Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, Volume 232, Issue 8, 2018, pp. 1444 -1459
DOI: 10.1177/0954405416673095

Multi-Objective Discrete Particle Swarm Optimisation Algorithm for Integrated Assembly Sequence Planning and Assembly Line Balancing

Mohd Fadzil Faisae Ab Rashid[1, 2], Windo Hutabarat[1] and Ashutosh Tiwari[1]

[1] Manufacturing and Materials Department, Cranfield University, Bedford, UK

[2] Faculty of Mechanical Engineering, University Malaysia Pahang, Pahang, Malaysia

Abstract

In assembly optimisation, Assembly Sequence Planning (ASP) and Assembly Line Balancing (ALB) have been extensively studied because both activities are directly linked with assembly efficiency that influences the final assembly costs. Both activities are categorised as NP-hard and usually performed separately. ASP and ALB optimisation presents a good opportunity to be integrated, considering the benefits such as larger search space that leads to better solution quality, reduces error rate in planning and speeds up time-to-market for a product. In order to optimise an integrated ASP and ALB, this work proposes a Multi-Objective Discrete Particle Swarm Optimisation (MODPSO) algorithm that used discrete procedures to update its position and velocity in finding Pareto optimal solution. A computational experiment with 51 test problems at different difficulty levels were used to test the MODPSO performance compared with existing algorithms. A statistical test of the algorithm performance indicates that the proposed MODPSO algorithm presents significant improvement in terms of the quality of the solution set towards the Pareto optimal set.

Keywords

Integrated assembly sequence planning and assembly line balancing; Particle Swarm Optimisation; Multi-objective optimisation; Discrete Particle Swarm Optimisation

Published by Sage. This is the Author Accepted Manuscript issued with: Creative Commons Attribution Non-Commercial License (CC:BY:NC 4.0). This article may be used for personal use only. The final published version (version of record) is available online doi:10.1177/0954405416673095. Please refer to any applicable publisher terms of use.

1. Introduction

In the recent years, various multi-objective optimisation techniques have been proposed to solve assembly optimisation problems [1]–[3]. This trend shows the attention given by researchers to assembly optimisation activities because of their complexity and relevance of the actual industrial problems. The research in assembly optimisation can be classified according to product development and production stage [4]. Although there are many activities classified in the assembly optimisation, Assembly Sequence Planning (ASP) and Assembly Line Balancing (ALB) are the most prominent because they are directly linked with assembly efficiency.

ASP refers to a task for which planners, usually based on their particular heuristics in assembling all the components of a product, arrange a specific collision-free assembly sequence according to the product design description [5], [6]. Figure 1 shows a common assembly representation using a precedence graph. From this graph, numerous feasible assembly sequences can be generated such as {1 6 7 2 4 3 5}, {1 7 3 6 5 4 2} or {1 6 2 7 4 3 5}. Based on this example, ASP is about determining the most optimum sequence to assemble a product from all feasible assembly sequences.

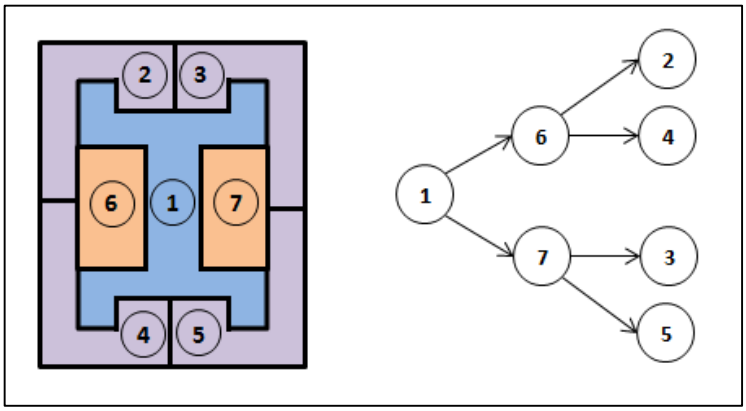


Figure 1: Assembly representation using precedence graph

Once the optimum assembly sequence has been determined from ASP activity, the assembly jobs will be assigned to workstations, so that every workstation will have equal or almost equal work load. The assembly job assignment activity refers to ALB that is defined as the decision problem of optimally

partitioning assembly work among stations with respect to some objectives and constraints [7], [8]. For example, let the optimum assembly sequence from ASP is {1 6 2 7 4 3 5} and this assembly jobs will be assigned to three workstations. There are many possible assembly job assignment combinations such as {(1 6), (2 7 4), (3 5)} or {(1 6 2), (7 4), (3 5)} or {(1 6 2 7), (4 3), (5)}. In this process, ALB will determine the best assembly job combination which comes out with equal or almost equal workload between workstations.

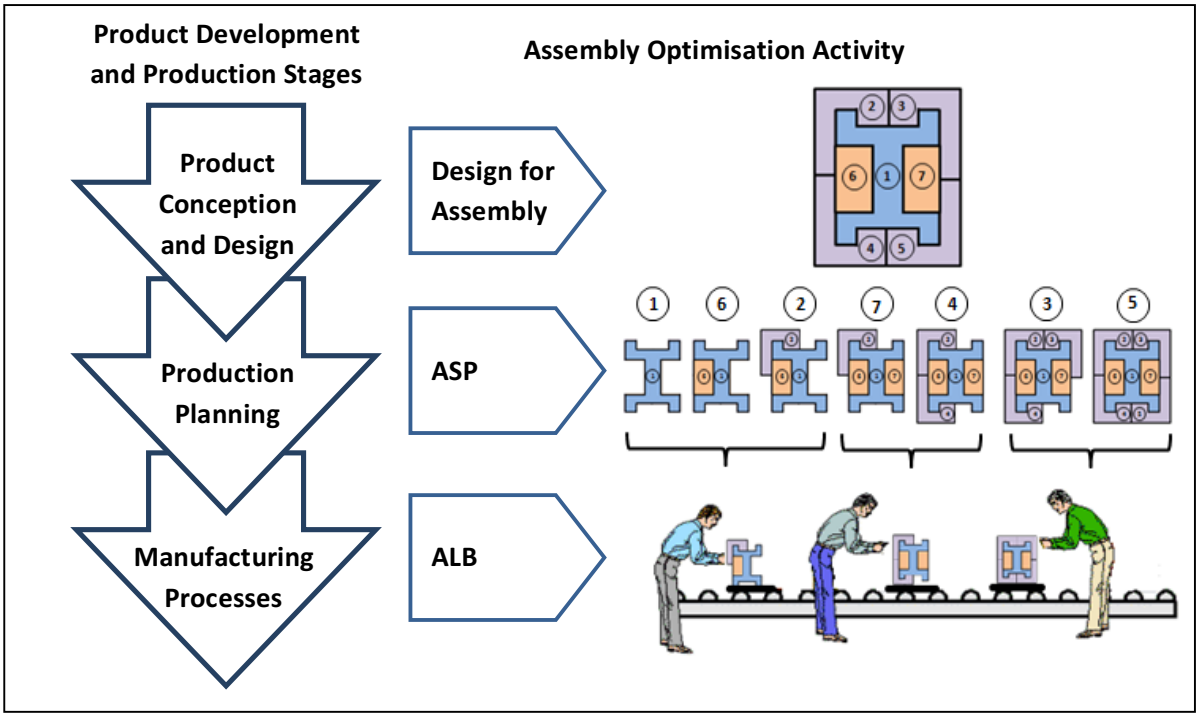


Figure 2: Main assembly optimisation activities in product development and production stages

Figure 2 shows the main assembly optimisation activities in every product development and production stages. Based on this figure, ASP and ALB activities are performed individually because they take place in different stages of the product development process. ASP is located in the production planning while ALB is in the manufacturing process stage [4]. However, current global market drives a demand for shorter product life-cycles and also for products to be more competitive in terms of time-to-market, quality and also manufacturing cost. One approach to stay competitive is by integrating manufacturing activities across the different stages [9]. From the aspect of assembly optimisation, the ASP and ALB optimisation activities present a good integration opportunity with various potential benefits to the manufacturer.

The main benefit of using an integrated ASP and ALB is that the quality of assembly plans could be improved as the search space of the integrated problem would become larger than when optimising each problem sequentially. Besides, the integrated ASP and ALB could reduce the rate of errors in both planning and costing during the manufacturing stages [5]. At the same time, the integrated optimisation also speeds up the aspect of time-to-market for a product.

In ASP and ALB optimisation, several objectives have been used to determine the optimal solution for the problem. When an optimisation problem involves more than one objective, this problem is known as multi-objective optimisation [10]. Traditionally, the simplest way to optimise multi-objective problem is to bundle all the objectives into a single fitness using some kind of weighted assignment. However, this approach requires high prior knowledge on the importance of one objective over another. Therefore, instead of focusing on one single optimum point, the researchers might be interested in all the best options available, which are known as Pareto optimal solutions [11]. Furthermore, by having a set of optimum solutions, the decision makers in the industry are offered more flexibility in selecting the solution that is deemed suitable with a variety of preferences.

Although various optimisation algorithms have been developed and used to optimise different multi-objective problems, to the best of the authors' knowledge, only techniques based on Genetic Algorithm (GA) and Ant Colony Optimisation (ACO) have been proposed to optimise integrated ASP and ALB [1], [12], [13]. Chen proposed a hybrid Genetic Algorithm to optimise integrated ASP and ALB, where GA is combined with heuristic search [14]. The objectives were to minimise cycle time, maximise workload smoothness, minimise tool changes, minimise the number of tools, and minimise the total penalty of assembly relations. Although this paper does not clearly state the integration of ASP and ALB, this relationship was acknowledged following the optimisation objectives.

Tseng and Tang studied combining ASP together with ALB based on assembly "connectors" (i.e. the connector basis) by using Genetic Algorithm. In their work, optimisation was conducted in three stages. First, each part was assigned to a specific connector type. Then, the algorithm generated the assembly planning based on connectors. Finally, the algorithm assigned the connectors to stations

and selected proper types of stations. Meanwhile, in the second and third stages, GA was applied to generate connector-based assembly in a sequential order, as well as to determine the suitable station types for the sequential order. However, when using this approach, whenever the number of connectors is increased, a few of the parameters that govern GA performance need to be reset [5].

Another work by Tseng *et al.* on integrated ASP and ALB was done in 2008. This work adopted the Hybrid Evolutionary Multi-objective Algorithms (HEMOA) that was based on GA [15]. On the basis of multi-objective optimisation, the Pareto optimal approach was adopted in this study. However, the weighted sum approach was employed to obtain a better solution, instead of measuring the solution crowding that has been frequently used in recent multi-objective algorithm.

