2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

# A Discrete Artificial Bee Colony Algorithm for the Assignment and Parallel Machine Scheduling Problem in DYO Paint Company

Damla Kizilay,
Dept of Engineering,
Yasar University,
İzmir, Turkey,
damla.kizilay@yasar.edu.tr

M. Fatih Tasgetiren,
Dept of Engineering,
Yasar University,
İzmir, Turkey,
fatih.tasgetiren@yasar.edu.tr

Onder Bulut,
Dept of Engineering
Yasar University,
İzmir, Turkey,
onder.bulut@yasar.edu.tr

Bilgehan Bostan
Production Planning Manager
DYO Paint Company
İzmir, Turkey
bilgehan.bostan@dyo.com.tr

***Abstract*—This paper presents a discrete artificial bee colony algorithm to solve the assignment and parallel machine scheduling problem in DYO paint company. The aim of this paper is to develop some algorithms to be employed in the DYO paint company by using their real-life data in the future. Currently, in the DYO paint company; there exist three types of filling machines groups. These are automatic, semiautomatic and manual machine groups, where there are several numbers of identical machines. The problem is to first assign the filling production orders (jobs) to machine groups. Then, filling production orders assigned to each machine group should be scheduled on identical parallel machines to minimize the sum of makespan and the total weighted tardiness. We also develop a traditional genetic algorithm to solve the same problem. The computational results show that the DABC algorithm outperforms the GA on set of benchmark problems we have generated.**

## I. INTRODUCTION

THE DYO paint company produces several number of industrial paints for Turkish market. The DYO paint company is the first domestic brand of Turkish paint industry. In the DYO Paint Company, there are different types of paints, different types of packages and three types of filling machines groups, namely, automatic, semiautomatic and manual, where there are several number of identical machines. Currently, they face two main problems in the factory. The first one is how to assign the jobs to machine groups, and the second one is how to schedule these assigned jobs on the identical parallel machines. These two main problems are carried out by human expertise based on experience without employing any intelligent algorithm. Due to the poor assignment and scheduling of jobs in the paint filling unit, the machine utilization is very low and customer orders are mostly not satisfied on time.

Parallel machine scheduling has a wide range of literature. Since the parallel machines are identical in the DYO paint company, the literature related to identical parallel machines is given below.

In parallel machine scheduling problem, $n$ jobs are scheduled through $m$ machines in parallel and different performance measures such as total flow time, maximum completion time (makespan) and total tardiness are considered.

For the single machine systems, it is stated that minimizing the total tardiness is NP-hard problem in [1], therefore, the parallel machine problem with total tardiness criterion is strongly NP-hard, too [2]. This paper considers the scheduling problem involving identical parallel machines where the objective is the minimization of makespan and total weighted tardiness of all jobs. By using the notation in [3-4], $P_m//\sum T_i$ denotes that $P_m$ is the identical parallel machine problem, and $\sum T_i$ is the total tardiness. Regarding the makespan and total flowtime criteria, $P_m//C_{max}$ and $P_m//\sum \mathrm{w_j} C_j$ are both NP-hard, too [5-6].

The first study about parallel machine was at the end of the 50s in [7]. Then, some priority rules are developed to assign jobs to the identical parallel machines in [8]. Some exact algorithms can be found in [9-12]. However, due to the NP-Hard nature of these problems, many researchers have concentrated on heuristic methods. Most heuristic methods are based on List Scheduling, in which the jobs are sorted using a rule and based on this rule, they are assigned to the machines according to their earliest time to finish their operations. These kinds of heuristic methods are studied in [13]-[16]. In addition, a decomposition heuristic and hybrid simulated annealing heuristic are proposed in [17]. Also, for the scheduling problem in parallel machines which has objective to minimize the total tardiness, a genetic algorithm was used in [18]. For minimizing the completion time of the parallel machine flowshop scheduling problem, a tabu search was used in [19]. The tabu search and simulated annealing algorithms were compared in [20]. Recently, a hybrid heuristic algorithm was proposed in [21]. In order to minimize total tardiness of parallel machine problems, that include non-cumulative setup times, a tabu search was used in [22].

