a solution is used for $N_{Attempt}$ (a predefined value) times, the entry will be removed from the list. By using the above strategy, exploitation of different peaks in the Big Valley landscape is preserved. Figure 8 shows the algorithm for the dance accumulation strategy which corresponds to the PerformWaggleDance() method in Figure 6.

4.4.2 Dance Observation Strategy

Before a bee leaves its hive to start the foraging process, it decides if it will observe and follow a dance shown by previous dancer with a probability of P_{follow}. In the BCO algorithm suggested by Chong et al. (2006), P_{follow} is adjusted dynamically via a profitability lookup table. A different strategy is adopted in BCBV. P_{follow} is adjusted based on the profitability rating of a bee, Pf_i and the average profitability of the waggle dance list, Pf_{WL}. Pf_i and Pf_{WL} are defined in (10), and (11) respectively:

$$Pf_i = \frac{1}{C^i_{\max}} \tag{10}$$

$$Pf_{WL} = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{C^i_{\max}} \tag{11}$$

where C^i_{max} is the makespan of the schedule generated by a forager f_i (Chong et al., 2007) and *n* is the number of dancing bee. P_{follow} is determined by (12) where the 0.9 and 0.6 are obtained after a series of parameter tuning tests.

$$P_{follow} = \begin{cases} 0.60, & Pf_i < 0.90 * Pf_{WL} \\ 0.00, & otherwise \end{cases} \tag{12}$$

4.4.3 Dance Selection Strategy

After a bee has decided to observe and follow a dance, the next step is to select a dance from *WL*. Some selection strategies tested in the development of the BCBV algorithm are:

- Most Recently Added (MRA) – the most recently added solution to *WL* is selected.
- Round Robin (RR) – the solution is selected in a round robin manner, one after another in succession.
- Roulette Wheel (ROUL) – the solution is selected based on the C^i_{max}. A more profitable dance will have a higher chance to be selected.
- Random Improving Move (RIM) – the solution that is better than the current makespan in *WL* is selected in an unbiased manner.

These strategies are aimed at exploring the Big Valley landscape more effectively. As initial experiments show that RR gives the best results on a set of benchmark dataset, it is adopted in the BCBV model in this paper. Figure 9 shows the algorithm which corresponds to the ObserveNSelectDance( ) method in Figure 6.

```
Step 1: Pfollow ← 0.00
Step 2: Sx ← solution found at previous iteration.
Step 3: If Pfi < 0.90 * PfWL
            Pfollow ←0.60
Step 4: Generate a random number r ← [0,1].
Step 5: If r < Pfollow
            Sx ← Pick a solution from WL using a dance selec-
                tion strategy.
Step 6: Return Sx.
```

Figure 9: Algorithm for ObserveNSelectDance().

5 EXPERIMENTS AND RESULTS

In this section, the benchmark problems, benchmark algorithms and some experimental results will be presented. The experiments are conducted on a 16-node Linux cluster where each node has two Dual Core Xeon 3.0GHz processors with 4GB RAM.

5.1 Benchmark Problems

The Taillard JSSP dataset used in this paper is obtained from the library of JSSP sample instances maintained by Taillard, E. (http://mistic.heig-vd.ch/taillard/). Our experiments focus on the first 50 problem instances (ta_01-ta_50) for which only 19 optimal solutions have been found to date. The sizes of these problems range from 15 to 30 jobs

and 15 to 20 machines. The performance measure used is the percentage difference in makespan θ of a solution found, C_{max}, from the known optimum/upper bound, $C_{optimal}$, calculated as in (13):

$$\theta = \left\{ \left(C_{\max} - C_{optimal} \right) / C_{optimal} \right\} \quad (13)$$

5.2 Benchmark Algorithms

To evaluate the performance of the proposed bee colony algorithm, we include three other meta-heuristics in our experimental study. The first is the TSA by Nowicki and Smutnicki (1996) which is considered to be one of the best in the class of optimization algorithms for the JSSP. The second is the SBP for the JSSP proposed by Adams et al. (1988). The third is the BCNS by Chong et al. (2007).

5.3 Parameter Settings

As TSA and SBP are deterministic algorithms, results for these two algorithms are taken after one run. In contrast, the experimental results for BCNS and BCBV are the averages over five runs due to their stochastic characteristic.

Table 2: Parameter Setting for the Algorithms.

