8th IEEE International Conference on Automation Science and Engineering
August 20-24, 2012, Seoul, Korea

Scheduling Flexible Assembly Lines using Variants of Differential Evolution

Lui Wen Han Vincent[1] and S.G. Ponnambalam[2]

Abstract— **A schedule which optimally allocates scarce resources is important to improve the performance of a Flexible Assembly Line (FAL). This paper investigates the performance of different variants of a Differential Evolution-based algorithm in solving a FAL scheduling problem. The performance of the FAL is evaluated based on the weighted sum of Earliness/Tardiness (E/T) penalties and the balance of the FAL. Using a real production data set, the control parameters of the different variants are tuned extensively to improve the performance of the algorithm. A comparison between the different variants of the Differential Evolution algorithm with current work in literature shows the superiority of DE variants.**

I. INTRODUCTION

A Flexible Assembly Line (FAL) is a set of assembly stations that are interlinked with an automated material handling system. It is flexible because the assembly stations are capable of handling operations for different product types. In FALs, scheduling has always been a difficult problem. A schedule not only helps to allocate resources over time, it also has to ensure the performance of the FAL is optimized and the constraints of the system are not violated. Compared to other scheduling problems such as single machine scheduling and flow shop scheduling, FAL scheduling is more challenging. This is because the flexible nature of the assembly stations renders many different solutions.

In FAL scheduling, researchers often investigate techniques to produce schedules which maximizes the balance of the FAL. This is often achieved by perturbing the assignment of operations to assembly stations in the FAL. Due to the advent of the Just-in-TIme (JIT) production concept, schedules which minimize the Earliness/Tardiness (E/T) penalties are becoming equally important in recent years. There are three general methods to solve scheduling problems, namely efficient optimal methods, enumeration methods, and heuristic methods [1]. The efficient optimal method is only limited to a small class of simple problems, whereas the enumeration method entails huge computational burden when the search space becomes large. On the other hand, heuristic methods such as Particle Swarm Optimization (PSO) [2], Genetic Algorithm (GA) [3], and Differential Evolution (DE) [4] offer a compromise between the quality of solution and computational efficiency.

In FAL scheduling, although the structure of the problem can be exploited, the attractiveness of heuristic methods lies in their ease of implementation. There are not many publications which use heuristic methods to solve FAL scheduling problems. Guo et al. [5] proposed to solve the FAL scheduling problem with respect to two performance criteria: the weighted sum of E/T penalties and the balance of the FAL using a Bi-level Genetic Algorithm (BIGA). While encouraging results were reported, several assumptions in the paper have greatly simplified the problem. Nonetheless, this paper provides a good framework for future modifications. Guo et al. [6] addressed the FAL balancing problem by proposing a slightly different FAL model by removing the assumption that a workstation spends an equal amount of time processing operations when more than one operation is assigned to that workstation. This work was further enhanced in [7] when a time learning curve model was introduced to remove the assumption that operator efficiencies are constant. Recently, a DE-based algorithm was proposed [8] to solve a FAL scheduling problem based on the FAL model in [5]. The results show that the DE-based algorithm produces solutions with a higher average fitness compared to BiGA [5].

This paper is an extension to the work in [8]. The FAL model has been reformulated to remove the assumption that a workstation spends an equal amount of time processing operations when more than one operation is assigned to that workstation. This reformulation is different compared to the work by Guo et al. [6]. In [6], the authors were addressing the FAL balancing problem while this paper is focused on the combination of the weighted sum of E/T penalties and the balance of the FAL. Hence, the objective functions used are different. In comparison to the work in [5], this paper reformulates the equations to calculate the completion time of orders as no information was provided in [5] to calculate the completion time of orders. With a reformulation of the FAL model, the DE-based algorithm in [8] is extended. In particular, the algorithm is extended to optimize the amount of time a workstation spends on its operations. The remaining of this paper is organized as follows: Section 2 describes the problem formulation and Section 3 describes the DE-based algorithm. In Section 4, the results are shown and discussed. Finally, the paper is concluded in Section 5.