The remaining paper is organized as follows. Section 2 introduces the problem definition whereas Section 3 introduces the discrete artificial bee algorithm. The details of the genetic algorithm proposed for the problem is provided in Section 4. Section 5 discusses the computational results over

benchmark problems in total of both objectives – makespan and total flowtime. Finally, Section 6 summarizes the concluding remarks.

## II. Problem Definition

The problem will be handled by considering two stages: Assignment of the jobs to the machine groups and then scheduling of the partial jobs in each machine group having identical parallel machines.

The generalized assignment problem (GAP) is to find a minimum cost assignment of jobs to agents such that each job is assigned to exactly one agent, subject to capacity constraint of agents [23-25]. The mathematical model of GAP can be described as follows. Let $I = \{1,2, \dots, m\}$ be a set of agents, and let $J = \{1,2, \dots, n\}$ be a set of jobs. Define $c_{ij}$ as the cost of assigning job $j$ to agent $i$ (or assigning agent $i$ to job $j$); $r_{ij}$ as the resource required by agent $i$ to perform job $j$, and $b_i$ as the resource availability (capacity) of agent $i$. In addition, $x_{ij}$ is a $0-1$ variable that is 1 if agent $i$ performs job $j$ and 0 otherwise. The standard integer mathematical formulation of the GAP is as follows:

$$min\, Z = \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} \quad (1)$$

$$s.t:$$

$$\sum_{i\in I} x_{ij} = 1 \;\; \forall j \in J \quad (2)$$

$$\sum_{j\in J} r_{ij}x_{ij} \le b_i \;\; \forall i \in I \quad (3)$$

$$x_{ij} \in \{0,1\} \;\; \forall i \in I \;\; \forall j \in J \quad (4)$$

Constraint (2) ensures that each ensures that each job is assigned to exactly one agent and constraint (3) ensures that the total resource requirement of the jobs assigned to an agent does not exceed the capacity of the agent. The GAP may have no solution, and even the problem of finding a feasible assignment is NP-complete [26].

The assignment problem in DYO is similar to the GAP. Jobs and agents can be considered as production orders and parallel machine groups, respectively. Processing times of the jobs can be considered as the total resource requirement and capacity of each machine group can be considered as the capacity of agents. However, it is very difficult to determine the cost of assigning jobs to machine groups in our problem on hand. By inspiring from the above GAP formulation, we develop a heuristic solution representation and the problem formulation by defining an agent $x$ (machine groups) as follows:

- $j$ = number of jobs, $j = \{1,2,..,n\}$
- $i$ = number of parallel machine groups, $i = 1,2,..,g$
- $n_i$ = number of jobs assigned to each machine group $i$.
- For this reason, $n = \sum_{i=1}^{g} n_i$.
- $\pi_{ij}^{i}$ = partial set of jobs assigned to each machine group $i$.
- $\pi_{ij}$ =job $j$ assigned to machine group $i$.
- $p_{ij}$ = processing time of job $j$ on machine group $i$.
- $d_{ij}$ = due date of job $j$ on machine group $i$.
- $w_{ij}$ = weight of job $j$ on machine group $i$.

Since DYO schedules 80 jobs every day and finish them to completion in one shift, the ready times are considered to be zero.

