© Universiti Tun Hussein Onn Malaysia Publisher's Office

IJIE

Journal homepage: http://penerbit.uthm.edu.my/ojs/index.php/ijie
ISSN : 2229-838X e-ISSN : 2600-7916

The International Journal of Integrated Engineering

Modelling and Optimization of Energy Efficient Assembly Line Balancing Using Modified Moth Flame Optimizer

Mohd Fadzil Faisae Ab. Rashid[1*], Nik Mohd Zuki Nik Mohamed[1], Ahmed Nurye Oumer[2]

[1]Department of Industrial Engineering, College of Engineering,
Universiti Malaysia Pahang, 26300, Gambang, Pahang, MALAYSIA

[2]Department of Mechanical Engineering, College of Engineering,
Universiti Malaysia Pahang, 26300, Gambang, Pahang, MALAYSIA

*Corresponding Author

DOI: https://doi.org/10.30880/ijie.2022.14.01.003
Received 21 September 2020; Accepted 15 February 2021; Available online 07 March 2022

Abstract: Energy utilization is a global issue due to the reduction of fossil resources and also negative environmental effect. The assembly process in the manufacturing sector needs to move to a new dimension by taking into account energy utilization when designing the assembly line. Recently, researchers studied assembly line balancing (ALB) by considering energy utilization. However, the current works were limited to robotic assembly line problem. This work has proposed a model of energy efficient ALB (EE-ALB) and optimize the problem using a new modified moth flame optimizer (MMFO). The MMFO introduces the best flame concept to guide the global search direction. The proposed MMFO is tested by using 34 cases from benchmark problems. The numerical experiment results showed that the proposed MMFO, in general, is able to optimize the EE-ALB problem better compared to five comparison algorithms within reasonable computational time. Statistical test indicated that the MMFO has a significant performance in 75% of the cases. The proposed model can be a guideline for manufacturer to set up a green assembly line in future.

Keywords: Manufacturing systems, line balancing, energy utilization, moth flame optimization

1. Introduction

The world demand for energy is increasing at a fast rate with the increasing trend of modernization and industrialization. Since the last half a century, the world's energy demand has increased by two folds. Moreover, annual rate of global primary energy consumption between 2009 and 2030 is expected to rise by 1.6%. Even though the development of renewable energy sources has been increasing, more that 80% of the total energy still comes from non-renewable fossil fuels like oil, coal and natural gas, which are the major contributors to greenhouse gas (GHG) emissions. For instance, in 2012, the energy use of the CO_2 emitting energy sources increased by additional 1.4% from the year before [1].

The manufacturing sector is one of the high energy demanding sectors. According to the report by the United Nation Environment Program (UNEP), the manufacturing sector consumes about 35% of the global electricity. Moreover, it is responsible for over 20% of the global carbon dioxide emissions, which is quite significant. These scenario forced many scientists who are working in the manufacturing field to pay attention to the energy and resource-efficiency challenges, and focus on making manufacturing processes more sustainable [2].

**Corresponding author: ffaisae@ump.edu.my*

One of the stages that should be considered in sustainable manufacturing is the design stage. It includes design for production and design for green use. Design for green use involves minimizing energy consumption, reducing waste, and end of life design. Besides the product design, the assembly layout design is also important to be considered to achieve sustainable manufacturing. In terms of assembly process, an assembly layout design that considers the energy factor could contribute to efficient energy utilization. This can be realized by optimizing the number of machines and tools that utilize energy to operate. However, minimizing the number of machines and tools in the assembly process could lead to inaccurate solution due to the differences in power consumption for different machines and tools.

Assembly line balancing (ALB) is an activity to optimize the assembly layout by assigning a balance assembly workload among workstations. This problem is a combinatorial optimization problem, which requires extensive computational effort [3]. Traditionally, the most important factor that is considered during the assembly task assignment is to ensure that every workstation has almost similar workload by fulfilling the assembly precedence constraints [4]. The ALB problem was categorized into a simple assembly line balancing (SALB) and a general assembly line balancing (GALB) [5].

Researchers implemented different optimization objectives as an indicator to measure the fitness of the assembly line. Based on the survey that was conducted by the researcher, for SALB problem, the main optimization objective is to minimize the number of workstation. This is followed by minimizing cycle time and minimizing the smoothness index of the workload [6]. These objective functions were matched with the SALB categories which focus on the mentioned objectives. Besides the objective functions, researchers also studied the different constraints in the assembly process. Sungur and Yavuz for instance, considered the worker's assignment on top of the main ALB optimization objective [7]. In addition to the main objective function, some researchers have considered to minimize the number of resources such as machine and tool in an ALB [8], [9]. The additional constraints in the ALB made the problem modelling become nearer to real situation.

On the other hand, researchers also studied the energy factor in the ALB to produce an assembly line with efficient energy utilization. However, the number of works on ALB that have considered energy utilization is relatively small compared to the total ALB published works. Nilakantan considered energy utilization in a robotic assembly line [10]. In this work, the assembly tasks need to be assigned to different robots. In this problem, the different robots will consume different energy to perform a similar task. The same energy model was also used in different published papers [10], [11]. This model is useful when the assembly line uses different robot models in the assembly process.

Michalos in 2015 presented the ALB with efficient energy for an automotive assembly problem [12]. Energy utilization was minimized in two stages; during the design and the assembly configuration stages. In the design stage, the energy utilization was minimized by reducing the number of robots. Then in the configuration stage, the best assembly task assignment was optimized to make sure that the best configuration was achieved. Besides that, researcher also considered energy utilization for robotic assembly line on a two-sided ALB [13]. In this paper, they considered the power consumption in both the operation and standby modes. The power in standby mode is assumed to be 10% of the power in operation mode. Meanwhile, Urban and Chiang studied energy efficiency of an unpaced assembly line with stochastic assembly time [14]. However, in this paper, they assumed that energy utilization can be reduced indirectly when the cycle time is reduced. Therefore, they did not present any mathematical model for energy utilization in their work.

Various optimization algorithms were proposed to optimize the ALB problem. According to a review paper, among the popular algorithms used in ALB were genetic algorithm (GA), simulated annealing (SA), ant colony optimization (ACO) and particle swarm optimization (PSO) [15]. The GA has been implemented in various versions of ALB problem. The application of GA to optimize ALB can be traced since 1992 [16]. Since that, researchers had made various progresses to the GA for ALB. [17] for example; hybridized the heuristics approach in GA by embedding heuristic result in the initialization stage. This hybrid approach had also been used by different researchers for the mixed-model ALB [18]. Besides that, researcher also hybridized the GA with a Tabu search to enhance performance [19]. Meanwhile, Gao combined the GA with a local search to optimize the robotic ALB [20]. Later, the researcher adopted a One-Fifth Success Rule (OFSR) and the probability of selecting the best solution in GA to control the diversity of generated solution for ALB [21].

Besides, the well-established algorithms that were presented above, researchers also implemented several relatively new algorithms such as the artificial bee colony [22], [23], teaching-learning based algorithm [24], [25], Benders decomposition [26] and cuckoo search algorithm [27].