Parameter	TSA	SBP	BCNS	BCBV
N_{max}	∞_T	∞_T	2000	2000
Population, l	n.a.	n.a.	No. of Jobs	10
Alpha, α	n.a.	n.a.	1.0	n.a.
Beta, β	n.a.	n.a.	1.0	n.a.
Rating, ρ_{ij}	n.a.	n.a.	0.99	n.a.
Waggle dance scaling factor, A	n.a.	n.a.	100	n.a.
Probability to perform waggle dance, p	n.a.	n.a.	0.001	n.a.
Max. no. of Elite Solution	20	n.a.	20	∞
Max size of Taboo List	8	n.a.	n.a.	15
Waggle dance replacement threshold, D_R	n.a.	n.a.	n.a.	0.15
Waggle dance recruitment counter, $N_{Attempt}$	n.a.	n.a.	n.a.	50

∞_T denotes the experiments are allowed to run until termination.

Table 3: Impact of Different D_R Settings in BCBV.

D_R	θ(%)		
	Min	Max	Mean
0.05	4.88	12.17	9.11
0.15	3.84	11.13	8.02
0.25	4.54	11.36	8.38
0.35	4.49	11.14	8.92
0.45	4.41	13.17	8.86

Due to the different natures of the four algorithms, it is difficult to allocate the exact same amount of computation time and resource to each of them. For a meaningful comparison, both TSA and SBP were allowed to run until termination. Both BCNS and BCBV were allowed to run for 2000 iterations. The parameter settings for all four algorithms are presented in Table 2. The parameter settings are obtained after a series of tuning experiments.

Table 3 shows the impacts of different D_R settings on BCBV on a set of five representative problems from Taillard dataset. They include ta_04, ta_13, ta_29, ta_40, ta_41. The average θ as well as the maximum and minimum θ of the five problems are shown in the table.

D_R controls the number of distinct peaks (local optima) in the landscape stored in *WL*. When D_R is set too low (e.g. 0.05), too many solutions are stored in *WL* leading to slow convergence of the search. When D_R is set too high (e.g. 0.45), only a few solutions are stored in *WL* leading to insufficient diversification in the search. Setting D_R to 0.15 gives the best result for the five problem instances.

5.4 Experimental Results

Table 4 summarizes the performance results of the four algorithms investigated. Besides the average, minimum and maximum θ, the number of best solutions found amongst the four algorithms on the 50 problem instances is also reported. From the results presented in Table 4, TSA records the closest results to the optimal and find the best results for 37 out of 50 problem instances. In comparison with TSA, BCBV gave similar performances for both the mean and maximum θ. BCBV also obtained a lower θ and found more best solutions compared to BCNS and SBP. While both bee heuristics underperform TSA, they offer a comparable performance after 2000 iterations and their performance can be improved when more iterations are used. From the CPU time (speed) perspective, our experimental results show that BCBV is 2.6 times slower than TSA.

Table 4: Performance of TSA, SBP, BCNS and BCBV on 50 Problem Instances in Taillard JSSP Benchmark.

Category	TSA	SBP	BCNS	BCBV
Mean θ (%)	6.93	11.77	9.16	8.43
Min θ (%)	1.04	5.29	2.84	4.01
Max θ (%)	13.66	20.63	16.22	13.93
Best Solutions Found	37	0	2	11

5.5 Long Running Behaviors of BCBV

To see the long running performance of BCBV algorithm against BCNS and TSA, the averaged θ over five problems versus the number of iterations is plotted in Figure 10. The same set of five problems as mentioned in Section 5.3 is used. A snapshot is taken at every 1000-th iteration where the average θ for each algorithm is calculated.

To investigate whether TSA, BCNS and BCBV have a similar starting point, a snapshot on the average θ is taken at the initial stage. Results show that θ is 24.10% for TSA,

22.02% for BCNS, and 21.14% for BCBV, which shows that these three algorithms have similar starting points.

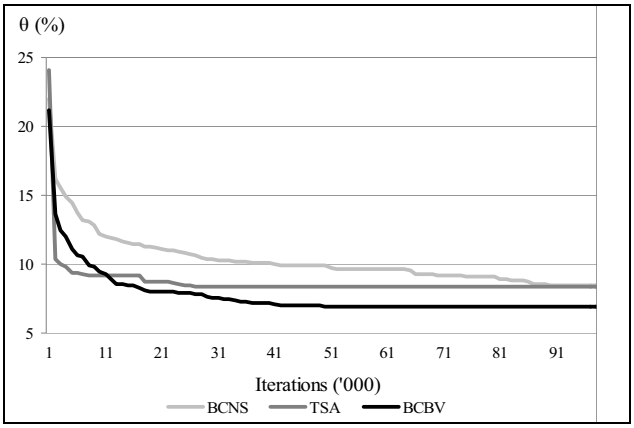

Figure 10: Long Running Behaviors of TSA, BCNS and BCBV.