Once the assignments are made and partial job sets $\pi_i's$ are determined by the proposed algorithms, the completion time of jobs $C_{\pi_{ij}}$ and the tardiness $T_{\pi_{ij}}$ can be computed as follows:

$$C_{max} = max\left\{C_{\pi_{ij}}\right\}. \quad (5)$$

$$T_{\pi_{ij}} = max\left\{0, C_{\pi_{ij}} - d_{\pi_{ij}}\right\}. \quad (6)$$

$$TWT = \sum_{i=1}^{g}\sum_{j=1}^{n} w_{\pi_{ij}}T_{\pi_{ij}}. \quad (7)$$

$$C_{max}^{max} = max\left(C_{max}(\pi_i)\right) \;\; i = 1,..,g \quad (8)$$

Since the objective function is bi-objective, we give a weight α to the first part of the objective function whereas $1-\alpha$ is given to the second part as follows:

$$min\, f(\pi) = \alpha(C_{max}^{max}) + (1-\alpha)TWT \quad (9)$$

The first part tries to maximize the machine utilization and the second part tries to minimize the total weighted tardiness. To solve the problem described above, we propose a DABC and a GA and their details are given in subsequent sections.

## III. Discrete Artificial Bee Colony Algorithm

### A. Traditional ABC Algorithm

In the ABC model, the colony consists of three groups of bees: employed bees, onlookers and scouts [27]. The number of solutions in the population is equal to the number of food sources and represented by D-dimensional real-valued vector. The ABC algorithm is stated to be an iterative process [27-32]. The ABC algorithm can be summarized as follows:

Initial food sources of the basic ABC algorithm are randomly created according to the range of the boundaries as follows:

$$x_{ij} = x_j^{min} + \left(x_j^{max} - x_j^{min}\right) \times r \quad (10)$$

$NP$ represents the number of food sources, namely, $i = 1,\dots,NP$; $D$ represents the number of decision variables, $j = 1,..,D$; and $r$ represents a uniform random number between 0 and 1. In the initial population a counter value 0 is used for each food source, i.e. $count_i = 0$.

In the employed bee phase, the neighboring food source is generated as follows:

$$v_{ij} = x_{ij} + \emptyset_{ij}\left(x_{ij} - x_{kj}\right) \quad (11)$$

Where $j \in \{1,..,D\}$ and $k \in \{1,..,NP\}$. $\emptyset_{ij}$ is a uniform random number which is generated in the range $[-1,1]$. If $v_{ij}$ does not fit the boundaries, it is restricted to initial range with equation 10. Then its fitness value is obtained as follows:

$$fitness_i = \begin{cases} 1/(1+f_i) & if\ f_i \geq 0 \\ 1+abs(f_i) & if\ f_i < 0 \end{cases} \quad (12)$$

If the new food source $v_i$ is better than $x_i$, it is replaced by $v_i$ in a greedy manner. The counter $count_i$ is kept as 0, else it is increased by 1. For all the employed bees in the population, the same process goes over again.

In the onlooker bee phase, the roulette wheel selection is used and for the each food source a probability is generated as follows:

$$p_i = \frac{fitness_i}{\sum_{i=1}^{NP} fitness_i} \quad (13)$$

A uniform random number r is generated in the range [0,1]. if $r$ is smaller than the probability $p_i$, a neighboring food source is generated with equation 11. If the new food source $v_i$ is better than $x_i$, it is replaced by $v_i$ in a greedy manner. The counter $count_i$ is kept as 0, else it is increased by 1. For all the onlooker bees in the population, same process goes over again.

In the scout bee phase, the sources abandoned are determined according to the counter of each solution. Determination is done through the comparison between the value of the counter $count_i$ and the control parameter "$limit$". Having greater $count_i$ than the "$limit$" means that the food source $x_i$ is abandoned. In order to provide diversification for the ABC algorithm, abandoned $x_i$ is forgotten and the new one is generated instead by using equation 10.

Because, the original ABC algorithm is designed for the real-parameter optimization problems, some modifications are needed for discrete/combinatorial problems. These modifications are explained below:

### B. Discrete ABC

In the discrete version, we still follow the basic framework of the original one as follows:

1. Initialize the population.
2. Employed bee phase to exploit the food sources.
3. Onlooker bee phase to search for new food sources.
4. Scout bee phase to search for new food sources.
5. Keep the best food source found so far.
6. If a termination criterion has not been satisfied, go to step 2; otherwise stop the procedure and report the best food source found so far.