Based on the presented literatures on ALB works, research on ALB can be divided into two major groups. The first group is the research which concentrated on the ALB problem modelling, while the second group focused on the optimization algorithm. The research that focused on problem modelling has been extensively studied including ALB problems which considered workers and also resources. However, there is a lack of study which considers energy utilization in ALB. The existing work that studies energy utilization in assembly line is limited to robotic assembly line. Meanwhile, in terms of optimization algorithm, recent researchers tried to explore different optimization techniques. On the other hand, the interest on well-established algorithms is still there, but the direction is towards enhancing the algorithm performance.

This paper, therefore, aims to model and optimize energy efficient ALB problem. In contrast to existing works that only focus on the robotic assembly line; the proposed model can be implemented for an assembly line that utilizes the electrical power to operate. This paper only focus on the simple assembly line problem type E (SALB-E). For optimization purpose, this work proposed a modification to a new algorithm called moth flame optimizer (MFO). MFO was introduced by Mirjalili in 2015 [28]. This algorithm is selected to explore the potential of the new algorithm, since no prior work has implemented MFO for ALB.

The rest of this paper is organized as follows. Section 2 details the problem modelling for energy efficient ALB, including a numerical example. Section 3 explains the proposed modified MFO algorithm. Section 4 discusses the computational experiment methods and results from benchmark and case study problems. Finally, section 5 presents the conclusion and future research direction.

2. Energy Efficient ALB Modelling

The energy efficient assembly line balancing (EE-ALB) is modelled according to the simple assembly line balancing type E model (SALB-E). The purpose of the problem is to design an assembly line with a balanced workload and, at the same time, minimizes energy utilization in idle mode. In this work, energy utilization is only considered in idle mode because energy consumption in the operation mode is the same for all configurations. This is because the total work content and the resources used to conduct a specific assembly task are the same.

The ALB problem is presented in a precedence graph as shown in Figure 1. The number inside the node represents the assembly task, while the edge represents the precedence constraint. For example in Figure 1, assembly task 2 cannot be started until task 1 is completed.

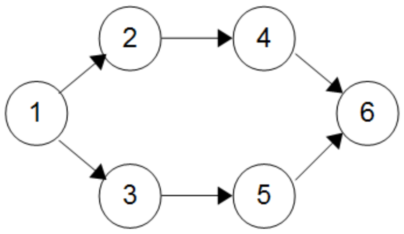

Fig. 1 - Example of precedence graph

For problem modelling purpose, the following assumptions are applied:

1. Assembly operation is conducted manually by worker.
2. One assembly workstation is operated by one worker only. Any worker can be allocated at any workstation and can perform any assigned tasks.
3. The assembly line only assembles one homogenous type of product.
4. The power consumption in the idle mode is 10% of the power consumption during the operation mode [13], [29].
5. Since the power during operation time will be the same, only the power in the idle mode is considered.
6. No similar machine is allowed at one workstation.

For the optimization of EE-ALB problem, two optimization objectives are considered. The first objective is to minimize the smoothness index (*SI*), while the second is to minimize idle energy utilization (*IEU*). In equation (1) and (2), *ct* is the cycle time, while pt_k is the processing time in the k^{th} station.

$$SI = \sqrt{\sum_{k=1}^{K} (ct - pt_k)^2} \tag{1}$$

$$ct = \max_{k=1:K} [pt_k] \tag{2}$$

$$\sum_{k=1}^{K} x_{i,k} = 1 \qquad i = 1, \dots, n$$
$$x_{i,k} = \begin{cases} 1 \text{ if the task } i \text{ is assigned in workstation } k \\ 0 \text{ otherwise} \end{cases} \tag{3}$$

$$\sum_{k=1}^{K} x_{a,k} - \sum_{k=1}^{K} x_{b,k} \leq 0 \qquad a \in n, b \in C_a \tag{4}$$

$$\sum_{i=1}^{n} t_i x_{i,k} \leq ct_{max} \qquad \forall k \tag{5}$$

The first constraint in equation (3) ensures that an assembly task is assigned to one workstation. Equation (4) represents the precedence constraint that must be followed. The $x_{a,k}$ and $x_{b,k}$ will equivalent to 1 if the task *a* or *b* is assigned into station *k*, or otherwise $x_{a,k}$ and $x_{b,k}$ is 0. C_a refers to the set of successors for task *i*. In other words, this constraint ensures that the successor/s for task *i* will be assigned to the similar or the following workstation. The constraint in equation (5) ensures that the maximum cycle time (ct_{max}) is obeyed. In equation (5), t_i refers to the assembly time for *i*th task.

The idle energy utilization (*IEU*) in the assembly line for a period of one hour is calculated as follows:

$$IEU = \frac{3600}{ct} \cdot \sum_{k=1}^{K} \sum_{h=1}^{H} \left(ct - \left(\sum_{i=1}^{n} (t_i . x_{i,k}) g_{i,h} \right) g_{k,h} . 0.1 p_h \right) \tag{6}$$

In equation (6), *H* is the total number of equipment and *K* is the total workstation. Meanwhile p_h is the power rating for the *h*th equipment. $g_{i,h} = 1$ if the equipment *h* is needed for task *i*, otherwise $g_{i,h} = 0$. Meanwhile $g_{k,h}$ equal 1 when equipment *k* is needed in workstation *k*.

This problem involves two optimization objectives as mentioned in equation (1) and (6). To deal with the multi-objective optimization problem, the weighted sum approach is used. However, since the ranges of these objective functions are different, it needs to be normalized into similar range [0, 1].

$$\widehat{SI} = \frac{SI - SI_{min}}{SI_{max} - SI_{min}} \tag{7}$$

$$\widehat{IEU} = \frac{IEU - IEU_{min}}{IEU_{max} - IEU_{min}} \tag{8}$$

In equation (7) and (8), $\widehat{SI}$ and $\widehat{IEU}$ refers to normalized *SI* and *IEU* respectively. Therefore, using the weighted sum approach, the objective function is formulated as follows:

$$Min\ f = w_1 \widehat{SI} + w_2 \widehat{IEU} \tag{9}$$

In equation (9), w_1 and w_2 are the weight for the optimization objective, where the w_1 and w_2 are set at 0.5 throughout the computational experiment and case study problem.

3. Moth Flame Optimizer

The moth flame optimizer (MFO) is inspired by the flying and navigation mechanisms of the moth [28]. At night, this insect flies and navigates as guided by the moon light. They fly at a fixed angle relative to the moon. This ensures the effectiveness to fly in a straight line in a long distance, since the moon is far away from the earth. This mechanism is known as traverse orientation. However, when there is artificial light, the moths are trapped in the spiral flying movement because they try to maintain the fix angle to the light source as shown in Figure 2. Furthermore, the distance of the light source to the moth is very small compared to the moon distance.

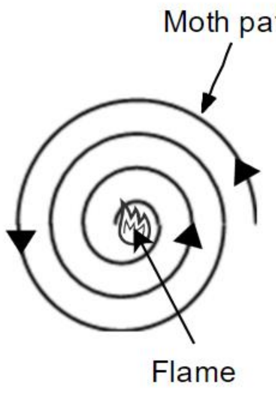

Fig. 2 - Moth flying in spiral direction towards flame

Mathematically, the MFO algorithm consists of three functions (*I*, *P*, *T*). The *I* element represents the function that generates the random initial population. Meanwhile, the *P* element is the function to move the moths from an initial position to the final position in the search dimension. The input of the *P* function is the existing moth position, while the output is the updated moth position. Finally, the *T* element characterizes the termination function which determines the continuity of the iteration process in MFO. Even though there are a few approaches that can be used, the maximum iteration number to terminate the optimization was still in used.