Another integrated ASP and ALB optimisation work was also formulated based on assembly connectors [16]. In this work, the Guided-modified weighted Pareto-based multi-objective genetic algorithm (G-WPMOGA) had been proposed to optimise the integrated ASP and ALB problem with two, three, and four objectives. The proposed algorithm displayed better performance in the problem with two and four objectives, but not in the problem with three objectives.

In the recent optimisation work of integrated ASP and ALB, the GA-based algorithms performed well in optimising the problem with low and medium difficulties. However, the performance of GA-based algorithms did not last when optimising high difficulty problem, especially the problem with a large number of tasks [17]. On the other hand, the researcher also concluded that the different GA-based variants should be used in order to optimise the problem with a different number of criteria [16].

Besides GA-based algorithm, the researcher implemented Ant Colony Algorithm to optimise the integrated ASP and ALB problems [12], [13]. In [12], they proposed an optimisation model to implement ACO for integrated ASP and ALB without numerical experiment. Meanwhile in [13], the authors optimised several objectives such as line efficiency, smoothness index, assembly time and number of workstations. However, the problem was treated as single objective, by combining all the

objectives. In order to overcome the limitation, a new algorithm to optimise integrated ASP and ALB problems is needed.

In many different works that compare algorithm performance, Particle Swarm Optimisation (PSO) has shown strong performance compared with competing algorithms. This algorithm is popular due to its simplicity and ability to quickly converge to a reasonably good solution [18]. PSO is a population-based stochastic optimisation technique that was developed by Kennedy and Eberhart in 1995. In PSO, the potential solutions, called particles, 'fly' through the problem space by following the current optimum particles [19]. In micro-assembly sequence planning, PSO was found to provide less computational time compared with other considered algorithms [20]. PSO was shown to outperform GA and Simulated Annealing in scheduling problems in the majority of applications by considering computation efficiency, optimality and robustness [21]. In another research, PSO was found to perform better than GA, Memetic Algorithm, Shuffled Frog Leaping and Ant Algorithm in solving continuous and discrete optimisation problems [22]. PSO algorithm also performs better compared with commercial software for robotic assembly line balancing problem [23].

Even though several researches on ASP and ALB implemented PSO, they were independent works [24]–[27]. At this point, no existing works are using PSO algorithms to optimise the integrated ASP and ALB problems. In this paper, the Multi-Objective Discrete PSO (MODPSO) to optimise integrated ASP and ALB is proposed. In comparison with [28]–[30] that used continuous encoding in PSO, the proposed algorithm implemented discrete encoding to match with discrete combinatorial optimisation problem. On the other hand, different from Discrete PSO as found in [31]–[34], the proposed algorithm implemented non-dominated sorting concept to deal with multi-objective. The proposed MODPSO algorithm also integrates the Crowding Distance concept from Elitist Non-Dominated Sorting Genetic Algorithm (NSGA-II) [35] to determine the leaders (*Pbest* and *Gbest*).

This work was motivated by the benefits of integrating ASP and ALB, and also the expected performance gained through using PSO in optimising multi-objective problems to overcome limitation of GA-based algorithms. Section 2 explains the problem representation for integrated ASP

and ALB. Section 3 details the proposed MODPSO algorithm, followed by experimental strategy and set up in Section 4. Section 5 presents the results and Section 6 discusses result of experiments that deploy various algorithms to optimise multi-objective ASP and ALB problems. Finally, Section 7 concludes the finding from the proposed MODPSO algorithm.

2. ASP and ALB Problem Representation

In order to incorporate ASP and ALB optimisations into a single integrated optimisation, a clear prerequisite is the availability of an integrated ASP and ALB representation. For this purpose, an integrated assembly task-based representation scheme is used to represent both ASP and ALB problems [36]. In this scheme, the assembly plan is represented by a precedence graph (Figure 3) and the assembly data is presented in a data matrix (Table 1). Each node in the precedence graph represents an assembly task, while the connecting arc represents assembly precedence.

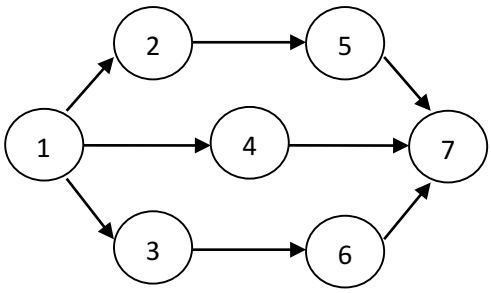


Figure 3: Example of precedence graph

Table 1: Data matrix

Task	Direction	Tool	Time
1	+*x*	*T1*	4
2	-*x*	*T2*	12
3	+*x*	*T1*	7
4	-*x*	*T3*	4
5	+*x*	*T1*	12
6	+*x*	*T1*	5
7	-*x*	*T2*	12

However, the precedence graph used to represent the assembly plan needs to be coded into a numerical format for computational application. For this purpose the precedence matrix is used to characterise assembly precedence graph. Precedence matrix is an $n \times n$ matrix that consists of 0 and 1 value. The precedence matrix in Table 2 represents the assembly precedence graph in Figure 3. In this matrix, the value 0 shows no precedence relation between task i and task j, while the value 1 shows that the task i must be performed prior to task j.

Table 2: Precedence matrix

i	j						
	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	1	1	1	0	0	0
2	0	0	0	0	1	0	0
3	0	0	0	0	0	1	0
4	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1
6	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0

2.1 Objective Function

Various objective functions have been designed and used to optimise ASP and ALB problems. A prior literature survey has collated objective functions that have been used by researchers in both problems [1]. This survey also found that the most frequently used ASP optimisation objectives are to minimise assembly direction change and number of tool change. In ALB works, even though there were various objectives such as maximising worker efficiency and equipment [37], the dominant optimisation objectives are to minimise cycle time, number of workstation and workload variance.

Number of assembly direction change (n_{dc}) is counted when the next assembly task requires a different assembly direction compared with the present assembly task. In Eq. 1 and 2, s refers to the position of a task in a feasible assembly sequence.

$$n_{dc} = \sum_{s=1}^{n-1} d_s; \qquad d_s = \begin{cases} 1 & \text{if direction } s \neq \text{direction } s+1 \\ 0 & \text{if direction } s = \text{direction } s+1 \end{cases} \qquad \text{Eq. 1}$$

Number of assembly tool change (n_{tc}) is also counted when the next assembly task requires a different assembly tool compared with the present assembly task.

$$n_{tc} = \sum_{s=1}^{n-1} t_s; \qquad t_s = \begin{cases} 1 & \text{if tool } s \neq \text{tool } s+1 \\ 0 & \text{if tool } s = \text{tool } s+1 \end{cases} \qquad \text{Eq. 2}$$

Cycle time (*ct*) is the time intervals at which product units must be finished in order to meet demand [38]. In this case, *ct* for particular assembly sequence is the highest processing time among all workstations. Processing time (*pt*) refers to the total assembly time in a particular workstation. Once the total processing time for the current workstation is larger than maximum allowable cycle time (ct_{max}), the present assembly task will be assigned to the next workstation. Normally, ct_{max} is determined from the number of demand or required output in the assignment period.

Number of workstation (*nws*) can be determined after all the assembly tasks were assigned to workstations. Once completed, the number of generated workstation is used as the fourth objective. The number of workstation depends on the cycle time where larger cycle time leads to a smaller number of workstation and vice versa.

Workload variation (*v*) calculates the average of idle time in workstations. In this case, a smaller workload variation shows that the assembly line has an almost equal load between workstations.

$$v = \frac{\sum_{i=1}^{nws} (ct - pt_i)}{nws} \qquad \text{Eq. 3}$$

3. Proposed Multi-objective Discrete PSO (MODPSO)

Various version of PSO algorithm has been proposed to optimise multi-objective problems for independent ASP and ALB [28], [30], [33], [39], [40]. One of the PSO versions for optimising multi-objective problems is known as Discrete PSO (DPSO) that was first proposed by Rameshkumar

for scheduling problem [31] and later adopted to optimise ASP problem [40]. However, in these works, the multi-objective problem was handled by bundling all objectives into a single objective that leads to only one solution. This approach required high-quality prior knowledge and experience on the importance of an objective compared with others. Another PSO version called Multi-Objective PSO (MOPSO) was proposed by Coello and Lechuga with the objective to extend the application of PSO for multi-objective problem [29]. This algorithm uses the original PSO operators for generating new particle position and velocity, but uses the non-dominated approach to find the set of optimum solutions.

In order to treat the problem as real multi-objective optimisation, this work proposed to apply the Pareto-based approach in the proposed algorithm. In PSO, the potential solution is represented by a particle, which brings three important vectors; particle position (X_i), particle velocity (V_i) and particle best solutions (*Pbest*). Figure 4 shows the working flow of Multi-Objective DPSO (MODPSO) algorithm.

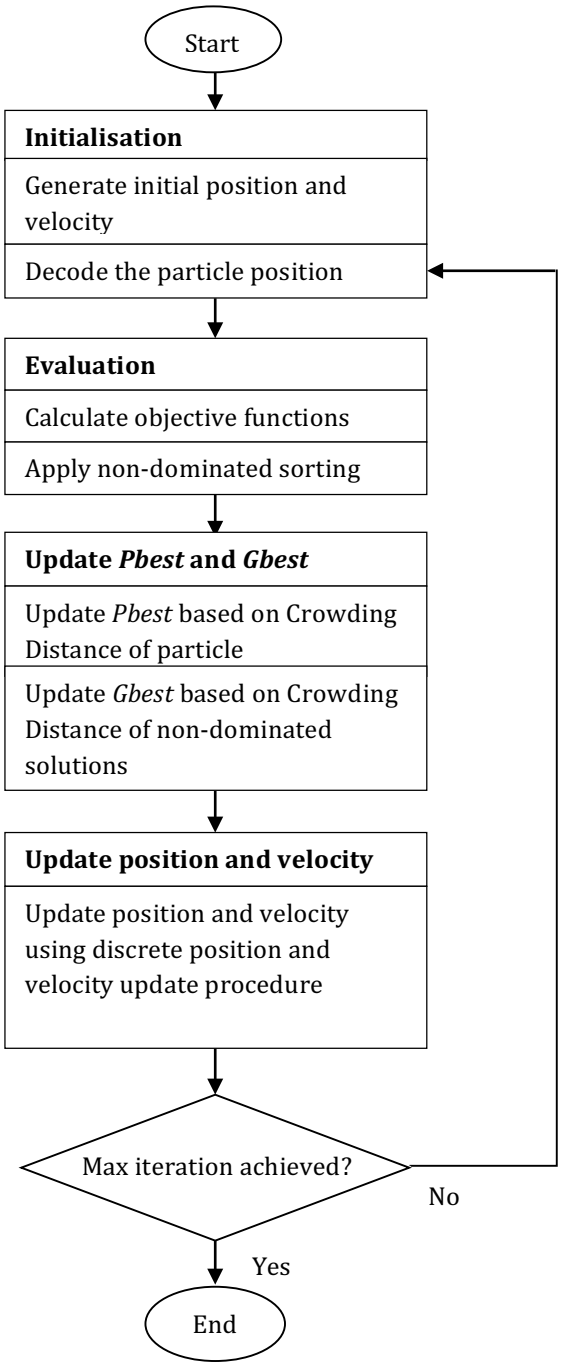


Figure 4: Flow chart of the MODPSO

3.1 Initialisation

The number of particle (*npar*) and maximum number of iteration ($iter_{max}$) was set in this step. Then the initial population known as swarm was produced by generating the *npar* set of initial position (*X*) and velocity (*V*) that consists of permutation of integer from 1 to *n* in random orders. Next, the swarm was decoded to generate feasible sequences according to the precedence constraint

using topological sort procedure. Topological sort is an approach to establish feasible sequence by selecting only one available assembly task in each iteration. The topological sort procedure is presented as follow [41].