### C. Solution Representation

We employ a unique solution representation inspired from the GAP. For 15 production orders and 3 machines groups, the solution representation is given in Fig. 1:

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\pi_{ij}$ | 3 | 2 | 2 | 1 | 1 | 3 | 1 | 3 | 3 | 1 | 2 | 1 | 3 | 2 | 2 |

Fig. 1. Solution Representation

In Fig. 1, $\pi_{1j} = 1$ represents the jobs $j$ assigned to manual machine group, $\pi_{2j} = 2$ represents the jobs $j$ assigned to semi-automatic machine group, and $\pi_{3j} = 3$ represents the jobs $j$ assigned to automatic machine group.

### D. Initial Population

In the DABC algorithm, the initial population is established randomly by assigning $\pi_{ij}$ values in the range [1,3]. For each food source in the population, one strategy amongst three is assigned to each food source randomly. These strategies generating new food sources will be explained later on.

### E. An Example

In the following example in Fig. 2, there are 15 jobs and 3 machine groups. Fig.2 indicates that the partial job set of machine groups will be determined as $\pi_{1j}^1 = \{4,5,7,10,12\}$ for manual machine group, $\pi_{2j}^2 = \{2,3,11,14,15\}$ for semi-automatic machine group, and $\pi_{3j}^3 = \{1,6,8,9,13\}$, respectively.

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\pi_{ij}$ | 3 | 2 | 2 | 1 | 1 | 3 | 1 | 3 | 3 | 1 | 2 | 1 | 3 | 2 | 2 |
| $p_{ij}$ | 2 | 5 | 4 | 9 | 7 | 1 | 10 | 2 | 3 | 8 | 5 | 9 | 1 | 4 | 6 |
| $d_{ij}$ | 3 | 8 | 6 | 15 | 11 | 2 | 15 | 3 | 5 | 12 | 8 | 15 | 2 | 6 | 9 |

Fig. 2. An Example of Solution representation

We assume that there are two parallel machines in each machine groups. So these partial job sets will be scheduled on parallel machine by using a list-scheduling approach as illustrated in Fig. 3 to Fig. 5 as follows:

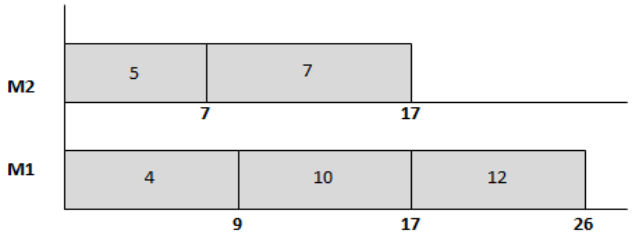


Fig. 3. Manual Machines

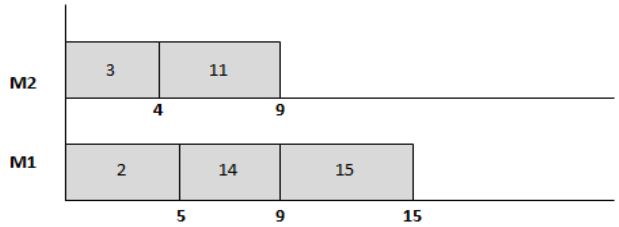


Fig. 4. Semi-automatic Machines

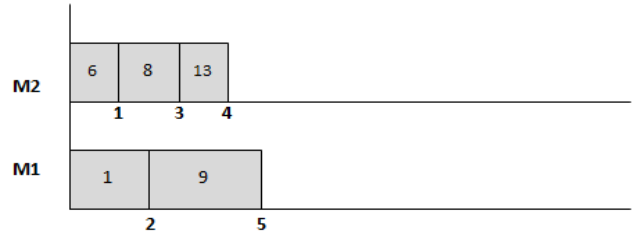


Fig. 5. Automatic Machines

### F. Neighborhood Structures

As the neighborhood structures, we employ shift and swap moves in the DABC algorithm. As shown in Fig. 6, $N_1(\pi) = shift(\pi)$ means that machine group assignment of job 4 is shifted from 1 to 3 and of job 12 from 1 to 3, too. Since the assignments are changed in the solution $\pi_{ij}$, the new partial solutions will be obtained as $\pi_{1j}^1 = \{5,7,10\}$, $\pi_{2j}^2 = \{2,3,11,14,15\}$, $\pi_{3j}^3 = \{1,4,6,8,9,12,13\}$, respectively.