II. PROBLEM FORMULATION

This section describes the mathematical model of the FAL scheduling problem considered in this paper. The nomenclature in this paper can be found in Table I.

[1]Lui Wen Han Vincent is with the Department of Electrical Engineering, Monash University, Australia whlui1@student.monash.edu

[2]S.G. Ponnambalam is with the School of Engineering, Monash University, Sunway, Malaysia sgponnambalam@monash.edu

TABLE I
NOMENCLATURE

Notation	Description
P_n	The n^{th} production order
O_{nl}	The l^{th} operation of the n^{th} order
M_{kj}	j^{th} machine of the k^{th} machine type
S_{nl}	Actual starting time of operation O_{nl}
X_{nlkj}	Indicates whether operation O_{nl} is assigned to machine M_{kj} (If so, X_{nlkj} = 1. Otherwise, X_{nlkj} = 0)
α_n	Tardiness weight of order P_n
β_n	Earliness weight of order P_n
D_n	Due date of order P_n pre-determined by customer
F_n	Actual completion time of order P_n
λ_n	Indicates if the tardiness of order P_n is greater than 1. (If so, λ_n = 1. Otherwise, λ_n = 0.)
sc	The number of scheduling statuses.
SPT_r	Processing time of the r^{th} scheduling status
SB_r	Balance index of the r^{th} scheduling status
SS_r	The set of orders involved in the r^{th} scheduling status
$m(m \geq 1)$	The number of orders in the r^{th} scheduling status
PB_{nr}	The balance index of order P_n in the r^{th} scheduling status
o_n	The number of operations of order P_n
OPT_{nl}	The average processing time of operation O_{nl}
MPT_{nr}	The maximum value of OPT_{nl} for the r^{th} scheduling status of the n^{th} order
ST_{nl}	Standard processing time of operation O_{nl}
EM_{nlkj}	Operator's efficiency to process operation O_{nl} on machine M_{kj}
η_{nlkj}	Fraction of time spent by machine M_{kj} on operation O_{nl}.
SM_{nl}	The set of workstations suitable for operation O_{nl}

A. FAL Environment

The FAL consists of a set of workstations which can be divided into two machine types. A workstation is a physical location which consists of a machine, an operator, and a buffer. In any instance, a machine can only process up to two operations, and an operator can only operate one machine.

The FAL has to process two production orders named order i and order j. There are products of different types, each required in different quantities for each order. Each machine type can process different operations, and these operations have to be performed according to their precedence relations.

For a FAL with two production orders, there are different sequences, called scheduling modes, in which the orders can be completed. In this paper, five scheduling modes are considered as shown in Fig. 1. In each scheduling mode, there are scheduling statuses which define the order that is being processed. Further, certain scheduling modes are special occurences of other modes. For example, if order i is completed at time t_1, mode (c) turns into mode (e). Hence, mode (c) is a special occurence of mode (e).

B. Objective Function

The quality of the generated schedule is measured through two performance criteria. Firstly, the weighted sum of E/T penalties has to be minimized. This criterion is defined as

$$Z(S_{n1}, X_{nlkj}) = \sum_{n=1}^{p} (\alpha_n \cdot (F_n - D_n) \cdot \lambda_n + \beta_n \cdot (D_n - F_n)(1 - \lambda_n)) \quad (1)$$

The actual completion times of each order, F_n, have to be computed for each scheduling mode so that the scheduling mode which minimizes the first criterion can be selected. In this paper, a general procedure is provided to perform this computation as such information was not provided in [8]. Recall that operations have to be processed according to their precedence relations. As such, the workstation which is processing the slowest operation dictates the time required to complete one unit of the order. This is because each operation, having a precedence relation, has to wait for the next operation to be completed. For example, assume there are five operations 1, 2,..., 5, whereby operation 1 has to be processed first, followed by operation 2, and so on. If operation 3 is the slowest operation and it is processed by workstation 2, all other workstations have to wait for workstation 2 to complete operation 3 before they can process their subsequent operations. Based on this reasoning, the general procedure to compute F_n is detailed in the next two paragraphs.