3.1 Initialization

The initialization step involves setting up the initial solutions. The most important element is the moth position matrix (*M*) that represents the solution. The *M* matrix consists of *n* moth with *d* dimensions.

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} & \cdots & m_{1,d} \\ m_{2,1} & m_{2,2} & \cdots & m_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \dots & m_{n,d} \end{bmatrix}$$

To create this matrix, random distribution is used to generate the initial population based on the following equation:

$$M_{i,j} = (ub_i - lb_i) * rand[0,1] + lb_i \tag{10}$$

The ub_i and lb_i indicated the upper and lower bound for the i^{th} variable. For each of the moth, the fitness of the solution is presented in *OM* matrix.

$$OM = \begin{bmatrix} OM_1 \\ OM_2 \\ \vdots \\ OM_n \end{bmatrix}$$

Besides the moth matrix, there is a secondary matrix that stores the flame information. The moth and flame matrices are of the same size, but they might store different information. The flame matrix (*F*) keeps the best solution found by a particular moth in their journey. Apart from *F*, there is also the fitness matrix for the flame, *OF*. The *F* and *OF* matrices are presented as follows:

$$F = \begin{bmatrix} F_{1,1} & F_{1,2} & \cdots & F_{1,d} \\ F_{2,1} & F_{2,2} & \cdots & F_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ F_{n,1} & F_{n,2} & \dots & F_{n,d} \end{bmatrix}$$

$$OF = \begin{bmatrix} OF_1 \\ OF_2 \\ \vdots \\ OF_n \end{bmatrix}$$

3.2 Evaluation

In the continuous problem, the evaluation process can be simply made by replacing the moth position, $M_{i,j}$ into the objective function. However, for a discrete combinatorial optimization problem with precedence constraint, $M_{i,j}$ need to be decoded into a feasible solution. In this case, the topological sort procedure is implemented. The $M_{i,j}$ is defined as a priority factor for jth task in ith solution to determine the selected assembly task to form a feasible assembly sequence, seq_i. To illustrate the topological sort procedure, an example of the problem given in Figure 1 is considered. In this example, the assembly process consists of six tasks. Therefore, the moth position consists of six dimensions. Lets assume the first solution is M_1 = [4.263 8.427 1.869 7.336 2.293 1.483].

To decode the M_1, the algorithm needs to identify the available candidate task. The candidate task consists of the assembly task without the precedence. Referring to Figure 1, the only candidate task is task 1. Therefore, task 1 is stored into seq_1 = [1]. Next the selected task is removed from the precedence graph. Now, the candidate task consists of tasks 2 and 3. In this situation, M_1 for the 2nd and 3rd dimensions are compared. Since $M_{1,2}$ is larger than $M_{1,3}$, the assembly task 2 is selected to be stored in seq_1. Now seq_1 = [1 2]. This approach is repeated until all six assembly tasks are selected into seq_1. For the given M_1 above, the feasible assembly sequence that was decoded using the topological procedure is seq_1 = [1 2 4 3 5 6]. Once the feasible assembly sequence is decoded, the evaluation process is conducted by using the fitness function in equation (9).

3.3 Updating Moths and Flames

In the P function, the moth position is updated by referring to the flame position function as follows:

$$M_i = S(M_i, F_j) \tag{11}$$

For M_i is the ith moth, S is the function of spiral and F_j is the jth flame. The spiral function S is a logarithm that uses the following equation:

$$S(M_i, F_j) = D_i \cdot e^{\gamma q} \cdot \cos(2\pi q) + F_j \tag{12}$$

D_i calculates the distance between the ith moth and jth flame. Meanwhile, constant γ represents the spiral shape [0, 1] and q is a random number between [-1, 1]. D_i taking absolute value of subtraction between flame position, F_j and the moth position M_i as follows:

$$D_i = |F_j - M_i| \tag{13}$$

The q parameter defines the closeness of the moth to the flame. For this purpose, q is calculated as follows:

$$q = (\varphi - 1) * rand[0,1] + 1 \tag{14}$$

In equation (14), φ is a coefficient that linearly decreases over the iteration from -1 to -2. By having the φ coefficient, the spiral shape moth path towards the flame can be achieved.

Up to this point, we have only discussed the updating procedure using the equation (12), which obliges the moth to fly towards a flame. This can cause the moth to trap in local optima, since it only follows a single flame source. To improve the exploration in MFO, the flame is updated by sorting their fitness from the best to the worst. By using this approach, the moth will follow different flames based on their fitness level. This mechanism will diverse the search direction in MFO.

Based on the presented MFO procedure above, the moths update its' position by relying highly on the flame position, F_j. This procedure will make the chances to be trapped in local optima to be higher. The only mechanism to avoid this problem is by sorting the flame from best to worst, while maintaining the moth in original orders. This will divert the moth flying direction to reduce moth trapping in local optima.

However, the flame sorting mechanism makes the search direction become too diversified, since the moth needs to follow a specific flame. In the case of the problem with y number of flames, there are also y different search directions in the algorithm. In this paper, we have proposed to improve the updating procedure by taking into account the best flame for the current iteration, $F_{best,}$ in the updating formula. For this purpose, the absolute distance from the F_{best} to the moths is calculated. In the meantime, if the absolute distance is directly included in the updating formula, the premature convergence can occur. Therefore, the updating procedure from Sine-Cosine Algorithm (SCA) is adopted [30]. In SCA, the following formulas are used to update the position.

$$X_i' = \begin{cases} X_i + r_1 \sin(r_2) \times |r_3 F_{best} - X_i|, & r_4 < 0.5 \\ X_i + r_1 \cos(r_2) \times |r_3 F_{best} - X_i|, & r_4 \geq 0.5 \end{cases} \tag{15}$$

X_i represents the existing position, while X_i' is the updated position. Meanwhile r_1, r_2, r_3 and r_4 are the random numbers [0, 1]. To integrate this procedure into MFO, only the second part of the formula is considered as follows:

$$V_i = \begin{cases} r_1 \sin(r_2) \times |r_3 F_{best} - M_i|, & r_4 < 0.5 \\ r_1 \cos(r_2) \times |r_3 F_{best} - M_i|, & r_4 \geq 0.5 \end{cases} \tag{16}$$

Therefore, the updating procedure in equation (12) is replaced with a modified equation as follows:

$$S(M_i, F_j) = D_i \cdot e^{\gamma q} \cdot \cos(2\pi q) + V_i + F_j \tag{17}$$

The modified updating formula makes all the flames move towards the F_{best}. However, the moving direction is not straight forward because of the sine and cosine functions. This gives the algorithm a better guided exploration ability. Furthermore, the F_{best} from iteration to iteration might be different. This makes the search direction to be diversified, but in a guided mode.

4. Computational Experiment

This section explains the computational experiment that is used to measure the performance of the modified MFO. For this purpose, a set of well-known benchmark ALB problems taken from http://assembly-line-balancing.mansci.de/ are used [31]. Since these benchmark problems did not consider energy utilization, the equipment and power consumptions for each of the assembly tasks in the benchmark problems were randomly generated.