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\pi_{ij}$ | 3 | 2 | 2 | **1** | 1 | 3 | 1 | 3 | 3 | 1 | 2 | **1** | 3 | 2 | 2 |
| $\pi_{ij}$ | 3 | 2 | 2 | **3** | 1 | 3 | 1 | 3 | 3 | 1 | 2 | **3** | 3 | 2 | 2 |

Fig. 6. Shift move

In the swap move, machine assignments of two randomly selected jobs will be exchanged. As shown in Fig. 7, $N_2(\pi) = swap(\pi)$ means that machine group assignment of job 4 and job 13 were exchanged. Again, because the assignments are changed in the solution $\pi_{ij}$, the new partial solutions will be obtained as $\pi_{1j}^1 = \{5,7,10,13\}$, $\pi_{2j}^2 = \{2,3,11,14,15\}$, $\pi_{3j}^3 = \{1,4,6,8,9,12\}$, respectively.

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\pi_{ij}$ | 3 | 2 | 2 | **1** | 1 | 3 | 1 | 3 | 3 | 1 | 2 | 1 | **3** | 2 | 2 |
| $\pi_{ij}$ | 3 | 2 | 2 | **3** | 1 | 3 | 1 | 3 | 3 | 1 | 2 | 3 | **1** | 2 | 2 |

Fig. 7. Swap move

### G. Employed Bee Phase

In the employed bee phase, new food sources are obtained through some strategies around the neighborhood of the current position. We employ three types of neighborhood structures. These structures are based on shift and swap operators. Using these strategies denoted as $S_i(\pi_{ij})$, new food sources in the neighborhood are obtained for the employed bees as follows:

$S_1$: Perform one shift, one swap move to solution $\pi_{ij}$.

$S_2$: Perform two shifts, two swap moves to solution $\pi_{ij}$.

$S_3$: Perform three shift, three swap moves to solution $\pi_{ij}$.

After obtaining a neighboring food source by employing strategy $S_i(\pi_{ij})$ assigned to the food source $i$, we apply a variable neighborhood search algorithm (VNS) [33] to the new food source to further enhance the solution quality. For the selection, a new source will always be accepted if it is better than the current food source.

The number of employed bees is taken as the population size $NP$. The VNS local search procedure will be explained in detail later on.

### H. Onlooker Bee Phase

In the onlooker bee phase, a food source $\pi_{kj}$ is determined by the tournament selection of size 2. In the tournament selection, two food sources are randomly chosen from the population, and the better one is chosen according to their fitness values ( $\pi_{kj} = TS_{best\ of\ two}(\pi_{kj} \in NP)$ ). Then, similar to the employed bee phase, corresponding strategy $S_k(\pi_{kj})$ is applied to the food source selected. After applying the corresponding strategy, the VNS algorithm is applied to the food sources.

The number of onlooker bees is taken as the population size $NP$. The VNS local search procedure will be explained in detail later on.

### I. Scout Bee Phase

In the scout bee phase, a tournament selection with the size of 2 is again used to discard the worse of two randomly selected food sources that have been picked out from the population. Then, the scout obtains a food source by the strategy assigned to it.

### J. Variable Neighborhood Search

The following VNS local search described in [34] is employed in both the employed bee and onlooker bee phases. The aim is to further improve the objective function on the partial job sets. Sequentially, the VNS local search is applied to each partial job set. As the neighborhood structures, single insert and swap move is applied to the permutation in the each partial job set. The VNS local search is given in Fig. 8.