Firstly, for every order in each scheduling status, the slowest operation, MPT_{nr}, is identified, where

$$MPT_{nr} = \max\,(OPT_{nl}) \quad (2)$$

$$OPT_{nl} = \frac{ST_{nl}}{\sum_{kj} X_{nlkj} EM_{nlkj} \eta_{nlkj}} \quad (3)$$

Based on the values of MPT_{nr}, the values of SPT_r can be computed for each scheduling mode. The computation of SPT_r differs depending on which order has to be completed first. As an example, for scheduling mode (a), SPT_1 is the time t_1. At time t_1, the number of order 1 units completed is t_1/MPT_{11}. SPT_3 is the time required to complete the remaining order 1 units. It can be computed as P_1-(remaining order 1 units$/MPT_{13}$). The number of order 2 units that have been simultaneously processed at this time is SPT_3/MPT_{23}. Finally, SPT_2, the time to process the remaining order 2 units, can be computed as P_2-(remaining order 2 units$/MPT_{22}$).

With the values of SPT_r, the completion time of each order can be computed as follows:

$$F_i = SPT_1 + SPT_3; F_j = F_i + SPT_2 \quad (4)$$

$$F_i = \sum_{r=1}^{sc} SPT_r; F_j = SPT_1 + SPT_3 \quad (5)$$

$$F_i = SPT_1 + SPT_2; F_j = t_1 + SPT_2 \quad (6)$$

where (4) is for scheduling modes (a) and (d), (5) is for scheduling modes (b) and (d), and (6) is for scheduling modes (c) and (e). Note (4) and (5) are for more than one scheduling mode because mode (d) is a special occurence of modes (a) and (b).

The second performance criterion involves maximizing the balance of the FAL, which is measured through a balance index. The balance index is defined as

$$B\,(S_{n1}, X_{nlkj}) = \frac{\sum_r SPT_r \cdot SB_r}{\sum_r SPT_r} \quad (7)$$

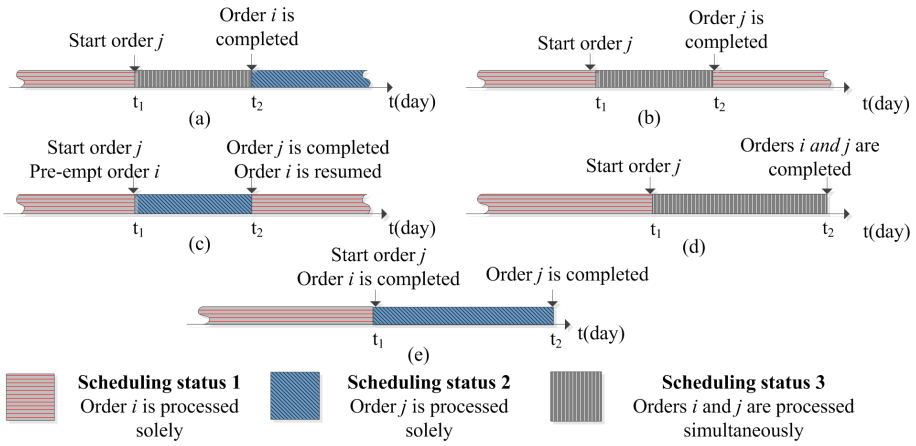

Fig. 1. Scheduling modes for FAL with two production orders [5]