The TSA algorithm converges very quickly to find its best solution, leveling off at approximately 31000 iterations and showing no performance improvement thereafter. This is due to the termination of the algorithm by its stopping condition when its elite solution list becomes empty. The graph for TSA is extended beyond its termination point for comparison with BCBV and BCNS. The BCNS algorithm converges much slower (in terms of number of iterations) and takes approximately 85000 iterations to reach the performance level of the TSA algorithm. Converging slower than TSA, the BCBV algorithm starts to outperform TSA at approximately 13000-th iteration and its solutions continued to improve before leveling off at approximately 37000-th iteration. The trend of BCBV and BCNS suggests that by taking such an evolutionary approach, the quality of the solutions obtained may improve further given a longer running time. This characteristic is important if the objective is to obtain very high quality solutions given ample computational facilities and time.

6 CONCLUSION

A Bee Colony Optimization algorithm with Big Valley landscape exploitation has been proposed to solve JSSP problem. The algorithm is based on the foraging behavior of honey bees where the waggle dance is used as a communication medium among bees. To have an effective exploitation and exploration on the landscape structures, accumulation and selection strategies are introduced on the solution list *WL* with waggle dances that represents peaks in a landscape. Experimental results on the Taillard JSSP benchmark show that BCBV is able to achieve comparable performance to TSA, SBP and BCNS using small number of iterations. Given ample computation time, BCBV is able to deliver performance better than TSA, SBP and BCNS.

For future works, we will continue to explore other features of the Big Valley landscape as well as parameter tuning to improve the performance of the BCBV algorithm. We will also incorporate other bee related features such as the direction of waggle dance and scout bees in order to make the algorithm more comprehensive.

ACKNOWLEDGMENT

The authors would like to thank Science University of Malaysia and Ministry of Higher Education of Malaysia for the scholarship awarded to Li-Pei Wong to pursue his Ph.D. in Nanyang Technological University, Singapore.

REFERENCES

Adams, J., E. Balas, and D. Zawack. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science* 34(3):391-401.

Baykosoglu, A., L. Ozbakir, and P. Tapkan. 2007. Artificial bee colony algorithm and its application to generalized assignment problem. In *Swarm Intelligence: Focus on Ant and Particle Swarm Optimization*, 532-564. Austria: Itech Education and Publishing.

Biesmeijer, J. C. and T. D. Seeley . 2005. The use of waggle dance information by honey bees throughout their foraging careers. *Behavioral Ecology and Sociobiology* 59(1):133-142.

Blazewicz, J., W. Domschke, and E. Pesch. 1996. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research* 93(1):1-33.

Boese, K. D., A. B. Kahng, and S. Muddu. 1994. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters* 16(2):101-113.

Bonabeau, E., and C. Meyer. 2001. A whole new way to think about business. *Harvard Business Review*, 106-114.

Cavalieri, S., F. Crisafulli, and O. Mirabella. 1999. A genetic algorithm for job-shop scheduling in a semiconductor manufacturing system. In *Proceedings of the 25th Annual Conference of the IEEE Industrial Electronics Society*, 957-961.

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

Chong, C. S., M. Y. H. Low, A. I. Sivakumar, and K. L. Gay. 2007. Using a bee colony Algorithm for neighborhood search in job shop scheduling problems. In *Proceedings of 21st European Conference on Modeling and Simulation (ECMS 2007)*.

Dyer, F. C. 2002. The biology of the dance language. *Annual Review of Entomology* 47, 917-949.

Grabowski, J., E., Nowicki, and S. Zdrzalka. 1986. A block approach for single-machine scheduling with release dates and due dates. *European Journal of Operational Research* 26(2):278-285.

Gupta, A. K., and A. I. Sivakumar. 2006. Job shop scheduling techniques in semiconductor manufacturing. *International Journal of Advanced Manufacturing Technology* 27(11-12):1163-1169.

Lee, C. Y., L. Lei, and M. Pinedo. 1997. Current trends in deterministic scheduling. *Annals of Operations Research* 70(0):1-41.