For the purpose of algorithm comparison, the performance of the modified MFO is compared with the original MFO algorithm and Sine-Cosine Algorithm (SCA). In this case, SCA is used since the modified MFO adopted the updating procedure from SCA. Besides that, the modified MFO was also compared with other population-based algorithms such as Genetic Algorithm (GA), Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) algorithms. These algorithms (GA, ACO and PSO) were selected because of their popularity in optimizing the ALB based on literature review. Furthermore, GA, ACO and PSO were among the well-established algorithms that were used in different optimization problems.

For the computational experiment, the population size for all algorithms was set to 30, while the maximum iteration was 500. For each of the problem, the optimization is repeated 10 times with different pseudo-random seeds. In the end, the minimum, mean and standard deviation of the fitness were measured. The benchmark test problems consist of six problems. These problems are categorized as small size (Mitchell and Sawyer), medium size (Kilbridge and Tonge) and large size (Lutz2 and Arc). Each of the benchmark problems are tested with different maximum cycle time, ct_{max} as suggested by the original benchmark problem. In total, there are 34 cases that have been considered. Next, the proposed MMFO is implemented to optimize the case study problem.

Table 1 presents the optimization results for EE-ALB obtained from ten runs. For each of the cases, the results are presented in terms of minimum fitness, mean fitness and standard deviation (SD). In Table 1, the bolded value means the best obtained minimum and mean fitness for a particular case.

Table 1 - Minimum fitness obtained by different algorithms

Problem	Cycle time	Fitness	GA	ACO	PSO	SCA	MFO	MMFO
Mitchell (21 tasks)	14	Minimum	0.1855	0.1824	0.1855	0.1847	0.1824	**0.1811**
		Mean	0.2008	**0.1838**	0.1967	0.2018	0.1949	0.1845
		SD	0.0089	0.0014	0.0084	0.0191	0.0148	0.0049
	15	Minimum	0.1958	0.1847	0.1980	0.1851	**0.1811**	0.1824
		Mean	0.2269	0.1901	0.2195	0.2111	0.1998	**0.1859**
		SD	0.0393	0.0073	0.0233	0.0222	0.0234	0.0045
	21	Minimum	0.1637	**0.1425**	0.1454	0.1565	0.1465	**0.1425**
		Mean	0.1895	0.1455	0.2052	0.1760	0.1843	**0.1427**
		SD	0.0283	0.0034	0.0345	0.0165	0.0286	0.0005
	26	Minimum	0.2079	0.1437	0.1917	0.1747	0.1537	**0.1425**
		Mean	0.2508	0.1552	0.2459	0.1844	0.1851	**0.1524**
		SD	0.0263	0.0116	0.0560	0.0087	0.0472	0.0066
	35	Minimum	0.1476	0.1242	0.1452	0.1242	0.1242	**0.1177**
		Mean	0.1622	**0.1242**	0.1615	0.1399	0.1390	0.1270
		SD	0.0102	0.0000	0.0161	0.0107	0.0095	0.0085
	39	Minimum	0.1341	**0.1295**	0.1613	0.1341	**0.1295**	0.1332
		Mean	0.1528	0.1344	0.1788	0.1601	0.1470	**0.1334**
		SD	0.0132	0.0061	0.0111	0.0152	0.0161	0.0004
Sawyer (30 tasks)	27	Minimum	0.2719	**0.2207**	0.2318	0.2749	0.2256	0.2235
		Mean	0.2938	0.2393	0.3127	0.2998	0.2532	**0.2367**
		SD	0.0230	0.0196	0.0497	0.0224	0.0216	0.0176
	30	Minimum	0.2787	0.2343	0.2809	0.2743	0.2733	**0.2281**
		Mean	0.2966	0.2689	0.3051	0.2862	0.2804	**0.2595**
		SD	0.0158	0.0196	0.0257	0.0078	0.0040	0.0230
	33	Minimum	0.2940	0.2624	**0.2478**	0.2849	0.2817	0.2691
		Mean	0.3156	**0.2755**	0.2957	0.3137	0.3004	0.2775
		SD	0.0168	0.0136	0.0350	0.0194	0.0198	0.0052
	36	Minimum	0.2297	**0.2118**	0.2274	0.2178	0.2521	0.2128
		Mean	0.3128	**0.2162**	0.2900	0.2712	0.2806	0.2247
		SD	0.0573	0.0065	0.0496	0.0345	0.0243	0.0150
	41	Minimum	0.2708	**0.1714**	0.2839	0.2521	0.2503	0.2157
		Mean	0.3002	**0.2089**	0.3200	0.2867	0.2772	0.2464
		SD	0.0458	0.0301	0.0437	0.0269	0.0306	0.0255
	47	Minimum	0.2414	0.2024	0.2344	0.2564	**0.1951**	0.1961
		Mean	0.3051	0.2433	0.2846	0.2676	0.2404	**0.2346**
		SD	0.0561	0.0233	0.0347	0.0133	0.0275	0.0324
	54	Minimum	0.2440	0.2136	0.2155	0.2475	0.2173	**0.2113**
		Mean	0.2720	0.2245	0.2613	0.2690	0.2489	**0.2139**
		SD	0.0374	0.0121	0.0340	0.0235	0.0214	0.0025
	75	Minimum	0.2460	0.2369	0.2444	0.2429	0.2013	**0.1953**
		Mean	0.2504	0.2399	0.2630	0.2526	0.2384	**0.2233**
		SD	0.0043	0.0041	0.0281	0.0151	0.0211	0.0170
Kilbridge (45 tasks)	79	Minimum	0.2968	0.2344	0.2810	0.2985	0.2631	**0.2308**
		Mean	0.3303	**0.2475**	0.3056	0.3216	0.2849	0.2511
		SD	0.0237	0.0141	0.0237	0.0133	0.0235	0.0148
	92	Minimum	0.2853	**0.1977**	0.3070	0.2854	0.2825	0.2042
		Mean	0.3367	**0.2316**	0.3269	0.3165	0.3145	0.2456
		SD	0.0342	0.0265	0.0119	0.0182	0.0232	0.0233
	110	Minimum	0.2938	**0.2253**	0.2383	0.2970	0.2831	0.2367
		Mean	0.3266	0.2607	0.3110	0.3202	0.3171	**0.2558**
		SD	0.0241	0.0236	0.0440	0.0201	0.0295	0.0161
	111	Minimum	0.2896	0.2233	0.2667	0.3023	0.2276	**0.1988**
		Mean	0.3405	0.2470	0.3109	0.3476	0.2765	**0.2301**
		SD	0.0396	0.0180	0.0447	0.0379	0.0372	0.0189
	138	Minimum	0.3111	0.2082	0.3568	0.2831	0.3039	**0.2064**
		Mean	0.3629	0.2399	0.3760	0.3697	0.3640	**0.2394**
		SD	0.0422	0.0181	0.0196	0.0535	0.0466	0.0312
	184	Minimum	0.2665	0.1715	0.2370	0.2125	0.2212	**0.1708**
		Mean	0.4353	0.1930	0.3635	0.3384	0.2961	**0.1918**