where

$$SB_r = \sum_{n, P_n \in SS_r} \frac{PB_{nr}}{n} \tag{8}$$

Furthermore, the term PB_{nr} is defined as

$$PB_{nr} = \frac{\sum_i OPT_{nl}}{o_n \cdot max(OPT_{nl})} \tag{9}$$

Since the problem involves two performance criteria, Equations 1 and 7 have to be combined to form an objective function. The objective function is defined as

$$\begin{aligned} OBJ\left(S_{n1}, X_{nlkj}\right) =& Z\left(S_{n1}, X_{nlkj}\right) + \\ & 10 \cdot \left(1 - B\left(S_{n1}, X_{nlkj}\right)\right) \end{aligned} \tag{10}$$

where the term $\left(1 - B\left(S_{n1}, X_{nlkj}\right)\right)$ represents the imbalance of the FAL. A one percent decrease in the balance of the FAL incurs an imbalance penalty, which is set as 10. This term ensures that the objective function is a minimization function since (1) minimizes the E/T penalties whereas (7) maximizes the balance of the FAL. The value is set as 10, consistent with the work of Guo et al. [5], so that a direct comparison can be made between this work and theirs.

C. Assumptions

1) An operation cannot be interrupted once it has begun.
2) There is no shortage of materials, machine breakdown, and absence of operators in the FAL.
3) There is no Work-In-Process (WIP) in the workstations initially. The FAL is empty at time t = 0.

D. Constraints

1) *Allocation constraint:* Operation O_{nl} can only be assigned to workstations with the suitable machine type. This is defined as

$$\sum_{kj, M_{kj} \notin SM_{nl}} X_{nlkj} = 0 \tag{11}$$

In addition, each operation must be processed at least once. This is defined as

$$\sum_{kj} X_{nlkj} \geq 1 \tag{12}$$

2) *Operation precedence constraint:* An operation can only be started if its preceding operation is completed.

III. BI-LEVEL DIFFERENTIAL EVOLUTION

BiDE has two levels which work in tandem to generate a schedule which optimizes the performance of the FAL with respect to the criteria in Section II-B. The first level, called DEA1, optimizes the assignment of operations to workstations whereas the second level has two sub-levels, DEA2(a) and DEA2(b). **The sub-level DEA2(b) is the extension on the algorithm in [8]**. DEA2(a) optimizes the scheduling mode and starting time of each scheduling status, whereas DEA2(b) optimizes the proportion of time spent by a workstation on its corresponding operations if the workstation processes more than one operation at a time. For every vector in DEA1, there are NQ and NR number of vectors in DEA2(a) and DEA2(b), respectively. The best scheduling mode and starting time of each scheduling status for every DEA1 vector is chosen by its corresponding DEA2(a) and DEA2(b) vectors.

The pseudocode of the BiDE algorithm is shown in Algorithm 1. In the next subsections, the important features of BiDE are discussed.

A. Representation

Candidate solutions to the problem are represented through parameter vectors. A suitable representation is important because it affects all subsequent steps in BiDE.

In DEA1, each parameter vector represents a possible set of operation assignments to workstations for the three scheduling statuses considered in this paper. Hence, each vector consists of three sub-vectors. Every sub-vector represents the operation assignment in one scheduling status. The number of parameters in each sub-vector is equal to the number of workstations in the FAL. Each parameter represents a workstation; its value denotes the operation assigned to that workstation. As there are two machine types in the FAL, every sub-vector can be further divided into two parts, each part for a different combination of machine

Algorithm 1 BiDE

```
Randomly initialize population vectors for DEA1 and DEA2;
for i = 0 to GEN do
    Mutation for both levels;
    Crossover for both levels;
    Encode DEA1 vectors;
    Repair DEA1 vectors;
    Calculate the average processing time of operations, OPT_nl for every DEA1 vector using corresponding DEA2(b) vectors;
    Calculate weighted sum of E/T penalties and choose the best scheduling mode from DEA2(a) based on the values of OPT_nl;
    Selection for DEA2(a);
    Calculate balance index for vectors in DEA1;
    Evaluate fitness function for vectors in DEA1;
    Selection for DEA1 and DEA2(b);
end for
```

type and scheduling status. Thus, there are six parts in every parameter vector.