Procedure: Topological Sort
Begin

n; number of tasks

$st = 0$; number of selected task

While $st \leq n$

- Establish available set
- $st = st + 1$
- Select one task from available set and place in st^{th} position of feasible sequence
- Remove all outgoing arcs from selected task
- Eliminate selected task from precedence graph

End While

End Procedure

In the procedure above, the available set consists of tasks without incoming arc. Then, one of the tasks in available set was selected using a predetermined selection rule. There are a few selection rules regularly used such as random selection, weight-based selection and ordered-based selection. Next, all the outgoing arcs from selected task were removed and the selected task was eliminated from the graph to avoid selecting similar task. In this work, the selection rule for topological sort was set to follow the ordered-based selection. It means that the first available task found in the particle order will be selected to be placed in the feasible sequence.

3.2 Evaluation

In this step, the decoded feasible sequence was evaluated by using the predefined objective functions. The objective functions were calculated using procedures and formulas in Eq. 1 - 3. Next, the non-dominated sorting was applied to establish the Pareto set solution. This approach is adopted from Deb in 2002 [10]. The Pareto set was updated in every iteration by evaluating each particle with solution in the Pareto set.

3.3 Update Pbest and Gbest

Pbest is the best personal particle solution while *Gbest* is the best solution for all particles. To evaluate and determine the *Pbest* and *Gbest*, a mechanism to select the best solution within all particles is needed. For this purpose, Crowding Distance (*CD*) that provides the estimation of solutions density surrounding that solution was used. For *Pbest*, the *CD* was calculated within the solution in the swarm, whereas the *CD* for *Gbest* was calculated within the Pareto set. The following algorithm was used to calculate the *CD* of each point in the set *R* [10].

Crowding Distance Calculation Procedure:

Step 1 Call the number of the solution in *R* as l = $|R|$. For each *i* in the set, assign $d_i = 0$.

Step 2 For each objective function m = 1, 2,…, *M*, sort the set in a descending order of r_m.

Step 3 For *m* = 1, 2,…, *M*, assign maximum (max_m) and minimum (min_m) value for each objective *m*.

Step 4 Calculate d_i^m for each objective m for solution *i*.

$$d_i^m = \left(\frac{I_{up^i}^m - I_{low^i}^m}{max_m - min_m}\right) \qquad \text{Eq. 4}$$

Step 5 Calculate summation of d_i^m.

$$CD_i = \sum_{m=1}^{M} d_i^m \qquad \text{Eq. 5}$$

In Eq. 4, $I_{up^i}^m$ is the nearest upper m^{th} objective value for solution *i*. Meanwhile, $I_{low^i}^m$ represents the nearest lower m^{th} objective value for solution *i*. In this case, if the objective value is located at the first or last place in the r_m, the max_m and min_m value is used to replace the nearest value respectively.

For *Pbest*, if the current particle has larger *CD* compared to the existing *Pbest*, the *Pbest* is replaced with the current position; otherwise, the existing *Pbest* is reused. Meanwhile, *Gbest* was selected as the highest *CD* among all Pareto solutions. In the proposed MODPSO, *Gbest* does not

represent the most optimum solution as in traditional PSO, but it will be the leader to update the swarm position and velocity for the next iteration.

3.4 Update Position and Velocity

The final step in MODPSO is to update swarm position and velocity. The purpose of this step is to establish new swarm set that follows the current *Pbest* and *Gbest*. In the original PSO, the position and velocity are updated using the following formulas:

$$X_i^{t+1} = X_i^t + V_i^{t+1} \quad \text{Eq. 6}$$

$$V_i^{t+1} = c_1 V_i^t + c_2\left(Pbest_i^t - X_i^t\right) + c_3(Gbest_t - X_i^t) \quad \text{Eq. 7}$$

All the operations in Eq. 6 and 7 can be easily performed for the continuous problem. However, for the discrete problem, the following discrete position and velocity update procedure were proposed to replace the original operations [31], [33].

Subtraction operator (position – position): This operation was found in Eq. 7, $\left(Pbest_i^t - X_i^t\right)$ and $(Gbest_t - X_i^t)$ and produced the velocity. Let $X_1^t = [x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}, x_{1,5}, x_{1,6}, x_{1,7}]$, $X_2^t = [x_{2,1}, x_{2,2}, x_{2,3}, x_{2,4}, x_{2,5}, x_{2,6}, x_{2,7}]$ and $V_1^t = X_1^t - X_2^t$. In this case, if x_1 and x_2 in the jth position are equal, then the $v_1 = 0$. Otherwise, $v_1 = x_1$.

Addition operator (position + velocity): For the addition of position and velocity in Eq. 6, if the jth element of velocity (v_j) is equal to zero, the jth position value (x_j^t) is inserted into the jth element of the new position (x_j^{t+1}). In the meantime, if v_j is nonzero and does not appear in the new position, then $x_j^{t+1} = v_j$. Otherwise, x_j^{t+1} is equal to zero.

Multiplication operator (coefficient + velocity): This operation was performed to make an adjustment on the influence of *Pbest* and *Gbest* on the new velocity. This operation can be represented as $V_2 = c \times V_1$, where coefficient $c \in [0, 1]$ is used to control the effect of V_1 that inherit in V_2. For this purpose, a random number, $rand \in [0, 1]$ is generated. If $rand < c$, $v_2 = v_1$, or else, $v_2 = 0$. In this work, the coefficient c_1, c_2 and c_3 were set at 0.7.

Addition operator (velocity + velocity): This operation was performed to sum up the velocities in Eq. 7. For new velocity, $V = V_1 + V_2$, the jth element of V can be derived as follows:

$$v_j = \begin{cases} v_{1,j} \text{ if } v_{1,j} \neq 0, v_{2,j} = 0 \\ v_{1,j} \text{ if } v_{1,j} \neq 0, v_{2,j} \neq 0, r < cp \\ v_{2,j} \text{ otherwise} \end{cases} \quad \text{Eq. 8}$$

In Eq. 8, r is a random number between 0 and 1, while $cp \in [0, 1]$ is inheriting constant that influences either v_1 or v_2 into new velocity.

Table 4 presents the comparison of the proposed MODPSO with NSGA-II, DPSO and MOPSO algorithms in terms of major algorithm stages. In general, the MODPSO algorithm applied similar strategies with NSGA-II for Initialisation, Evaluation and Selection stages, but different in Regeneration stage. The MODPSO Regeneration strategy used discrete position and velocity update procedure that was adopted from DPSO algorithm. Meanwhile, indifferent with MOPSO, the proposed MODPSO used different Selection and Regeneration strategies to handle multi-objective problem.

Table 3: Comparison of the NSGA-II, DPSO, MOPSO and the proposed MODPSO

Algorithm stage	**NSGA-II**	**DPSO**	**MOPSO**	**MODPSO**
Initialisation	Random initial population	Random initial particles	Random initial particles	Random initial particles
Evaluation	Individual fitness evaluation	Weighted based evaluation	Individual fitness evaluation	Individual fitness evaluation
Selection	Best Crowding Distance of non-dominated solution	Best weighted fitness	Random selection from less density hypercube of non-dominated solution	Best Crowding Distance of non-dominated solution
Regeneration	Crossover and mutation operators	Discrete PSO procedure to update position and velocity	Standard PSO operators to update position and velocity	Discrete PSO procedure to update position and velocity

4. Experimental Design

In order to test the proposed MODPSO, the experimental design was set up. The main purpose of this experiment was to test the performance of the proposed MODPSO compare with other algorithms using a set of wide range of problem difficulties. In previous work, a tuneable test problem generator for ASP and ALB has been developed [17]. The results indicate that the ASP and ALB problem difficulties can be increased using larger number of tasks (*n*), lower Order Strength (*OS*), lower Time Variability Ratio (*TV*) and higher Frequency Ratio (*FR*).

For experimental purpose, each of input variables was divided into five levels from low to high difficulty as in Table 5. Then a reference variable setting (datum) was selected as a baseline, while the rest of the problem variable setting were generated by changing only one variable value at a time. In total, there were 17 test problems (including reference setting) generated from one reference variable setting. In order to confirm algorithm performance, three different reference variable settings were used (Level 1, 3 and 5). Therefore, the complete number of test problem that involved in this experiment was 51 problems as shown in Table 6. The bolded problem setting (Problem 1, 18 and 35) represented the reference variable setting for Level 1, 3 and 5 respectively.