Problem	**Cycle time**	**Fitness**	**GA**	**ACO**	**PSO**	**SCA**	**MFO**	**MMFO**
		SD	0.1013	0.0126	0.1536	0.1400	0.1245	0.0211
Tonge (70 tasks)	320	Minimum	0.2263	0.2386	0.2285	0.2320	**0.1862**	0.2112
		Mean	0.2781	0.2422	0.2700	0.2566	**0.2153**	0.2251
		SD	0.0341	0.0023	0.0391	0.0231	0.0269	0.0142
	364	Minimum	0.2341	**0.1709**	0.2311	0.2052	0.1924	0.1832
		Mean	0.2924	**0.2034**	0.2711	0.2262	0.2114	0.2054
		SD	0.0390	0.0197	0.0437	0.0176	0.0112	0.0221
	410	Minimum	0.2410	0.1795	0.1986	0.2021	0.1899	**0.1598**
		Mean	0.2776	0.1904	0.2150	0.2370	0.2086	**0.1755**
		SD	0.0314	0.0108	0.0125	0.0251	0.0196	0.0111
	468	Minimum	0.2272	0.1944	0.2160	0.2040	0.2014	**0.1829**
		Mean	0.2556	0.1983	0.2272	0.2247	0.2210	**0.1859**
		SD	0.0195	0.0028	0.0143	0.0246	0.0159	0.0048
	527	Minimum	0.2172	0.1683	0.1762	0.1638	**0.1461**	0.1574
		Mean	0.2299	0.1752	0.2177	0.1954	0.1794	**0.1599**
		SD	0.0145	0.0072	0.0250	0.0184	0.0214	0.0032
Lutz2 (89 tasks)	17	Minimum	0.5783	0.5844	0.6254	0.5977	0.6002	**0.5552**
		Mean	0.6138	0.6081	0.6406	0.6249	0.6283	**0.5598**
		SD	0.0242	0.0178	0.0201	0.0258	0.0167	0.0034
	18	Minimum	0.6284	0.5854	0.5962	0.6112	0.5994	**0.5704**
		Mean	0.6495	0.6056	0.6279	0.6203	0.6179	**0.5892**
		SD	0.0120	0.0126	0.0201	0.0139	0.0168	0.0158
	19	Minimum	0.6120	0.5721	0.6129	0.5707	**0.5547**	0.5651
		Mean	0.6356	0.5932	0.6357	0.6147	0.5910	**0.5752**
		SD	0.0164	0.0146	0.0158	0.0272	0.0296	0.0119
	20	Minimum	0.6185	0.5849	0.6445	0.6132	0.5923	**0.5672**
		Mean	0.6323	0.5957	0.6564	0.6416	0.6134	**0.5916**
		SD	0.0144	0.0128	0.0087	0.0225	0.0246	0.0216
Arc (111 tasks)	10027	Minimum	0.2504	0.2386	0.2406	0.2627	**0.1731**	0.1938
		Mean	0.3041	0.2621	0.2798	0.3023	0.2272	**0.2125**
		SD	0.0469	0.0221	0.0419	0.0276	0.0380	0.0220
	10743	Minimum	0.2536	0.2420	0.2232	0.2162	**0.1960**	0.2222
		Mean	0.3093	0.2555	0.2440	0.2515	**0.2110**	0.2338
		SD	0.0611	0.0122	0.0182	0.0323	0.0205	0.0106
	11378	Minimum	0.2235	0.2083	0.1810	0.2296	0.1634	**0.1614**
		Mean	0.2682	0.2245	0.2287	0.2568	0.1962	**0.1841**
		SD	0.0344	0.0102	0.0377	0.0232	0.0191	0.0141
	11570	Minimum	0.2957	0.1777	0.2010	0.2179	0.1897	**0.1730**
		Mean	0.3188	0.1887	0.2356	0.2527	0.2275	**0.1756**
		SD	0.0204	0.0104	0.0259	0.0228	0.0272	0.0024
	17067	Minimum	0.2158	0.1361	0.2750	0.1617	0.1835	**0.1328**
		Mean	0.3488	0.1459	0.2983	0.2112	0.2333	**0.1454**
		SD	0.0920	0.0190	0.0205	0.0439	0.0447	0.0132

Based on the minimum results in Table 1, there is no single algorithm that dominates a particular benchmark problem in this computational experiment. To further analyze the results, a standard competition ranking was used. For this purpose, the best fitness will be assigned rank 1, while the worst fitness is ranked as 6. When there is a tie, the following rank is left from the calculation. Table 2 below presents the frequency of the ranking obtained by the algorithms.

Table 2 - Frequency of the rank for algorithms

Algorithm		Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6
GA	Best	0	1	1	2	17	13
	Mean	0	0	2	4	6	22
ACO	Best	8	16	5	3	1	1
	Mean	8	19	6	0	1	0
PSO	Best	1	0	5	10	8	10
	Mean	0	0	3	8	13	10
SCA	Best	0	2	6	11	7	8
	Mean	0	0	4	15	13	2
MFO	Best	8	4	12	8	1	1
	Mean	2	5	19	7	1	0
MMFO	Best	19	12	3	0	0	0
	Mean	24	10	0	0	0	0

Based on Table 2, the proposed MMFO came out with the most frequent results with rank 1. The MMFO has been ranked as 1 in 19 cases (56%). From this number, the MMFO showed a better performance compared with other algorithms in 18 cases, while in one of the case, the MMFO has a tie ranked with other algorithms. This is followed by ACO and MFO with 24% of the cases. In terms of mean fitness, the proposed MMFO has obtained the best mean fitness in 70% of the cases. While in the remaining 30% of the cases, the MMFO is in second position in terms of the mean fitness. Meanwhile, ACO is in the next position with 24% of the best mean, 56% in the second rank and 18% in the third rank. On the other hand, GA is the algorithm with the most frequently ranked in the fifth and sixth place, in terms of the best and mean fitness. This is followed by PSO and SCA algorithms.

Figure 3 and 4 show the error bars of the average rank for the considered algorithms. As mentioned earlier, the proposed MMFO has a better rank compared with the comparison algorithms. However, the error bars indicated inconclusive evidence regarding the performance of MMFO compared with ACO. Therefore, a statistical test was conducted to further analyze the results.