For DEA2(a), each parameter vector represents the starting time for every scheduling status. In this paper, order i is assumed to start at time t = 0. In this situation, the starting time of all scheduling statuses can be determined if the value of t_1 is known. Hence, t_1 is the only parameter of the vector.

In DEA2(b), each parameter vector represents the proportion of time spent by workstations on corresponding operations. Each vector has the same number of parameters as DEA1. If the workstation processes only one operation, the corresponding parameter in DEA2(b) is ignored because the workstation spends 100% of its time on the operation. If the workstation processes two operations, the value of the corresponding parameter denotes the processing time of the operation with the smaller number. The remaining time is spent processing the second operation.

B. Initialization

The parameter vectors are initialized randomly for both levels. For DEA1, each vector is initialized to be in the range [0,2]. If only one operation is assigned to the workstation, the parameters value is in the range [0,1]. If two operations are assigned to the workstation, the parameters value is in the range [1,2]. This process is defined by

$$x_{i,1} = \begin{cases} \text{rand(0,1)} & \text{if } R_i < 0.5 \\ \text{rand(1,2)} & \text{otherwise} \end{cases} \tag{13}$$

where $R_i \in [0, 1]$ is a random number, while rand(0,1) and rand(1,2) are uniformly distributed random numbers in the range [0,1] and [0,2] respectively.

For DEA2(a), the value of each parameter is initialized to be in the range [0, min(D_1, D_2)]. The time t_1 is restricted within the time required to complete the order with the shorter due date. For DEA2(b), the proportion of time spent by each workstation cannot exceed 1. Hence, each parameter is initialized randomly to values in the range [0,1].

C. Encoding

In scheduling, the most natural way to represent the candidate solutions are integer parameter vectors. For example, in a single machine scheduling problem, the vector [3,1,4,2,5] represents the job sequence. Job 3 is executed first as it is the first element in the vector. This is followed by job 1, 4, 2, and finally job 5. However, this form of representation cannot be used directly in BiDE because the mutation operation in DE involves the use of floating-point vectors. Instead, an encoding scheme is used to first map floating-point vectors to integer vectors. Through an encoding scheme, mutation and crossover can be performed using floating-point vectors before they are mapped to integer vectors.

The approach used in this paper is sub-range encoding [9]. Three modifications were made on the work in [9] to adapt sub-range encoding to BiDE. Firstly, recall that there are six parts in each parameter vector, each part representing a different combination of machine type and scheduling status. As a result, a different set of operations are allowed in each part of the vector. Hence, six sub-range arrays, $SR1, SR2, \ldots, SR6$, are used to represent the set of operations allowed in each part of the vector. Secondly, the original sub-range encoding scheme did not account for situations where a workstation can process up to two operations at a time. In this paper, if the value of the floating-point number is in the range [1,2], two numbers in the range [0,1] have to be generated to distinguish the two operations. This is achieved by applying the following equations:

$$\text{Operation (a)} = fp/2 \tag{14}$$

$$\text{Operation (b)} = (2 - \text{Operation(a)})\,/D_f \tag{15}$$

where fp is the existing floating point value and $D_f \in [1, 2]$ is a randomly selected floating- point number.

Finally, instead of checking the parameter vector as a whole, each part of the vector is checked with the set of operations that are allowed for that part. If there are any unassigned operations, the allocation constraint is violated—the part is invalid. In order to repair an invalid part, the following procedure is used:

1) Determine the operation with the highest number of occurrences in the part.
2) Replace the first occurrence of this operation with an unassigned operation.
3) Repeat steps 1–2 until there are no unassigned operations.