$VNSListScheduling\left(\pi_{ij}^i \in (\pi_{ij})\right)$
$d_{max} = 2$
$\pi^* = \pi_{ij}^i$
$d = 1$
$do\{$
  $\pi_1 = N_d(\pi^*)$ % $N_1(\pi^*) = Insert(\pi^*)$
  $if\ f(\pi_1) < f(\pi^*)\ then$ % $N_2(\pi^*) = Swap(\pi^*)$
    $\pi^* = \pi_1$
    $d = 1$
  $else$
    $d = d + 1$
$\}while\ (d \le d_{max})$
$Return\ f(\pi^*)$

Fig. 8. Referenced Local Search

The complete computational procedure of the DABC algorithm is given in Fig. 9.

$Procedure\ DABC$

*Step 1.* $Set\ parameters\ NP = 10\ and\ S_{max} = 3$

*Step 2.* $Establish\ initial\ population\ randomly$

*Step 3.* $Assign\ a\ strategy\ to\ each\ food\ source\ randomly$
$by\ S_i(\pi_{ij}) = rand()\%S_{max}$

*Step 4.* $Evaluate\ population\ and\ find\ \pi_{best}$

Step 5. $Repeat\ the\ following\ for\ each\ employed\ bee$
$\pi_{ij}(Employed\ Bee\ Phase)$

a. $Generate\ a\ new\ food\ source\ by\ startegy$
$\pi_{new} = S_i(\pi_{ij})$

b. $Determine\ partial\ solutions\ \pi_{ij}^{i}$,

c. $For\ each\ i, apply\ \pi_{new} = VNS\ ListScheduling(\pi_{ij}^{i})$

d. $Compute\ objective\ function\ value, f(\pi_{new})$

e. $if\ f(\pi_{new}) < f(\pi_{ij}),\ \ \pi_{ij} = \pi_{new}$

f. $if\ f(\pi_{new}) < f(\pi_{best})\ then\ \ \pi_{best} = \pi_{new}$

*Step 6.* $Repeat\ the\ following\ for\ each\ onlooker\ bee\ x_k$ $(Onlooker\ Bee\ Phase)$

a. $Select\ a\ \ food\ source\ by\ tournament\ selection\ \pi_{kj} = \ \ TS_{best\ of\ two}(\pi_{kj} \in NP)$

b. $Generate\ a\ new\ food\ source\ by\ startegy\ S_k$ $\pi_{new} = \ \ S_k(\pi_{kj})$

c. $Determine\ partial\ solutions\ \pi_{kj}^{k}$

d. $For\ each\ k, apply\ \pi_{new} = VNS\ ListScheduling(\pi_{kj}^{k})$

e. $Compute\ objective\ function\ value, f(\pi_{new})$

f. $if\ f(\pi_{new}) < f(\pi_{kj}),\ \pi_{kj} = \pi_{new}$

g. $if\ f(\pi_{new}) < f(\pi_{best}),\ \ \pi_{best} = \pi_{new}$

*Step 7.* $Repeat\ the\ following\ for\ each\ scout\ bee\ \pi_{kj}$ $(Scout\ Bee\ Phase)$

a. $Select\ a\ \ food\ source\ by\ tournament\ selection\ \pi_{kj} = \ \ TS_{worst\ of\ two}(\pi_{kj} \in NP)$

b. $Generate\ a\ new\ food\ source\ by\ startegy\ k$ $\pi_{new} = \ \ S_k(\pi_{kj})$

c. $\pi_{new} = Apply\ VNS\ ListScheduling(\pi_{new})$

d. $\pi_{kj} = \pi_{new}$

e. $if\ f(\pi_{new}) < f(\pi_{best}),\ \ \pi_{best} = \pi_{new}$

f. $If\ the\ stopping\ criterion\ is\ not\ met, got\ to\ Step\ 5, else\ stop\ and\ return\ \pi_{best}$

Fig. 9. Outline of the ABC algorithm

## IV. Genetic Algorithm

Genetic algorithms (GA) are a part of parallel search heuristics originated by the biological process of natural selection and evolution [35]. In GA optimization, solutions are coded into chromosomes in order to construct a population being evolved through generations. At each generation, we use crossover operator, which is a process of taking more than one parent solutions and producing a child solution from them. Then, mutation and perturbation occurs for some of the individuals. After that, they are gathered to select new individuals for next generation. This procedure is repeated until the stopping criterion is satisfied.