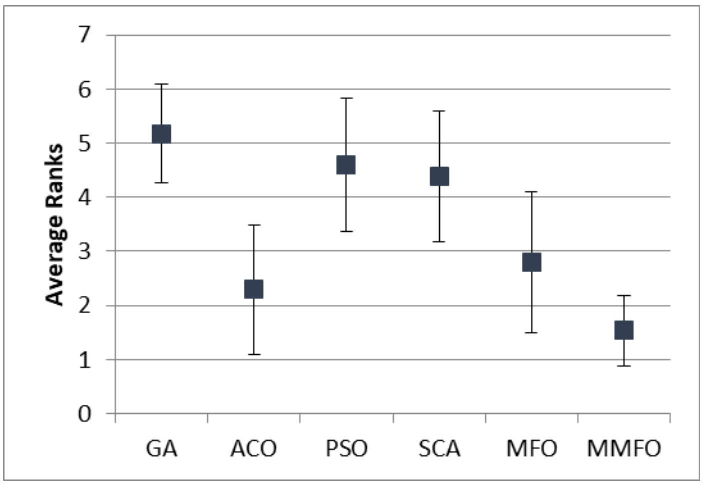

Fig. 3 - Average ranks for minimum fitness

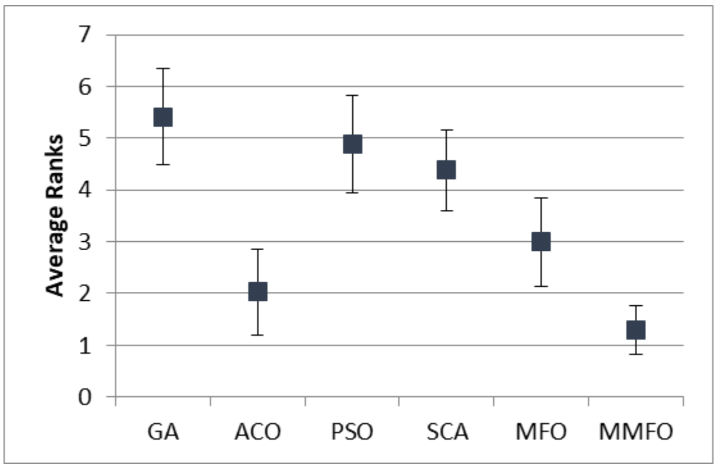

Fig. 4 - Average ranks for mean fitness

4.1 Statistical Test

In order to test for the significant difference in the algorithms, an ANOVA test is conducted. The purpose of statistical test is to measure either the obtained result using the proposed algorithm have significant difference or not. If the results show significant difference, it confirmed that the MMFO has relevant advantage statistically compared with other algorithms. Otherwise, the result obtained by MMFO did not have any differences with other algorithms.

In this case, one-way ANOVA with the confidence interval at 95% is used. The null hypothesis stated that there are no differences in the mean of fitness. Meanwhile, the alternative hypothesis specified that there are significant differences in the mean of fitness. In this test, when the *P*-value is larger than the α (α = 0.05, since confidence interval is 95%), the null hypothesis is accepted. Otherwise, if the *P*-value is less than α, the null hypothesis is rejected [32].

The results of ANOVA are summarized in Table 3. From this table, the *P*-value is less than α in 32 out of 34 cases. This result shows that there are significant differences in the mean of fitness in 94% of the problems. In other words, the results confirm that there is at least one algorithm that has a significant performance over other algorithms in majority of the problems. However, the ANOVA test did not specifically reveal the group of data (or algorithm) that has a significant performance over other group. To identify the group of data that has a significant performance, a post hoc analysis is conducted. In this case, the Fisher's least significant difference (*LSD*) is implemented [33]. The *LSD* is calculated using the following equation:

$$LSD = tc.\sqrt{MSW\left(\frac{1}{N_1}+\frac{1}{N_2}\right)} \qquad (18)$$

In the *LSD* equation, *tc* represents critical *t*-value from the *t*-distribution table, for 95% confidence interval and 54 degrees of freedom. *MSW* represents the mean square within the group, N_1 and N_2 are the numbers of sample data in the considered groups. Next, the absolute mean difference between particular groups with the comparison groups is calculated. When the absolute mean difference is larger than the *LSD*, it shows that a particular group has a significant difference compared with the comparison groups. In this study, we are interested to analyze the results from the proposed MMFO. Therefore, the absolute mean differences between MMFO and comparison algorithms are presented in Table 3. In this table, the bolded value shows that the MMFO has a significant difference over the comparison algorithm. Meanwhile, the value in the bracket means that the comparison algorithm has a significant difference compared with MMFO.

Based on the *LSD* analysis, the MMFO has a significant performance in 75% of the cases when compared with comparison algorithms. In detail, the MMFO has a significant performance compared with GA in all problems. Meanwhile, in comparison with PSO and SCA, the MMFO significantly performed better in 94% of the problems, and 71% of the problems compared with MFO. However, the MMFO only has a significant performance in 18% of the problems compared with ACO. This result shows that the proposed MMFO is able to come out with better performance in the majority of the problems, except when comparing with ACO. On the other hand, ACO also did not have a significant performance over MMFO, except in one problem (i.e. Sawyer with cycle time 41).

Table 3 - Summary of ANOVA and LSD test

Benchmark Problem	**ANOVA**			**LSD**	**Absolute Mean Difference Between MMFO and Comparison Algorithms**				
	CT	***MSW***	***P*-value**		**GA**	**ACO**	**PSO**	**SCA**	**MFO**
Mitchell_21	14	0.000126	0.063644	0.0101	**0.0164**	0.0006	**0.0122**	**0.0173**	**0.0104**
	15	0.000529	0.050966	0.0206	**0.0410**	0.0042	**0.0336**	**0.0252**	0.0139
	21	0.000515	**0.000933**	0.0203	**0.0468**	0.0028	**0.0625**	**0.0332**	**0.0415**
	26	0.001051	**7.2E-05**	0.0291	**0.0983**	0.0027	**0.0934**	**0.0320**	**0.0326**
	35	0.000107	**4.7E-06**	0.0093	**0.0352**	0.0028	**0.0345**	**0.0129**	**0.0120**
	39	0.000137	**1.82E-05**	0.0105	**0.0194**	0.0009	**0.0453**	**0.0267**	**0.0136**
Sawyer_30	27	0.000728	**0.000106**	0.0242	**0.0571**	0.0026	**0.0761**	**0.0632**	0.0165
	30	0.000317	**0.004645**	0.0160	**0.0371**	0.0093	**0.0456**	**0.0267**	**0.0208**
	33	0.000420	**0.019941**	0.0184	**0.0381**	0.0020	0.0182	**0.0362**	**0.0229**
	36	0.001289	**0.002231**	0.0322	**0.0880**	0.0085	**0.0653**	**0.0465**	**0.0559**
	41	0.001148	**0.000152**	0.0304	**0.0538**	(0.0375)	**0.0736**	**0.0402**	**0.0308**
	47	0.001146	**0.017039**	0.0304	**0.0705**	0.0087	**0.0500**	**0.0331**	0.0058
	54	0.000620	**0.003945**	0.0223	**0.0581**	0.0107	**0.0474**	**0.0552**	**0.0351**
	75	0.000298	**0.023901**	0.0155	**0.0271**	**0.0166**	**0.0397**	**0.0292**	0.0151
Kilbridge	79	0.000378	**4.56E-07**	0.0174	**0.0792**	0.0036	**0.0545**	**0.0705**	**0.0338**
	92	0.000571	**2.44E-07**	0.0214	**0.0911**	0.0139	**0.0813**	**0.0709**	**0.0689**
	110	0.000769	**0.000582**	0.0249	**0.0708**	0.0049	**0.0552**	**0.0644**	**0.0613**
	111	0.001108	**5.16E-06**	0.0298	**0.1104**	0.0168	**0.0808**	**0.1175**	**0.0464**
	138	0.001416	**7.17E-07**	0.0337	**0.1235**	0.0005	**0.1366**	**0.1303**	**0.1246**
	184	0.011592	**0.008305**	0.0965	**0.2435**	0.0012	**0.1718**	**0.1467**	**0.1043**
Tonge_70	320	0.000692	**0.005113**	0.0236	**0.0530**	0.0171	**0.0449**	**0.0315**	0.0098
	364	0.000789	**6.35E-05**	0.0252	**0.0870**	0.0020	**0.0657**	0.0209	0.0060
	410	0.000399	**4.51E-07**	0.0179	**0.1021**	0.0149	**0.0394**	**0.0615**	**0.0331**
	468	0.000246	**6.56E-06**	0.0141	**0.0697**	0.0124	**0.0412**	**0.0388**	**0.0351**
	527	0.000282	**4.31E-06**	0.0151	**0.0700**	**0.0152**	**0.0578**	**0.0355**	**0.0195**
Lutz 2_89	17	0.000377	**1.86E-05**	0.0174	**0.0540**	**0.0483**	**0.0808**	**0.0651**	**0.0685**
	18	0.000239	**8.12E-05**	0.0139	**0.0603**	**0.0164**	**0.0387**	**0.0311**	**0.0287**
	19	0.000415	**0.000209**	0.0183	**0.0605**	0.0180	**0.0606**	**0.0395**	0.0159
	20	0.000330	**2.14E-05**	0.0163	**0.0406**	0.0040	**0.0647**	**0.0500**	**0.0217**
Arc 111	10027	0.001189	**0.000192**	0.0309	**0.0916**	**0.0496**	**0.0673**	**0.0898**	0.0147
	10743	0.001055	**0.001392**	0.0291	**0.0755**	0.0217	0.0102	0.0177	0.0228
	11375	0.000634	**9.82E-05**	0.0226	**0.0841**	**0.0404**	**0.0447**	**0.0727**	0.0121
	11570	0.000478	**1.32E-06**	0.0196	**0.1432**	0.0131	**0.0600**	**0.0771**	**0.0519**
	17067	0.004225	**0.001915**	0.0583	**0.2034**	0.0005	**0.1529**	**0.0658**	**0.0879**