Lenstra, J. K., A. H. G. Rinnooy Kan, and P. Brucker. 1977. Complexity of machine scheduling problems. *Annals of Discrete Mathematics* 1:343-362.

Lucic, P., and D. Teodorovic. 2002. Transportation modeling: an artificial life approach. In *Proceedings of 14th IEEE International Conference on Tools with Artificial Intelligence 2002 (ICTAI 2002)*, 216-223.

Lucic, P. and D. Teodorovic. 2003. Computing with Bees: Attacking Complex Transportation Engineering Problems. *International Journal on Artificial Intelligence Tools* 12(3):375-394.

Nakrani, S., and C. Tovey. 2004. On honey bees and dynamic server allocation in Internet hosting centers. *Adaptive Behavior* 12(3-4):223-240.

Nowicki, E., and C. Smutnicki. 1996. A fast taboo search algorithm for the job shop problem. *Management Science* 42(6):797-813.

Nowicki, E., and C. Smutnicki. 2005. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling* 8(2):145-159.

Reeves, C. R. 1999. Landscapes, operators and heuristic search. *Annals of Operations Research* 86(0):473-490.

van Rijswijck, J. 2000. Are bees better than fruitflies? Experiments with a hex playing program. In *Advances in Artificial Intelligence*, Lecture Notes in Computer Science 1822/2000. 13-25. Berlin/Heidelberg: Springer.

von Frisch, K. 1974. Decoding the language of the bee. *Science* 185(4152):663-668.

Wedde, F. H., M. Farooq, and Y. Zhang. 2004. Beehive: An efficient fault-tolerant routing algorithm inspired by honey bee behavior. In *Ant Colony, Optimization and Swarm Intelligence*, Lecture Notes in Computer Science 3172:83-94. Berlin/Heidelberg: Springer.

Wong, L. P., M. Y. H. Low, and C. S. Chong. 2008. A bee colony optimization algorithm for traveling salesman problem. In *Proceedings of 2nd Asia International Conference on Modelling & Simulation*, 818-823.

Yamada, T. 2003. Studies on metaheuristics for jobshop and flowshop scheduling problems. Ph.D. Thesis. Kyoto University.

AUTHOR BIOGRAPHIES

LI-PEI WONG obtained his Bachelor degree and Master degree in Computer Science from Science University of Malaysia in 2001 and 2004. He is currently pursuing his PhD in Computer Science at the Nanyang Technological University (NTU), Singapore. His current research interest includes scheduling and optimization in the area of manufacturing, logistic, timetabling, and supply chain. His email address is <wonglipei@pmail.ntu.edu.sg>.

CHI YUNG PUAN is a Fourth Year Undergraduate Student currently enrolled in the Computer Science Course (CSC) at the School of Computer Engineering in Nanyang Technological University (NTU), Singapore. His current research interest is in artificial intelligence agents for the modeling and simulation and optimization of complex systems. His work and research in his Final Year Project: "Bee Colony Optimization: A Biologically Inspired Approach to Job Shop Scheduling" has been largely adapted into this paper. His email address is <puan0001@ntu.edu.sg>.

Dr. MALCOLM YOKE HEAN LOW is an Assistant Professor in the School of Computer Engineering at the Nanyang Technological University (NTU), Singapore. Prior to this, he was with the Singapore Institute of Manufacturing Technology, Singapore (SIMTech). He received his Bachelor and Master of Applied Science in Computer Engineering from NTU in 1997 and 1999 respectively. He was awarded a Gintic (now SIMTech) Postgraduate Scholarship in 1999. In 2002, he received his D.Phil. degree in Computer Science from Oxford University. His current research interest is in the application of parallel and distributed computing for the modeling, simulation, analysis and optimization of complex systems. His email address is <yhlow@ntu.edu.sg>.

Dr. CHIN SOON CHONG obtained his degree in Electrical and Electronics Engineering from the City University of London, UK. He joined Singapore Institute of Manufacturing Technology (SIMTech), and is currently in the Planning Operation Management Group. He obtained his Master of Engineering in Computer Integrated Manufacturing from Nanyang Technological University, Singapore. He has been involved in simulation, scheduling and optimization related projects in logistic and manufacturing IT domains. The projects include cargo container operation simulation, printing process simulation, manufacturing cycle-time modeling, scheduling and optimization for MNCs. His current research interest includes simulation, planning, scheduling, optimization in the area of manufacturing, logistic, and supply chain. He can be reached via email at <cschong@simtech.a-star.edu.sg>.