In the second level, since t_1 and the proportion of time spent by workstations are real values, the mutation operation can be applied directly to the vectors. Hence, an encoding scheme is not required for the second level.

D. Cost Function

Equation 10 can be used as the cost function of DEA1 since it is a minimization function. However, the Bi-level

Genetic Algorithm (BiGA) in literature uses a maximization/fitness function. In order to provide a meaningful comparison between BiDE and BiGA, the fitness function is used instead. This function is defined as

$$fitness = \frac{10}{OBJ\left(S_{n1}, X_{nlkj}\right) + 1} \tag{16}$$

DEA2(b) uses the same cost function as DEA1. Equation 1 is used as the cost function for DEA2(a). Given the time t_1, the best scheduling mode for each vector in DEA2 is the mode with the smallest weighted sum of E/T penalties. The DEA2(a) vectors are then compared among each other. The best vector, that is, the vector with the best scheduling mode and starting time of each scheduling status based on the operation assignments of the corresponding vector in DEA1, has the smallest value.

E. Variants of BiDE

Three variants of BiDE have been developed. The first variant, BiDE-1, assumes that the value of η_{nlkj} is equally divided between operations that are processed by the same workstation. The second variant, BiDE-2, relaxes the assumption made in the first variant. From the first two variants, the effect of DEA2(b) on the performance of BiDE can be investigated. Finally, the third variant, BiDE-3, builds on top of BiDE-2 by incorporating a local neighborhood mutation scheme [10]. This scheme attempts to balance the exploitative and explorative capabilities of BiDE by incorporating certain local neighborhoods aspects into BiDE.

F. Tuning of BiDE

In BiDE, DEA1, DEA2(a), and DEA2(b) can use a different set of parameter values. Hence, tuning BiDE is a difficult problem itself. An empirical approach is used to tune the parameters of BiDE. Further, the local neighborhood mutation scheme in BiDE-3 introduces four new control parameters in DEA1, DEA2(a), and DEA2(b): the scaling factors α and β, the scalar weight w, and the neighborhood radius k. In order to reduce the number of parameters to be tuned, several simplifications were made to each variant of BiDE:

1) BiDE-1 and BiDE-2:
   - The same values of F and C_R are used for DEA1, DEA2(a), and DEA2(b).
   - The population size in DEA2(b), is set as 1.
2) BiDE-3:
   - The values of α and β are the same as F.
   - The same values of α, β, k, and w are used for DEA1, DEA2(a), and DEA2(b).
   - The population size in DEA2(b), is set as 1.
   - A linear incremental scheme [11] is used to vary two parameters, w_{min} and w_{max} throughout the execution of the algorithm.

For BiDE-1 and BiDE-2, the control parameters are optimized sequentially. For each parameter, a set of values and schemes are selected to be evaluated. By optimizing each control parameter sequentially, a sub-optimal set of values and schemes can be found. This approach is similar to the work proposed by Nearchou and Omirou [9]. The sets of values and schemes are as follows:

1) $F = [0.5, 0.7, 0.9, \text{DETVSF}, \text{DERANDSF}]$
2) $C_R = [0.1, 0.3, 0.5, 0.7, 0.9, \text{Dynamic}C_R]$
3) $NP = [50, 100, 200]$
4) $NQ = [10, 20, 30]$
5) $GEN = [500, 1000, 2000, 3000]$
6) Mutation scheme = [DE/rand/1, DE/best/1, DE/target-to-best/1, DE/best/2, DE/rand/2]

where DETVSF (DE with time-varying scale factor) and DERANDSF (DE with random scale factor) are dynamic mutation scale factors [12], whereas dynamic C_R is a dynamic crossover rate [9].