The result of the computational experiment and statistical test indicated that the MMFO has a better overall performance, considering the percentage of the cases in rank 1 and 2 obtained by this algorithm and also the cases that this algorithm has a significant performance. However, this does not mean that the MMFO has better performance in all of the considered problems. The nearest challenger to the MMFO is the ACO algorithm. This is followed by the original MFO. The reason for the performance of MMFO is the balance mechanism for exploitation and exploration. The exploitation refers to how good the mechanism inside the algorithm manipulates the existing solution to reproduce new solution for the following iteration. Meanwhile, the exploration refers to the mechanism to explore the search space.

In MFO, the exploitation mechanism used is the behavior of the moth flying in spiral direction towards the flame. While the exploration mechanism is the flame sorting that makes the moth follow the different flame source when it has been updated. However, the flame sorting mechanism resulted in unguided global search direction. On the other hand, this mechanism has its own benefit because the unguided global search direction may suddenly produce better solution since the exploration is semi-random. This can be observed in Table 2, where the number of solution in rank 1 with a better fitness is relatively high, but lacking in the better mean.

The proposed MMFO adopted a concept of best flame that will guide the global search direction in algorithm. This concept made the exploration mechanism more structured, where all the flames move towards the best flame in sine and cosine wave direction. This mechanism has produced a positive effect especially in terms of consistencies of algorithm to produce good solution. This effect has been proven by the mean fitness result of MMFO which was only ranked in first and second positions.

On the other hand, ACO is well-known for having good performance in optimizing discrete combinatorial problem. The solution construction mechanism in ACO is somehow highly related to combinatorial problem like ALB. In comparison with other algorithms that generate the solution for the whole dimensions simultaneously, the solution construction in ACO produces the solution dimension by dimension, guided by the level of pheromone deposited by the ant. This mechanism provides a better chance for ACO to produce good solution in ALB.

Table 4 presents the average CPU time for algorithm to complete the iterations. On average, the MMFO consumed about 54% additional CPU time compared with MFO. Comparing with other algorithms, the MMFO roughly in the fourth or fifth place in term of the CPU time. This is because of additional mechanism in the MMFO that requires the flame to move towards the best flame in the search space. In comparison with original MFO, the MMFO needs to determine the best flame, then calculate distance of best flame to moths, and update the moth position using additional formula. On the other hand, ACO algorithm required longer CPU time because the solution is constructed task by task instead of the whole assembly sequence in the other algorithms. Meanwhile, the discrete combinatorial crossover mechanism in GA makes this algorithm consumed the highest CPU time. In the GA, the partially mapped crossover (PMX) mechanism involved swapping a substring, while the remaining elements need to be inspected one by one to avoid duplication of the same offspring.

Table 4 - Average CPU time

Problem	**CPU Time (seconds)**					
	GA	**ACO**	**PSO**	**SCA**	**MFO**	**MMFO**
Mitchell 21	12.10	11.63	11.33	10.82	8.18	12.39
Sawyer 30	34.11	46.09	33.83	33.35	24.76	40.16
Kilbridge 45	99.24	76.13	64.01	31.37	45.03	69.71
Tonge 70	778.42	608.45	444.17	493.56	343.22	546.68
Lutz2 89	1895.80	936.39	854.88	858.71	636.86	1028.36
Arc 111	3733.31	1407.45	1387.43	1818.75	1406.57	1948.80

5. Conclusions

Energy utilization becomes an important issue to be considered in the manufacturing sector as one of the biggest power consumption in the world. Considering the rise in energy cost and severe environmental impact from energy generation, the assembly process in the manufacturing sector should move to reduce energy utilization. This paper aims to model and optimize an energy efficient assembly line balancing (EE-ALB). In the proposed model, the energy utilization during the idle mode for the equipment which are used to assemble the product is considered.

For the optimization purpose, a modified moth flame optimizer (MMFO) is proposed to optimize this problem. MMFO improves the exploration ability in moth flame optimizer (MFO) by introducing the best flame concept which guides the global search direction in the algorithm. A computational experiment has been conducted using 34 cases from the ALB benchmark problems. The results indicated that the MMFO has a better overall performance compared with the comparison algorithms. Even though the computational time is higher than the MFO, it is still comparable with the well-established algorithms.

The ALB with energy utilization model which was presented in this paper could be a framework to design an efficient energy assembly line. In the future, the energy efficient assembly line concept will consider the different ALB classes such as two-sided and mixed-model problems.

Acknowledgement

The author would like to acknowledge the Ministry of Education, Malaysia and Universiti Malaysia Pahang for funding this research under FRGS grant RDU1901108 (FRGS/1/2019/TK03/UMP/02/3).

References

[1] F. Birol, “Redrawing the Energy-Climate Map: World Energy Outlook Special Report,” Paris, 2013.

[2] C. Das and S. Jharkharia, “Low carbon supply chain: a state-of-the-art literature review,” *J. Manuf. Technol. Manag.*, vol. 29, no. 2, pp. 398–428, 2018, doi: 10.1108/JMTM-09-2017-0188.

[3] M. Padrón, M. D. L. a. Irizarry, P. Resto, and H. P. Mejía, “A methodology for cost-oriented assembly line balancing problems,” *J. Manuf. Technol. Manag.*, vol. 20, no. 8, pp. 1147–1165, 2009, doi: 10.1108/17410380910997254.

[4] B. Das, J. M. Sanchez-Rivas, A. Garcia-Diaz, and C. A. MacDonald, “A computer simulation approach to evaluating assembly line balancing with variable operation times,” *J. Manuf. Technol. Manag.*, vol. 21, no. 7, pp. 872–887, 2010, doi: 10.1108/17410381011077964.

[5] N. Boysen, M. Fliedner, and A. Scholl, “A classification of assembly line balancing problems,” *Eur. J. Oper. Res.*, vol. 183, no. 2, pp. 674–693, Dec. 2007, doi: 10.1016/j.ejor.2006.10.010.