Table 4: Level of tuneable input setting

Level	*n*	*OS*	*TV*	*FR*
1	15	0.6	8	0.2
2	20	0.5	6	0.3
3	40	0.4	4	0.4
4	60	0.3	3	0.6
5	80	0.2	2	0.8

Table 5: Experimental design for integrated ASP and ALB

Test Problem Variable for Reference Setting at Level 1					Test Problem Variable for Reference Setting at Level 3					Test Problem Variable for Reference Setting at Level 5				
Problem	*n*	*OS*	*TV*	*FR*	Problem	*n*	*OS*	*TV*	*FR*	Problem	*n*	*OS*	*TV*	*FR*
1	**15**	**0.6**	**8**	**0.2**	**18**	**40**	**0.4**	**4**	**0.4**	**35**	**80**	**0.2**	**2**	**0.8**
2	20	0.6	8	0.2	19	15	0.4	4	0.4	36	15	0.2	2	0.8
3	40	0.6	8	0.2	20	20	0.4	4	0.4	37	20	0.2	2	0.8
4	60	0.6	8	0.2	21	60	0.4	4	0.4	38	40	0.2	2	0.8
5	80	0.6	8	0.2	22	80	0.4	4	0.4	39	60	0.2	2	0.8
6	15	0.5	8	0.2	23	40	0.6	4	0.4	40	80	0.6	2	0.8
7	15	0.4	8	0.2	24	40	0.5	4	0.4	41	80	0.5	2	0.8
8	15	0.3	8	0.2	25	40	0.3	4	0.4	42	80	0.4	2	0.8
9	15	0.2	8	0.2	26	40	0.2	4	0.4	43	80	0.3	2	0.8
10	15	0.6	6	0.2	27	40	0.4	8	0.4	44	80	0.2	8	0.8
11	15	0.6	4	0.2	28	40	0.4	6	0.4	45	80	0.2	6	0.8
12	15	0.6	3	0.2	29	40	0.4	3	0.4	46	80	0.2	4	0.8
13	15	0.6	2	0.2	30	40	0.4	2	0.4	47	80	0.2	3	0.8
14	15	0.6	8	0.3	31	40	0.4	4	0.2	48	80	0.2	2	0.2
15	15	0.6	8	0.4	32	40	0.4	4	0.3	49	80	0.2	2	0.3
16	15	0.6	8	0.6	33	40	0.4	4	0.6	50	80	0.2	2	0.4
17	15	0.6	8	0.8	34	40	0.4	4	0.8	51	80	0.2	2	0.6

The MODPSO for integrated ASP and ALB problem was coded using MATLAB software. For the performance comparison purpose, six other algorithms used to optimise integrated ASP and ALB are as follows.

i. Ant Colony Optimisation (ACO): This algorithm that has been used for simple assembly line balancing problem in Bautista and Pereira [42]. On the other hand, the ACO algorithm also has been used in integrated ASP and ALB [13]. This algorithm was selected based on its popularity.

ii. Hybrid Genetic Algorithm (HGA): HGA has been proposed by Chen and selected based on citation popularity for integrated ASP and ALB optimisation [14].

iii. Multi-Objective Genetic Algorithm (MOGA): This algorithm used in [43] to optimise ASP problem was chosen because genetic algorithm is one of the most frequently used algorithms for solving and optimising ASP problems [1]. In common with this work, it used task-based representation for ASP problem.

iv. Elitist Non-Dominated Sorting Genetic Algorithm (NSGA-II): NSGA-II was introduced by Deb *et. al* in 2002 [35]. This algorithm was selected because of its popularity in multi-objective optimisation.

v. Multi-Objective Particle Swarm Optimisation (MOPSO): The MOPSO acronym was introduced by Coello and Lechuga to extend the PSO application for Pareto-based multi-objective optimisation instead of weighted-based approach in earlier version [29].

vi. Discrete Particle Swarm Optimisation (DPSO): DPSO proposed by Rameshkumar for discrete problem. Instead of using normal mathematical operation to update position and velocity in PSO, this algorithm introduced special procedure to incorporate the discrete problem [31].

The originality of all the above algorithms was retained, as proposed by the researchers. For example, the HGA and the MOGA that were previously encoded by using permutation chromosome had been directly applied to the integrated ASP and ALB. Meanwhile, the ACO algorithm functioned by constructing the assembly sequence according to the pheromone level. Hence, different assembly sequences were generated by controlling the amount of pheromone level. Therefore, no modification was required to suit the algorithm to be integrated with ASP and ALB. Besides, the DPSO was also directly implemented because it was purposely proposed to address combinatorial problem.

On the other hand, the NSGA-II and the MOPSO were originally proposed to combat the continuous optimisation problem. In order to fit both algorithms to be integrated with ASP and ALB, the continuous value of chromosome/particle position had been defined as weight to determine the sequence of assembly task. For instance, the particle positioned at X_1 = [0.35, 7.27, 2.41, 6.38, 2.12] was decoded into X_1' = [2 4 3 5 1], giving priority to the larger position value. With this approach, the originality of NSGA-II and MOPSO had been successfully preserved since the original chromosome/particle was directly evaluated.

In this work, the population or swarm size was set at 20 with 500 iterations. For each problem, 30 simulation runs with different random seeds were performed and the output from each run was gathered and filtered to get the non-dominated solution.

4.1 Performance indicators

To evaluate the performance of each algorithm when dealing with different complexity problems, the following performance indicators adopted from [10] and [44] were used.

i. Number of non-dominated solution in Pareto optimal, $\tilde{\eta}$: Show the number of non-dominated solution generated by each algorithm in Pareto solution. The higher $\tilde{\eta}$ shows better algorithm performance.

ii. Error Ratio, *ER*: *ER* count the number of solution which is not members of the Pareto optimal set divide with number of solution that generated by algorithm q. Smaller *ER* shows better algorithm performance.

iii. Generational Distance, *GD*: *GD* finds an average distance of solution with the nearest Pareto optimal solution. Smaller *GD* provides better algorithm performance.

$$GD_q = \frac{\sum_{i=1}^{s_q} d_i}{s_q} \qquad \text{Eq. 9}$$

s_q – number of solution generated by algorithm q

$$d_i = min_{k=1}^{P} \sqrt{\sum_{m=1}^{M} (f_m^{(i)} - f_m^{*(k)})^2} \qquad \text{Eq. 10}$$

Where $f_m^{(i)}$ is m^{th} objective function value of solution i and $f_m^{*(k)}$ is the mth objective function value of kth member of Pareto optimal.

iv. *Spacing*: This indicator measures the relative distance between each solution.

$$Spacing = \sqrt{\frac{1}{N} \sum_{i=1}^{s_q} (d_i - \bar{d})^2} \qquad \text{Eq. 11}$$

where d_i is the distance between solution i and the nearest solution, while $\bar{d}$ is the average of all d_i. The smaller *Spacing* index shows better solution and has better space between each solution.

v. Maximum Spread, $Spread_{max}$: Measures the spread of solution found by each algorithm. The larger maximum spread is the better.

$$Spread_{max} = \sqrt{\sum_{i=1}^{M}(\min f_i - \max f_i)^2} \qquad \text{Eq. 12}$$

5. Experimental Results

Figure 5 shows the number of the non-dominated solution in Pareto optimal ($\tilde{\eta}$) for all test problems using different algorithms. This figure shows that the proposed MODPSO performed better than other algorithms in all test problems. In the majority of test problems, there was a significant gap between MODPSO and other algorithms in term of $\tilde{\eta}$ found. According to the output pattern, larger problem size will come out with broader gaps between MODPSO and other algorithms.

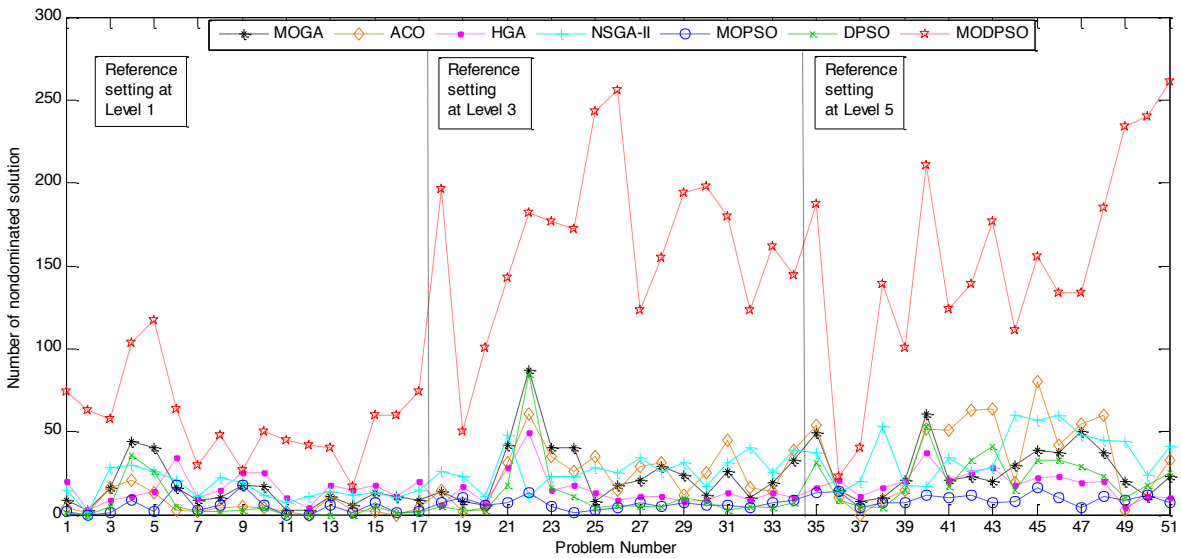


Figure 5: Number of non-dominated set throughout test problems

The Error Ratio (ER) for all algorithms is presented in Figure 6. From 51 test problems, MODPSO algorithms performed better in 82% of the problems. While the remaining 18% of problems was led by NSGA-II, where most of these problems involved larger task numbers (60 and 80 tasks). However, the mean of *ER* using MODPSO for all test problems remained the smallest (0.34) compared with NSGA-II (0.57) and other algorithms (between 0.81-0.93).