To tune the parameters sequentially, different combinations of F and C_R values are first tested with $NP = 200$, $NQ = 30$, mutation scheme = DE/best/1 and $GEN = 500$. Using the optimal combination of F and C_R values, different combinations of NP and NQ values are tested while the same GEN values and mutation schemes are used. Finally, different values of GEN are evaluated, followed by different mutation schemes. For every combination, 10 test runs are conducted. The combination which produces the highest mean best fitness and lowest standard deviation is the best combination. For BiDE-3, the optimal parameter values of BiDE-2 are used, except for GEN. The explorative capability of BiDE-3 delays its convergence. Hence, the value of GEN has to be optimized along with values of k, $w_{\min}$, and $w_{\max}$. The values to be evaluated for k are [10,20,30,50] whereas the values tested for GEN are [500,1000,2000,3000].

IV. RESULTS AND DISCUSSION

BiDE is evaluated using two real production data set from Guo, et. al. [8]. A different experiment is conducted for each data set. In Experiment 1, two production orders have to be completed. There are two different cases in this experiment. In Case 1, a workstation can only process one operation at a time. In contrast, a workstation can process up to two operations at a time in Case 2.

As in Experiment 1, there are two production orders in Experiment 2 but there are three different cases. In each case, a workstation can process up to two operations at a time. The order due dates are different in every case:

1) Case 1: order 1 = 28 days, order 2 = 32 days.
2) Case 2: order 1 = 32 days, order 2 = 28 days.
3) Case 3: order 1 = 32 days, order 2 = 20 days.

A. Sub-optimal Values of Control Parameters

The best neighborhood radius k is 20. The values $w_{\min} = 0.6$ and $w_{\max} = 1$ were used. They produce reasonable results, but more tests have to be performed on w_{min} and w_{max} in the future. Due to space constraints, the graphs which show the relationship between BiDE's performance and the parameters' values are not shown in this paper. However, the authors observed the combination of $F = DETVSF$ and $C_R = 0.5$ is clearly superior compared

TABLE II
OPTIMAL VALUES OF BIDE PARAMETERS

Variant	NP	NQ	F_{init}	C_R	GEN	k
BiDE-1	200	30	0.5	0.5	2000	/
BiDE-2	200	30	0.5	0.5	3000	/
BiDE-3	200	30	0.8	0.5	3000	20

TABLE III
COMPARISON BETWEEN BIDE AND BIGA

Experiment		Mean Best Fitness ± standard deviation		
	BiGA	BiDE-1	BiDE-2	BiDE-3
E1C1	1.580	4.44±0.33	4.54±0.40	**4.58±0.120**
E1C2	2.68	**4.53±0.30**	4.38±0.20	4.44±0.24
E2E1	1.42	2.92±0.54	4.07±0.38	**4.16±0.25**
E2E2	3.73	3.59±0.10	**4.04±0.26**2	4.019±0.13
E2E3	0.97	1.28±0.65	**4.79±0.56**	3.85±0.37

to other combinations. In addition, in general, the greater the size of the population, the better the performance of BiDE. However, since a population size of $NP = 200$ and $NQ = 30$ produces good results already, there is no point in increasing the population size further as this will come at the expense of computation time. Finally, after 2000 generations, there is no noticeable improvement in the performance of the algorithm, except for variant 3 of the BiDE algorithm. The sub-optimal values of all the parameters for all variants of BiDE are shown in Table II.

B. Performance of BiDE

10 test runs are conducted for each variant of BiDE. The mean best fitness and standard deviation for each variant of BiDE are compared with BiGA, as shown in Table III. The standard deviation for BiGA is not shown as it was not provided in [5].

The results indicate that BiDE is superior compared to BiGA. BiDE has achieved a higher mean best fitness compared to BiGA except for Case 2 of Experiment 2 in one variant. If the mean best fitness is subtracted with the standard deviation, the mean best fitness of BiDE is still greater compared to BiGA.

Next, the performance of the BiDE variants is compared. Both BiDE-2 and BiDE-3 have achieved higher values of mean best fitness except for Case 2 of Experiment 1. For all cases of Experiment 2, the mean best fitness minus standard deviation for BiDE-2 and BiDE-3 are still greater compared to BiDE-1. This shows that the performance of BiDE has been improved by introducing DEA2(b) into the algorithm.