[6] P. Sivasankaran and P. Shahabudeen, “Literature review of assembly line balancing problems,” *Int. J. Adv. Manuf. Technol.*, vol. 73, no. 9–12, pp. 1665–1694, 2014, doi: 10.1007/s00170-014-5944-y.

[7] B. Sungur and Y. Yavuz, “Assembly line balancing with hierarchical worker assignment,” *J. Manuf. Syst.*, vol. 37, pp. 290–298, Oct. 2015, doi: 10.1016/j.jmsy.2014.08.004.

[8] A. Corominas, L. Ferrer, and R. Pastor, “Assembly line balancing: General resource-constrained case,” *Int. J. Prod. Res.*, vol. 49, no. 12, pp. 3527–3542, Jun. 2011, doi: 10.1080/00207543.2010.481294.

[9] K. Ağpak, H. Gökçen, K. Ağpak, and H. Gökçen, “Assembly line balancing: Two resource constrained cases,” *Int. J. Prod. Econ.*, vol. 96, no. 1, pp. 129–140, Apr. 2005, doi: 10.1016/j.ijpe.2004.03.008.

[10] M. J. Nilakantan, G. Q. Huang, and S. G. Ponnambalam, “An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems,” *J. Clean. Prod.*, vol. 90, pp. 311–325, 2015, doi: 10.1016/j.jclepro.2014.11.041.

[11] J. . Nilakantan, S. G. Ponnambalam, and G. Q. Huang, “Minimizing energy consumption in a U-shaped robotic assembly line,” in *2015 International Conference on Advanced Mechatronic Systems (ICAMechS)*, 2015, pp. 119–124.

[12] G. Michalos, A. Fysikopoulos, S. Makris, D. Mourtzis, and G. Chryssolouris, “Multi criteria assembly line design and configuration - An automotive case study,” *CIRP J. Manuf. Sci. Technol.*, vol. 9, pp. 69–87, 2015, doi: 10.1016/j.cirpj.2015.01.002.

[13] Z. Li, Q. Tang, and L. Zhang, “Minimizing energy consumption and cycle time in two-sided robotic assembly line systems using restarted simulated annealing algorithm,” *J. Clean. Prod.*, vol. 135, pp. 508–522, 2016, doi: 10.1016/j.jclepro.2016.06.131.

[14] T. L. Urban and W. C. Chiang, “Designing energy-efficient serial production lines: The unpaced synchronous line-balancing problem,” *Eur. J. Oper. Res.*, vol. 248, no. 3, pp. 789–801, 2016, doi: 10.1016/j.ejor.2015.07.015.

[15] M. R. Abdullah Make, M. F. F. Ab. Rashid, and M. M. Razali, “A review of two-sided assembly line balancing problem,” *Int. J. Adv. Manuf. Technol.*, vol. 89, no. 5–8, pp. 1743–1763, Aug. 2017, doi: 10.1007/s00170-016-9158-3.

[16] E. Falkenauer and A. Delchambre, “A genetic algorithm for bin packing and line balancing,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 1992, vol. 2.

[17] 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, 2002.

[18] S. Akpınar and G. M. Bayhan, “A hybrid genetic algorithm for mixed model assembly line balancing problem with parallel workstations and zoning constraints,” *Eng. Appl. Artif. Intell.*, vol. 24, no. 3, pp. 449–457, 2011, doi: 10.1016/j.engappai.2010.08.006.

[19] S. Ö. Tasan and S. Tunali, “Improving the Genetic Algorithms Performance in Simple Assembly Line Balancing,” in *Proceedings of the 2006 International Conference on Computational Science and Its Applications - Volume Part V*, 2006, pp. 78–87, doi: 10.1007/11751649_9.

[20] 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, doi: 10.1016/j.cie.2008.09.027.

[21] M. H. Alavidoost, M. Tarimoradi, and M. H. F. Zarandi, “Fuzzy adaptive genetic algorithm for multi-objective assembly line balancing problems,” *Appl. Soft Comput.*, vol. 34, pp. 655–677, 2015.

[22] U. Saif, Z. Guan, L. Zhang, J. Mirza, and Y. Lei, “Hybrid Pareto artificial bee colony algorithm for assembly line balancing with task time variations,” *Int. J. Comput. Integr. Manuf.*, vol. 30, no. 2–3, pp. 255–270, 2017, doi: 10.1080/0951192X.2016.1145802.

[23] Q. Tang, Z. Li, and L. Zhang, “An effective discrete artificial bee colony algorithm with idle time reduction techniques for two-sided assembly line balancing problem of type-II,” *Comput. Ind. Eng.*, vol. 97, pp. 146–156, 2016, doi: 10.1016/j.cie.2016.05.004.

[24] Q. Tang, Z. Li, L. Zhang, and C. Zhang, “Balancing stochastic two-sided assembly line with multiple

constraints using hybrid teaching-learning-based optimization algorithm," *Comput. Oper. Res.*, vol. 82, pp. 102–113, 2017, doi: 10.1016/j.cor.2017.01.015.

[25] D. Li, C. Zhang, X. Shao, and W. Lin, "A multi-objective TLBO algorithm for balancing two-sided assembly line with multiple constraints," *J. Intell. Manuf.*, vol. 27, no. 4, pp. 725–739, 2016, doi: 10.1007/s10845-014-0919-2.

[26] S. Akpinar, A. Elmi, and T. Bektaş, "Combinatorial Benders cuts for assembly line balancing problems with setups," *Eur. J. Oper. Res.*, vol. 259, pp. 527–537, 2017, doi: http://dx.doi.org/10.1016/j.ejor.2016.11.001.

[27] Z. Li, N. Dey, A. S. Ashour, and Q. Tang, "Discrete cuckoo search algorithms for two-sided robotic assembly line balancing problem," *Neural Comput. Appl.*, vol. 30, no. 9, pp. 2685–2696, 2018, doi: 10.1007/s00521-017-2855-5.

[28] S. Mirjalili, "Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm," *Knowledge-Based Syst.*, vol. 89, pp. 228–249, 2015, doi: 10.1016/j.knosys.2015.07.006.

[29] X. Li, K. Xing, Y. Wu, X. Wang, and J. Luo, "Total Energy Consumption Optimization via Genetic Algorithm in Flexible Manufacturing Systems," *Comput. Ind. Eng.*, vol. 104, pp. 188–200, 2017, doi: 10.1016/j.cie.2016.12.008.

[30] S. Mirjalili, "SCA: A Sine Cosine Algorithm for solving optimization problems," *Knowledge-Based Syst.*, vol. 96, pp. 120–133, 2016, doi: 10.1016/j.knosys.2015.12.022.

[31] A. Scholl, "Benchmark Data Sets by Scholl," *Assembly Line Balancing Data Dets & Research Topics*, 1993. http://assembly-line-balancing.mansci.de/salbp/benchmark-data-sets-1993/.

[32] V. Bewick, L. Cheek, and J. Ball, "Statistics review 10: Further nonparametric methods," *Crit. Care*, vol. 8, no. 3, p. 196, 2004, doi: 10.1186/cc2857.

[33] A. J. Hayter, "The Maximum Familywise Error Rate of Fisher's Least Significant Difference Test," *J. Am. Stat. Assoc.*, vol. 81, no. 396, pp. 1000–1004, Dec. 1986, doi: 10.1080/01621459.1986.10478364.