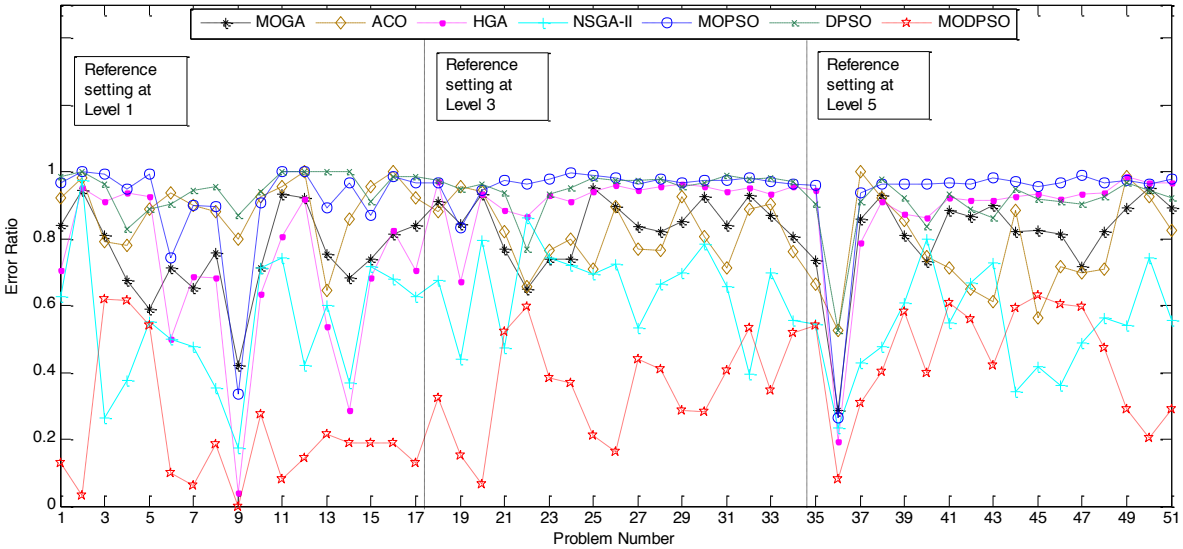


Figure 6: Error ratio throughout test problems

Meanwhile, Figure 7 presents the Generational Distance (*GD*) for algorithms throughout the test problems. For this indicator, MODPSO also performed better in 82% of the problems in almost similar problems as in *ER*. This is because the *GD* was measured between the solutions with the nearest Pareto solution. When the number of the non-Pareto solution increased (higher *ER*), the average of distance to Pareto solution would also increase. However, it still depends on how far the distance of non-Pareto with the Pareto solution. For example in problem 22, although the *ER* using MODPSO was better than other algorithms, the *GD* using this algorithm was in the third position after MOGA and ACO. It shows that the distance of non-Pareto solution in MODPSO was relatively larger than MOGA and ACO, since these algorithms also produced good $\tilde{\eta}$ for a particular problem.

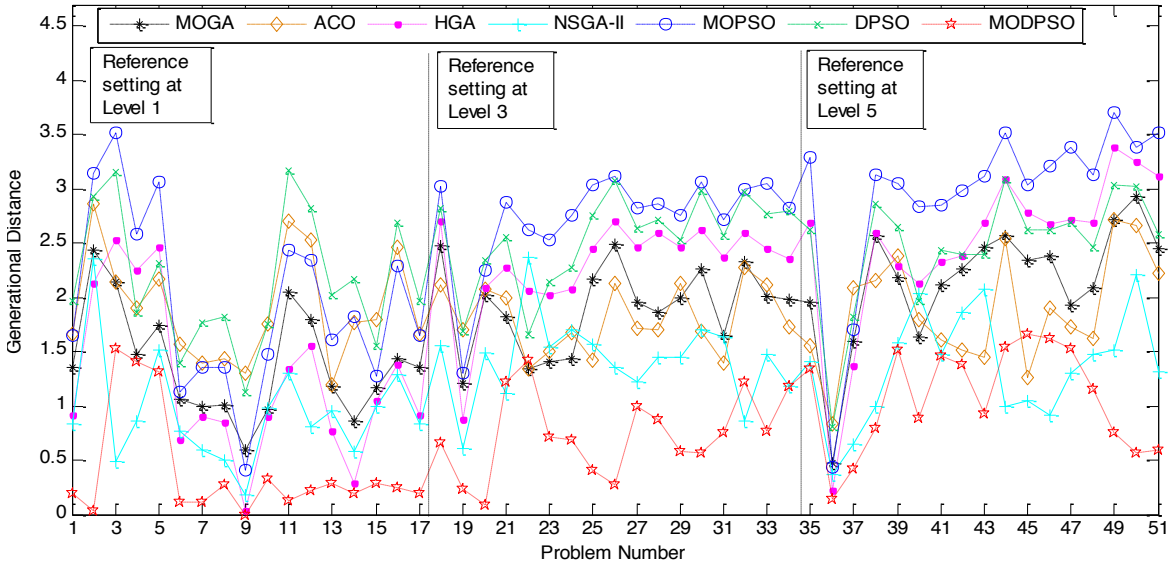


Figure 7: Generational distance throughout test problems

Figure 8 shows the performance of *Spacing* indicator that leads to different algorithms. For this indicator, MOPSO algorithm performed better in 37% of test problems. Then it was followed with MODPSO (22%), HGA (18%), DPSO (15%), MOGA (6%) and ACO (2%).

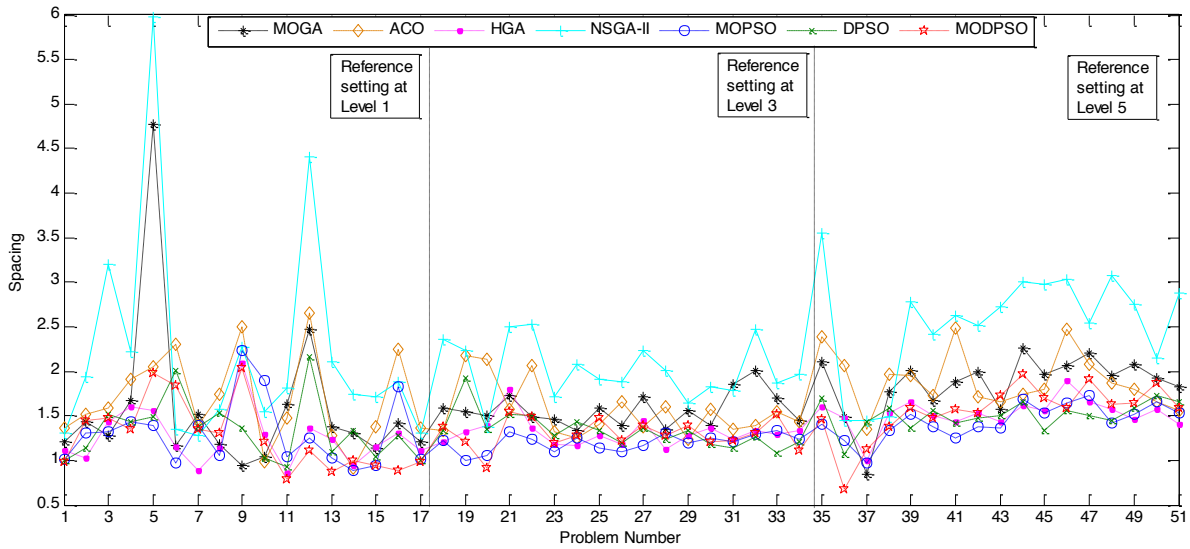


Figure 8: Solution *Spacing* throughout test problems

For Maximum Spread ($Spread_{max}$) in Figure 9, all algorithms show almost similar graph pattern with small gaps between one another. For this indicator, MODPSO algorithm performed better

in 71% of test problems. In this case, MODPSO achieved better performance in the problem with the larger number of tasks, as it performed better in all test problems with 60 and 80 tasks.

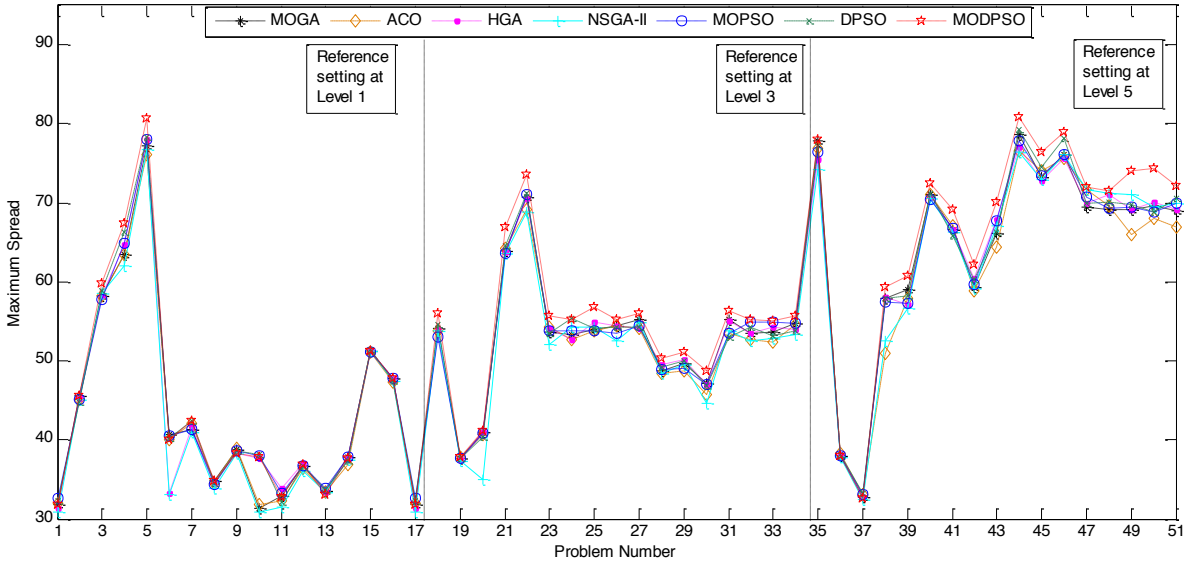
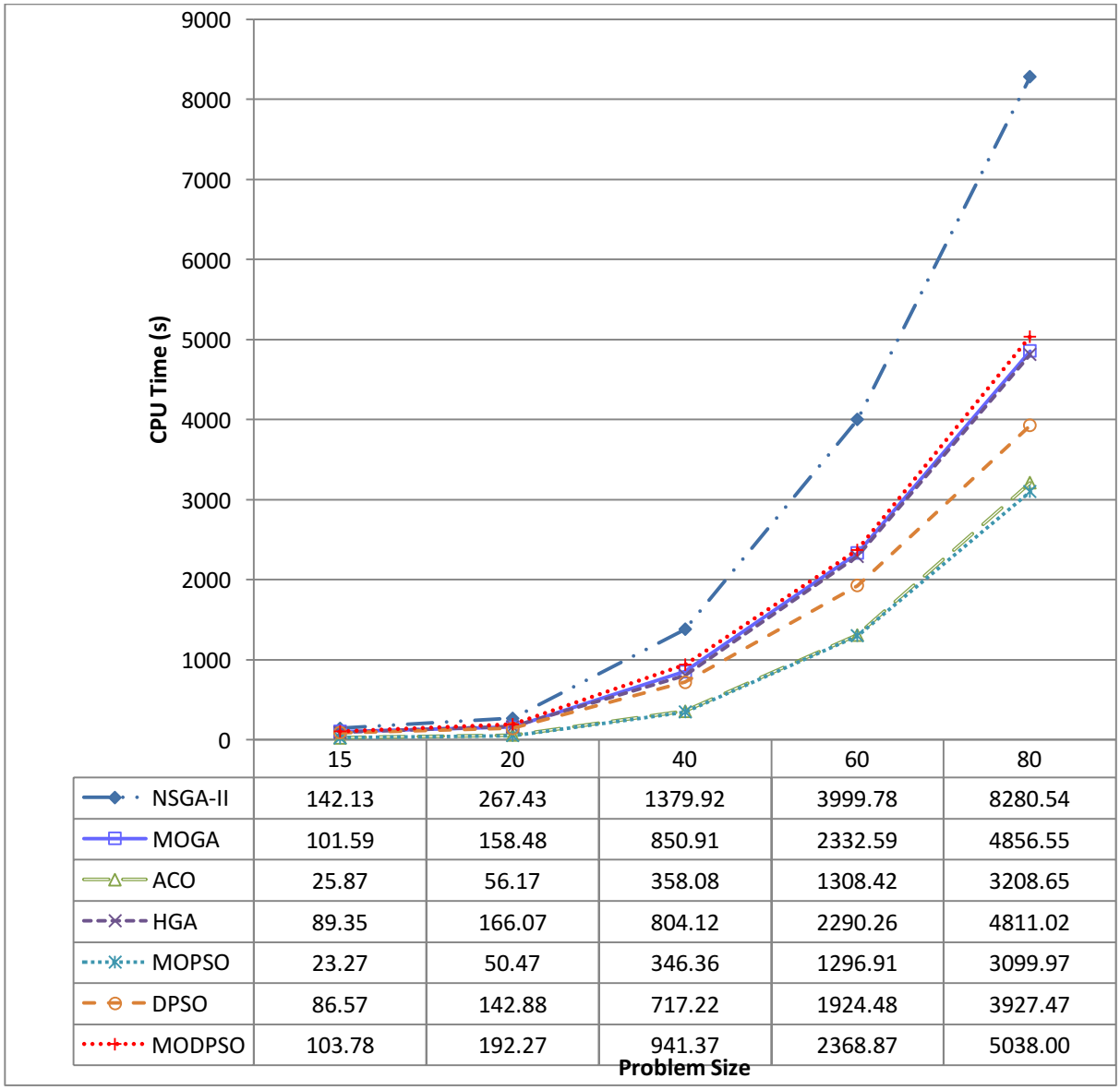