Further, the performance of BiDE-2 is compared with BiDE-3. The results show that BiDE-3 marginally outperforms BiDE-2. BiDE-3 achieved a higher value of mean best fitness in Experiment 1 and Case 1 of Experiment 2. In Case 2 of Experiment 2, although the mean best fitness of BiDE-3 is slightly lower, the results of BiDE-3 exhibit lower standard deviation values. It is is observed that the introduction of the local neighborhood mutation scheme further improves the performance of BiDE.

Finally, the code for the BiDE algorithm is not optimized. In an average of 10 test runs variant 1 requires 60s, variant 2 requires 120.7s, and variant 3 requires 160.6s. Although the time complexity of BiDE is in the order of minutes, there is no real time requirement in the generation of schedules. As long as a good schedule can be produced in a reasonable amount of time, it can help to improve the performance of the FAL. Unfortunately, this cannot be compared to the BiGA algorithm because no information was provided.

V. CONCLUSION

This paper has investigated the performance of a Bi-level Differential Evolution algorithm in solving a FAL scheduling problem. Three variants of the BiDE algorithm have been developed and evaluated. The experimental results show that all the variants of BiDE are capable of producing better results with a higher mean best fitness compared to current work in literature. Among the BiDE variants, the BiDE-3 variant produces the best results. This variant relaxes the assumption that a workstation spends an equal amount of time on shared operations using an additional optimization process. In addition, it uses a local neighborhood mutation scheme to improve its explorative and exploitative capabilities.

REFERENCES

[1] B. Maccarthy, "Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling," *International journal of production research*, vol. 31, pp. 59–79, 1993.

[2] C.-J. Liao, C.-T. Tseng, and P. Luarn, "A discrete version of particle swarm optimization for flowshop scheduling problems," *Computers & Operations Research*, vol. 34, pp. 3099–3111, 2005.

[3] O. Etiler, B. Toklu, M. Atak, and J. Wilson, "A genetic algorithm for flow shop scheduling problems," *Journal of the Operational Research Society*, vol. 55, pp. 830–835, 2004.

[4] R. Storn and K. Price, "Differential evolution — a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1996.

[5] Z. Guo, W. Wong, S. Leung, and J. Fan, "A genetic-algorithm-based optimization model for scheduling flexible assembly lines," *Int J Adv Manuf Technol*, vol. 36, pp. 156–168, 2008.

[6] Z. Guo, W. Wong, S. Leung, J. Fan, and S. Chan, "A genetic-algorithm-based optimization model for solving the flexible assembly line balancing problem with work sharing and workstation revisiting," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 38, pp. 218–228, 2008.

[7] Z. Guo, W. Wong, S. Leung, and J. Fan, "Intelligent production control decision support system for flexible assembly lines," *Expert Systems with Applications*, vol. 36, pp. 4268–4277, 2009.

[8] L. W. H. Vincent and S. G. Ponnambalam, "Scheduling flexible assembly lines using differential evolution," in *Proceedings of the second international conference on Swarm, Memetic, and Evolutionary Computing*, LCNS 7076, 2011, pp. 43–50.

[9] A. Nearchou and S. Omirou, "Differential evolution for sequencing and scheduling optimization," *Journal of heuristics*, vol. 12, pp. 395–411, 2006.

[10] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar, "Differential evolution using a neighbourhood-based mutation operator," *IEEE Transactions on Evolutionary Computation*, vol. 13, pp. 526–553, 2009.

[11] S. Das and P. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, pp. 4–31, 2011.

[12] S. Das, U. Chakraborty, and A. Konar, "Two improved differential evolution schemes for faster global search," in *Genetic and Evolutionary Computation Conference*, vol. 13, 2005, pp. 991–998.