Figure 9: Maximum spread throughout test problems

Figure 10 shows the average CPU time to complete 500 iterations for different algorithms. Based on the results, NSGA-II consistently required the highest computational time compared with other algorithms. This is related with NSGA-II feature which combined the parent and offspring chromosomes in the evaluation stage. In other words, the number of evaluated chromosomes in NSGA-II was doubled compared with other algorithms. On the other hand, MOPSO algorithm was the fastest due to the basic updating procedures implemented in this algorithm. The proposed MODPSO meanwhile, was the second highest computational time behind the NSGA-II. Besides implementing the discrete updating procedure, the proposed algorithm also required additional time to adopt the non-dominated sorting concept and calculate Crowding Distance for the leaders.



	15	20	40	60	80
NSGA-II	142.13	267.43	1379.92	3999.78	8280.54
MOGA	101.59	158.48	850.91	2332.59	4856.55
ACO	25.87	56.17	358.08	1308.42	3208.65
HGA	89.35	166.07	804.12	2290.26	4811.02
MOPSO	23.27	50.47	346.36	1296.91	3099.97
DPSO	86.57	142.88	717.22	1924.48	3927.47
MODPSO	103.78	192.27	941.37	2368.87	5038.00

Figure 10: Average CPU time

6. Discussion of Results

Table 6 presents the mean of performance indicators obtained using different reference variable settings. The number in bracket presents the algorithm ranking based on the mean of each indicator. According to this table, the MODPSO algorithm consistently performed better in $\tilde{\eta}$, *ER*, *GD* and $Spread_{max}$ for all the reference settings. Meanwhile, for *Spacing* indicator, HGA and MOPSO algorithms showed better performance than MODPSO. Based on this table, the proposed MODPSO algorithm came out with better performance in all indicators except in *Spacing* indicator.

In *Spacing* indicator, all non-dominated solutions found by the particular algorithm were taken into account, regardless of Pareto or non-Pareto solutions. This indicator showed the uniformity of the space between one solution and the nearest other. Thus, the algorithm that generated more non-dominated solutions has greater chances to produce better (smaller) *Spacing*. However, the solution distribution is more important in achieving better *Spacing*. This is because the solutions that are only distributed to particular side/s of solution space will have better *Spacing* compared with solutions that are distributed uniformly over the entire solution space, even though the number of non-dominated solution is much smaller. As an example in problem 3, the number of non-dominated solution found using MODPSO and MOGA were 152 and 84 respectively, but MOGA came out with better *Spacing* compared with MODPSO. Figure 11 shows the scatter-plot matrix for problem 3 using both algorithms. From these figures, the MODPSO solution was distributed in larger solution space compared with MOGA.

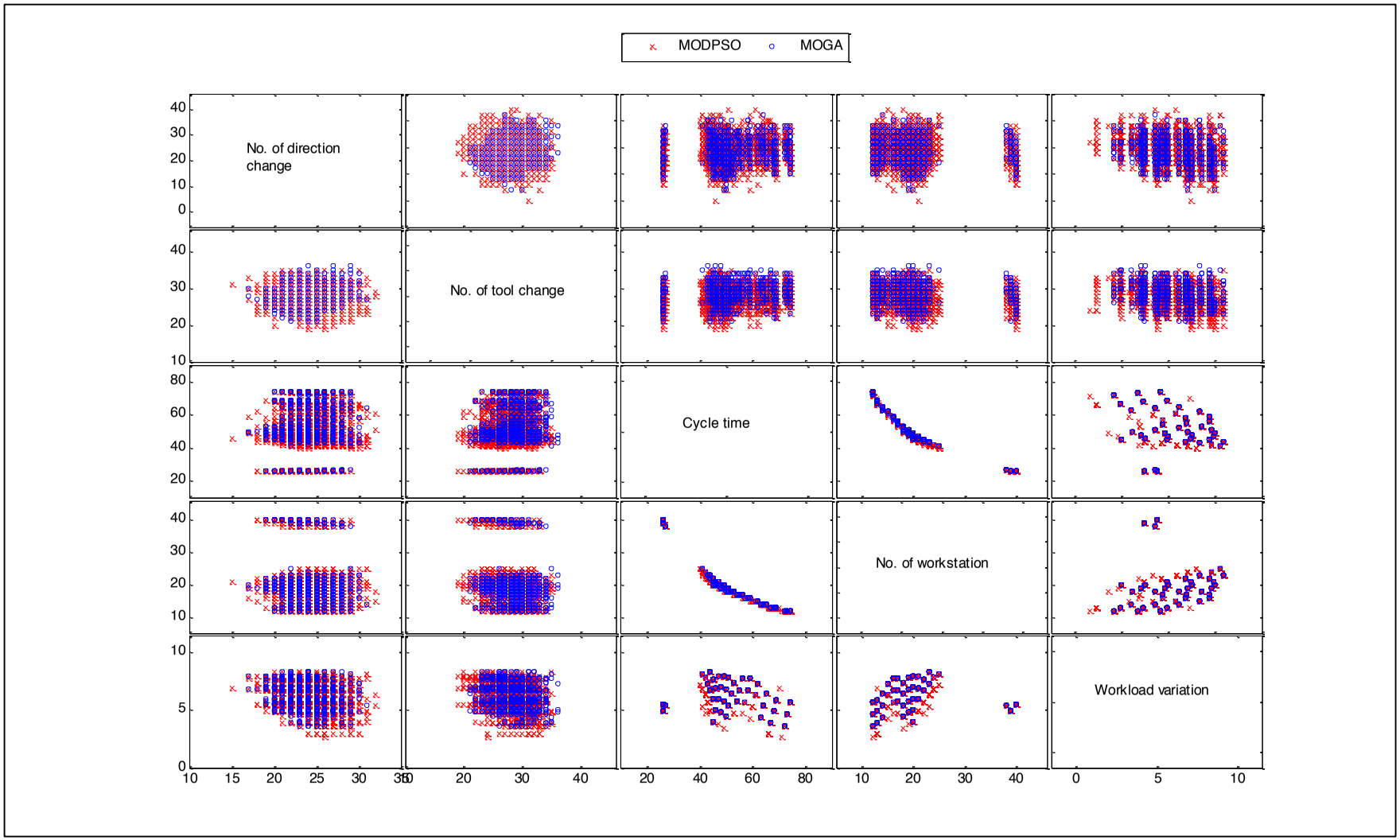


Figure 11: Scatter plot matrix for problem 3 using MOGA and MODPSO algorithms

Table 6: Mean of performance indicators by the different reference setting

*(Number in bracket shows algorithm ranking based on mean indicator value)

Reference setting	Indicator	Algorithms						
		MOGA	ACO	HGA	NSGA-II	MOPSO	DPSO	MODPSO
Level 1	$\tilde{\eta}$	14.0000 (4)	5.5882 (5)	15.4118 (3)	15.7059 (2)	4.8235 (7)	5.3529 (6)	57.2353 (1)
	ER	0.7530 (4)	0.8906 (5)	0.6901 (3)	0.5386 (2)	0.9034 (6)	0.9504 (7)	0.2174 (1)
	GD	1.3890 (4)	1.9020 (5)	1.2328 (3)	0.9307 (2)	1.9463 (6)	2.1462 (7)	0.4029 (1)
	Spacing	1.5743 (5)	1.6869 (6)	1.2509 (1)	2.2123 (7)	1.2932 (3)	1.3366 (4)	1.2690 (2)
	$Spread_{max}$	43.2394 (4)	43.0415 (6)	43.2337 (5)	42.2759 (7)	43.7500 (3)	43.8658 (2)	44.1101 (1)
Level 3	$\tilde{\eta}$	25.7059 (3)	25.5882 (4)	14.4706 (5)	27.1176 (2)	6.2941 (7)	11.6471 (6)	164.6471 (1)
	ER	0.8419 (4)	0.8210 (3)	0.9222 (5)	0.6531 (2)	0.9647 (7)	0.9544 (6)	0.3533 (1)
	GD	1.9089 (4)	1.8061 (3)	2.3055 (5)	1.4281 (2)	2.7429 (7)	2.5469 (6)	0.7438 (1)
	Spacing	1.5697 (6)	1.5558 (5)	1.3120 (3)	2.0204 (7)	1.2011 (1)	1.3345 (4)	1.3033 (2)
	$Spread_{max}$	52.9868 (2)	52.4293 (6)	52.9695 (4)	51.8761 (7)	52.8396 (5)	52.9752 (3)	54.5288 (1)
Level 5	$\tilde{\eta}$	27.9412 (4)	36.7059 (3)	18.9412 (6)	36.7647 (2)	9.5882 (7)	23.1765 (5)	152.7059 (1)
	ER	0.8076 (4)	0.7638 (3)	0.8756 (5)	0.5318 (2)	0.9247 (7)	0.8939 (6)	0.4460 (1)
	GD	2.1575 (4)	1.8858 (3)	2.4961 (6)	1.3660 (2)	2.9583 (7)	2.4755 (5)	1.0768 (1)
	Spacing	1.8601 (5)	1.8992 (6)	1.5246 (3)	2.5535 (7)	1.4429 (1)	1.5062 (2)	1.5530 (4)
	$Spread_{max}$	64.8871 (3)	63.9614 (7)	64.8369 (5)	64.4443 (6)	64.8487 (4)	65.2652 (2)	67.2345 (1)

Based on the means of the performance indicator, the algorithms with the basis of Genetic Algorithm (GA) showed good performance behind the proposed MODPSO. The NSGA-II consistently showed impressive performance in three indicators behind MODPSO although did not perform well in *Spacing* and $Spread_{max}$ indicators. Meanwhile, the MOGA algorithm showed medium performance in most of the indicators for all reference setting. By the calculated mean, this algorithm was located between the third and fourth rank. However, HGA showed inconsistent performance from one reference setting level to another. For the reference setting at Level 1, HGA shows quite good

performance at the third ranking. But when the reference setting was changed to Level 3 and 5, the HGA mean ranking dropped to the fourth and fifth position respectively.

On the other hand, the ACO algorithm showed improvement from one reference setting the level to another for $\tilde{\eta}$, *ER* and *GD*. On average, the ACO was placed in the fifth rank among all algorithms. The remaining two algorithms, DPSO and MOPSO were placed in the sixth and seventh ranks according to the indicator means. Both algorithms did not perform well in $\tilde{\eta}$, *ER* and *GD* but showed quite impressive performance in *Spacing* and $Spread_{max}$ indicators.

The performance of DPSO showed that the algorithm designed with weighted objective functional approach was unsuitable for finding a non-dominated solution although it used an efficient Regeneration procedure as in MODPSO. Meanwhile, the MOPSO's performance showed that the original PSO operator to update position and velocity was not good enough for the discrete problem. On the other hand, the NSGA-II that performed efficiently in three indicators showed that the Selection strategy based on Crowding Distance of non-dominated solution worked effectively, since the MODPSO that adopted similar strategy also did well. Based on the performance of NSGA-II and DPSO algorithms, the proposed MODPSO algorithm has inherited good features from NSGA-II and DPSO because the MODPSO algorithm mainly adopted strategies from these algorithms.

The results in Table 7 also indicate that the proposed MODPSO consistently performed better than GA-based algorithms (i.e. MOGA, HGA and NSGA-II) for all the indicators except *Spacing* in all reference settings. It shows that the MODPSO was able to optimise integrated ASP and ALB problem from various difficulty levels efficiently compared with GA-based algorithms.

6.1 Statistical Tests

To test the significance of the results, statistical tests were performed. In this case, ANOVA test was carried out to test if there was any significant difference between results obtained by an algorithm compared with other algorithms. The null hypothesis stated that there was no significant difference among all algorithm means. When the null hypothesis was accepted, it means that there

was no significant improvement achieved by any algorithms. The summary of ANOVA test is presented in Table 8.

In order to accept the null hypothesis, the calculated *f*-value must be smaller than the critical *f*-value (*f**). The *f** obtained from *f*-distribution table at 0.05 confidence interval was 3.86 [45]. Based on Table 8, only the *f*-value for $Spread_{max}$ fulfilled the requirement to accept the null hypothesis. Meanwhile, the *f*-values for $\tilde{\eta}$, *ER*, *GD* and *Spacing* indicators showed larger values compared with *f**. It means that four out of five performance indicators rejected the null hypothesis which brought the meaning that there were significant differences between algorithms. In this case, it shows that there were significant improvements achieved at least by one algorithm compared to others. Meanwhile, the acceptance of null hypothesis by $Spread_{max}$ indicator shows that all algorithms were able to explore the extreme minimum and maximum values in the search space.

Table 7: Summary of ANOVA test

	$\tilde{\eta}$	*ER*	*GD*	*Spacing*	$Spread_{max}$
SSB	512147.60	14.30	121.93	35.03	183.90
SSW	301524.1	8.1442	137.805	74.207	72260.1
MSB	85358	2.38395	20.3224	5.83856	30.648
MSW	861.5	0.02327	0.3937	0.21202	206.457
*f**	3.68	3.68	3.68	3.68	3.68
f	99.08	102.45	51.62	27.54	0.15

SSB: Sum of square between groups
SSW: Sum of square within groups
MSB: Mean squares between groups
MSW: mean squares within groups

*f**: critical f-value
f: calculated f-value

However, the ANOVA test did not tell us the exact algorithms that have significant mean differences. Therefore, a posteriori test known as the Tukey's Honestly Significant Different test (Tukey's HSD test) was performed to identify if there was any significant improvement achieved by the proposed MODPSO compared to other algorithms. The Tukey's HSD test was only conducted for the performance indicators that rejected the null hypothesis ($\tilde{\eta}$, *ER*, *GD* and *Spacing*) since only these groups showed the significant difference between algorithms. The summary of the Tukey's HSD test is presented in Table 8.

Table 8: Summary of Tukey's HSD test

		Absolute Mean Different Between MODPSO and Algorithm			
Indicator (HSD*)		$\tilde{\eta}$ (11.102099)	*ER* (0.089074)	*GD* (0.366382)	*Spacing* (0.268868)
Algorithm	MOGA	102.313725	0.461927	1.077286	0.292912
	ACO	102.235294	0.486218	1.123465	0.338841
	HGA	108.588235	0.490418	1.270296	0.012637
	NSGA-II	98.3333333	0.235592	0.500439	0.886947
	MOPSO	117.960784	0.592025	1.807975	0.062702
	DPSO	111.470588	0.593996	1.648349	0.017292

Table 8 presents the absolute mean difference between MODPSO and other algorithms. The number in bracket shows the critical HSD value (HSD*) that was calculated based on the Tukey's table [45]. When the absolute mean different between MODPSO and the particular algorithm is larger than HSD*, it means that the significant improvement has been identified between these two algorithms. Based on Table 8, the significant improvement was achieved by the proposed MODPSO compared with all other algorithms for $\tilde{\eta}$, *ER* and *GD* indicators. In the meantime, the significant *Spacing* improvements were observed between MODPSO and MOGA, ACO and NSGA-II, but not with HGA, MOPSO and DPSO. This result was consistent with earlier finding in Figure 8 and Table 6 that prioritised the HGA, MOPSO and DPSO algorithms together with MODPSO for *Spacing* indicator.

The Tukey's HSD test result explained that the proposed MODPSO performed well to converge to Pareto optimal solutions since the indicators that directly linked with it ($\tilde{\eta}$, *ER* and *GD*) showed significant improvement compared with other algorithms. On the other hand, the MODPSO only showed significant improvement in some cases in terms of uniformity of the found solution. Meanwhile, no significant improvement was found for the solution spreading although small difference as presented in Figure 9 was notified.

The proposed MODPSO algorithm showed better performance because of the fine tuning feature towards the end of iterations. This feature is important in ASP and ALB, where small changes

may lead to sudden improvement in results. The discrete updating procedure in MODPSO was designed to enable fine tuning towards the end of iterations. In PSO, all particles moved towards personal and global best solutions. According to the discrete updating procedure (Subtraction operator (X_i-X_j)) in MODPSO, zero velocity was given when similar elements in X_i and X_j were found (this is the case when all particles move towards the best solution at the end of iterations). When majority of velocity elements were zero, only small changes occurred in assembly sequence as presented by Addition operator (X_i+V_i). This feature allowed fine tuning of the assembly sequences in MODPSO.

7. Conclusions

In this work, a Multi-Objective Discrete Particle Swarm Optimisation (MODPSO) algorithm was proposed to optimise an integrated ASP and ALB problems. Indifferent with the existing algorithms, MODPSO that used Pareto-based approach to deal with multi-objective problem, adopted discrete procedure instead of standard mathematical operators to update its position and velocity. A set of 51 test problems with different range of difficulties was used to test the performance of MODPSO compared with other algorithms.

The results show that the MODPSO performed better in all test problems in finding a non-dominated solution ($\tilde{\eta}$), 82% in Error ratio (*ER*), 82% in Generational Distance (*GD*), 22% in *Spacing* and 71% in *Maximum Spread*. Meanwhile, the result in Table 7 presents that the MODPSO performed better in four out of five performance indicators in all difficulty levels. This result shows the proposed MODPSO successfully overcame the underperformance of GA-based algorithms for the test problem with a larger number of tasks.

A statistical test was conducted to identify any significant improvement achieved by the proposed MODPSO. The statistical test concluded that the MODPSO showed significant improvement compared with other algorithms to converge to Pareto optimal solutions. In terms of solution uniformity, the significant improvement achieved by MODPSO was only applied to certain

comparison algorithms. Furthermore, no significant improvement was achieved for the solution spreading using the MODPSO. Therefore, it can be concluded that the proposed MODPSO has shown good performance in terms of solution quality towards Pareto optimal solutions.

Instead of specific application to optimise integrated ASP and ALB problem, the proposed MODPSO also can be used to optimise other types of discrete problems represented using precedence graph. This includes the traveling salesman problem and vehicle routing problem with precedence constraint. However, the proposed MODPSO has limited performance in terms of solution uniformity as attained by *Spacing* indicator. Besides that, the CPU time for MODPSO is also among the highest within the comparison algorithms.

In future, an extensive effort to improve the solution uniformity and spreading is proposed to improve the quality of the overall solution. This might be achieved by hybridising the MODPSO with different algorithm with better solution spread. Besides, the MODPSO algorithm could be tested with higher complexity assembly problems, such as mixed-model, as well as two-sided and parallel lines, in order to better understand the behaviour of the algorithm at varied complexity levels. Furthermore, the application of the algorithm to the industrial problem is also suggested.

References

[1] M. F. F. Rashid, W. Hutabarat, and A. Tiwari, "A review on assembly sequence planning and assembly line balancing optimisation using soft computing approaches," *Int. J. Adv. Manuf. Technol.*, vol. 59, no. 1–4, pp. 335–349, Aug. 2011.

[2] N. Hamta, M. A. Shirazi, and S. F. Ghomi, "A bi-level programming model for supply chain network optimization with assembly line balancing and push-pull strategy," *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.*, vol. 230, no. 6, pp. 1127–1143, Feb. 2016.

[3] M. Li, Y. Zhang, B. Zeng, H. Zhou, and J. Liu, "The modified firefly algorithm considering fireflies' visual range and its application in assembly sequences planning," *Int. J. Adv. Manuf. Technol.*, vol. 82, no. 5–8, pp. 1381–1403, Jul. 2015.

[4] R. M. Marian, "Optimisation of Assembly Sequences using Genetic Algorithm," University of

South Australia, 2003.

[5] H.-E. Tseng and C.-E. Tang, “A sequential consideration for assembly sequence planning and assembly line balancing using the connector concept,” *Int. J. Prod. Res.*, vol. 44, no. 1, pp. 97–116, Jan. 2006.

[6] S. S. Ghandi and E. Masehian, “A breakout local search (BLS) method for solving the assembly sequence planning problem,” *Eng. Appl. Artif. Intell.*, vol. 39, pp. 245–266, Mar. 2015.

[7] C. Becker and A. Scholl, “A survey on problems and methods in generalized assembly line balancing,” *Eur. J. Oper. Res.*, vol. 168, no. 3, pp. 694–715, Feb. 2006.

[8] B. Sungur and Y. Yavuz, “Assembly line balancing with hierarchical worker assignment,” *J. Manuf. Syst.*, vol. 37, pp. 290–298, Oct. 2015.

[9] M. Haddadzade, M. R. Razfar, and M. H. F. Zarandi, “Multipart setup planning through integration of process planning and scheduling,” *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.*, vol. 230, no. 6, pp. 1097–1113, 2016.

[10] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*. Sussex, United Kingdom: John Wiley & Sons, 2002.

[11] S. Luke, *Essentials of Metaheuristics*, 1st Editio. California, USA: Lulu, 2010.

[12] Z. Yang, C. Lu, and H. W. Zhao, “An Ant Colony Algorithm for Integrating Assembly Sequence Planning and Assembly Line Balancing,” *Appl. Mech. Mater.*, vol. 397–400, pp. 2570–2573, Sep. 2013.

[13] C. Lu and Z. Yang, “Integrated assembly sequence planning and assembly line balancing with ant colony optimization approach,” *Int. J. Adv. Manuf. Technol.*, vol. 83, no. 1, pp. 243–256, 2016.

[14] R. Chen, K. Lu, and S. Yu, “A hybrid genetic algorithm approach on multi-objective of assembly planning problem,” *Eng. Appl. Artif. Intell.*, vol. 15, no. 2002, pp. 447–457, 2003.

[15] H.-E. Tseng, M.-H. Chen, C.-C. Chang, and W.-P. Wang, “Hybrid evolutionary multi-objective algorithms for integrating assembly sequence planning and assembly line balancing,” *Int. J. Prod. Res.*, vol. 46, no. 21, pp. 5951–5977, Nov. 2008.

[16] H. S. Wang, Z. H. Che, and C. J. Chiang, “A hybrid genetic algorithm for multi-objective product plan selection problem with ASP and ALB,” *Expert Syst. Appl.*, vol. 39, no. 5, pp. 5440–5450, Apr. 2012.

[17] M. F. F. Ab Rashid, W. Hutabarat, and A. Tiwari, “Development of a tuneable test problem generator for assembly sequence planning and assembly line balancing,” *Proc. Inst. Mech.*

Eng. Part B J. Eng. Manuf., vol. 226, no. 11, pp. 1900–1913, Aug. 2012.

[18] Z. Xinchao, "A perturbed particle swarm algorithm for numerical optimization," *Appl. Soft Comput.*, vol. 10, no. 1, pp. 119–124, Jan. 2010.

[19] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proc. ICNN'95 - Int. Conf. Neural Networks*, vol. 4, pp. 1942–1948, 1995.

[20] B. Shuang, J. Chen, and Z. Li, "Microrobot based micro-assembly sequence planning with hybrid ant colony algorithm," *Int. J. Adv. Manuf. Technol.*, vol. 38, no. 11–12, pp. 1227–1235, Sep. 2007.

[21] J. Gao, L. Sun, L. Wang, and M. Gen, "An efficient approach for type II robotic assembly line balancing problems," *Comput. Ind. Eng.*, vol. 56, no. 3, pp. 1065–1080, Apr. 2009.

[22] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms," *Adv. Eng. Informatics*, vol. 19, no. 1, pp. 43–53, Jan. 2005.

[23] J. Mukund Nilakantan and S. G. Ponnambalam, "Robotic U-shaped assembly line balancing using particle swarm optimization," *Eng. Optim.*, vol. 48, no. 2, pp. 231–252, 2015.

[24] C. Y. Liu and H. J. Wen, "Application of Multi-Objective Culture Particle Swarm Optimization in Complex Product Assembly Line Balancing," *Adv. Mater. Res.*, vol. 694–697, pp. 3526–3530, May 2013.

[25] M. Li, B. Wu, Y. Hu, C. Jin, and T. Shi, "A hybrid assembly sequence planning approach based on discrete particle swarm optimization and evolutionary direction operation," *Int. J. Adv. Manuf. Technol.*, vol. 68, no. 1–4, pp. 617–630, Feb. 2013.

[26] N. Hamta, S. M. T. Fatemi Ghomi, F. Jolai, and M. Akbarpour Shirazi, "A hybrid PSO algorithm for a multi-objective assembly line balancing problem with flexible operation times, sequence-dependent setup times and learning effect," *Int. J. Prod. Econ.*, vol. 141, no. 1, pp. 99–111, Jan. 2013.

[27] Y. Xing and Y. Wang, "Assembly sequence planning based on a hybrid particle swarm optimisation and genetic algorithm," *Int. J. Prod. Res.*, Nov. 2012.

[28] A. C. Nearchou, "Maximizing production rate and workload smoothing in assembly lines using particle swarm optimization," *Int. J. Prod. Econ.*, vol. 129, no. 2, pp. 242–250, Feb. 2011.

[29] C. A. Coello Coello and M. S. Lechuga, "MOPSO: a proposal for multiple objective particle swarm optimization," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, 2002, vol. 2, pp. 1051–1056.

[30] Y.-J. Tseng, J.-Y. Chen, and F.-Y. Huang, "A particle swarm optimisation algorithm for multi-

plant assembly sequence planning with integrated assembly sequence planning and plant assignment," *Int. J. Prod. Res.*, vol. 48, no. 10, pp. 2765–2791, May 2010.

[31] K. Rameshkumar, R. K. Suresh, and K. M. Mohanasundaram, "Discrete Particle Swarm Optimization (DPSO) Algorithm for Permutation Flowshop Scheduling to Minimize Makespan," *Adv. Nat. Comput.*, vol. 3612, pp. 572–581, 2005.

[32] D. Jianping, S. Chun, and L. Jun, "A Discrete Particle Swarm Optimization Algorithm for Assembly Line Balancing Problem of Type 1," *2011 Third Int. Conf. Meas. Technol. Mechatronics Autom.*, pp. 44–47, Jan. 2011.

[33] H. Lv and C. Lu, "An assembly sequence planning approach with a discrete particle swarm optimization algorithm," *Int. J. Adv. Manuf. Technol.*, vol. 50, no. 5–8, pp. 761–770, Jan. 2010.

[34] Y. Wang and J. H. Liu, "Chaotic particle swarm optimization for assembly sequence planning," *Robot. Comput. Integr. Manuf.*, vol. 26, no. 2, pp. 212–222, Apr. 2010.

[35] K. Deb, A. Member, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm :," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[36] M. F. F. Rashid, A. Tiwari, and W. Hutabarat, "An Integrated Representation Scheme for Assembly Sequence Planning and Assembly Line Balancing," in *Proceedings of the 9th International Conference on Manufacturing Research ICMR 2011*, 2011, no. September, pp. 125–131.

[37] S. Qu and Z. Jiang, "A memetic algorithm approach for batch-model assembly line balancing problem of sub-block in shipbuilding," *Proc. Inst. Mech. Eng. Part B J. Eng. Manuf.*, vol. 228, no. 10, pp. 1290–1304, Jan. 2014.

[38] D. E. Whitney, *Mechanical Assemblies: Their Design, Manufacture, and Role in Product Development, Volume 1*. Oxford University Press, 2004.

[39] H. Yu, J. P. Yu, and W. L. Zhang, "An Particle Swarm Optimization Approach for Assembly Sequence Planning," *Appl. Mech. Mater.*, vol. 16–19, pp. 1228–1232, Oct. 2009.

[40] H. G. Lv, C. Lu, and J. Zha, "A hybrid DPSO-SA approach to assembly sequence planning," *2010 IEEE Int. Conf. Mechatronics Autom.*, pp. 1998–2003, Aug. 2010.

[41] C. Moon, J. Kim, G. Choi, and Y. Seo, "An efficient genetic algorithm for the traveling salesman problem with precedence constraints," *Eur. J. Oper. Res.*, vol. 140, no. 3, pp. 606–617, Aug. 2002.

[42] J. Bautista and J. Pereira, "Ant algorithms for a time and space constrained assembly line

balancing problem," *Eur. J. Oper. Res.*, vol. 177, no. 3, pp. 2016–2032, Mar. 2007.

[43] Y.-K. Choi, D. M. Lee, and Y. Bin Cho, "An approach to multi-criteria assembly sequence planning using genetic algorithms," *Int. J. Adv. Manuf. Technol.*, vol. 42, no. 1–2, pp. 180–188, Jul. 2008.

[44] A. Yoosefelahi, M. Aminnayeri, H. Mosadegh, and H. D. Ardakani, "Type II robotic assembly line balancing problem: An evolution strategies algorithm for a multi-objective model," *J. Manuf. Syst.*, vol. 31, no. 2, pp. 139–151, Apr. 2012.

[45] F. L. Coolidge, *Statistics: A Gentle Introduction*. London: SAGE Publication, 2000.

Cranfield University

CERES Research Repository **https://dspace.lib.cranfield.ac.uk/**

School of Aerospace, Transport and Manufacturing (SATM) Staff publications (SATM)

Multi-objective discrete particle swarm optimisation algorithm for integrated assembly sequence planning and assembly line balancing

Ab Rashid, Mohd Fadzil Faisae

2016-10-24

Attribution-NonCommercial 3.0 International

Ab Rashid MFF, Hutabarat W, Tiwari A. (2018) Multi-objective discrete particle swarm optimisation algorithm for integrated assembly sequence planning and assembly line balancing. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture, Volume 232, Issue 8, June 2018 , pp. 1444-1459

http://dx.doi.org/10.1177/0954405416673095

Downloaded from CERES Research Repository, Cranfield University