In the proposed GA, we select one individual in random and the second one is with the tournament selection of size 2 to mate them. By using them, we generate an offspring with PTL crossover operator [36] where two cut points are determined and machine groups in cut points are either copied to the front or the end of the individual. The PTL crossover is given in Fig. 10.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Parent 1 | 3 | 2 | 2 | 1 | **1** | **3** | **1** | **3** | **3** | **1** | 2 | 1 | 3 | 2 | 2 |
| Parent 2 | 2 | 1 | 1 | 3 | **2** | **3** | **1** | **2** | **1** | **3** | 3 | 2 | 2 | 1 | 3 |

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Offspring1 | **1** | **3** | **1** | **3** | **3** | **1** | 2 | 1 | 1 | 3 | 3 | 2 | 2 | 1 | 3 |
| Offspring 2 | 2 | 1 | 1 | 3 | 3 | 2 | 2 | 1 | 3 | **1** | **3** | **1** | **3** | **3** | **1** |

Fig. 10. PTL crossover operator

As a mutation operator, offspring is mutated with two shift and two swap strategy with a mutation probability $P_m = 1/n$. To consistent with the DABC algorithm, we take crossover probability as $P_m = 1$. After generating offspring population, selection is carried out by using the tournament selection with size 2 between current population and offspring population. This procedure is repeated until the same stopping criterion is achieved. The following computational procedure explains the components of the proposed GA:

Step 1. Set the population size $NP$,
Step 2. Initialize the population randomly:
Step 3. For $i = 1,2, \dots, NP$, repeat the following sub-steps:
- a. For the individual $\pi_{ij}$, select a mate $\pi_{kj}$ from the population by the tournament selection with size 2.
- b. Produce a new offspring $o_{ij}$ by recombining them with PTL crossover.
- c. Mutate $\pi_{ij}$ with a mutation probability $P_m = 1/n$.
- d. Evaluate the new offspring $o_{ij}$ and apply VNS List Scheduling to $o_{ij}$.

Step 4. Make selection between current population by using tournament selection with size of 2 and update best so far solution $\pi_{best}$.
Step 5. If the termination criterion is reached, return the best solution found so far $\pi_{best}$; otherwise go to Step 3.

## V. Computational Results

The DABC and GA algorithms were coded in Visual C++ and run on an Intel(R) Core(TM) i5-3360M 2.8 GHz with 8GB memory. We generated our own benchmark problems as follows: For automatic machines group, the processing times are generated between 5 and 11, for semi-automatic machines groups, processing times are generated between 11 and 16, for manual machines groups, processing times are generated between 16 and 21. We devised 10 instances for 100 job problems, 200 job problems, 300 job problems, 400 job problems and 500 job problems.

For each instance of each problem size, we carried out 5 replications and we provide the average (Avg), minimum (Min), maximum (Max) and the standard deviation of five runs of 10 instances for each problem category. We fixed the population size at 10 for the DABC and 20 for the GA to make them have the same number of function evaluations. Since we have two objectives with the weight $\alpha$, we run both algorithms with $\alpha$ ranging from 0 to 1, i.e,

$\alpha = \{0.0,0.1,0.2,\ldots,1.0\}$. For the $\alpha = 0.0$, the computational results are given from Table I to Table XXII, respectively:

TABLE I

COMPUTATIONAL RESULTS OF DABC FOR $\alpha = 0.0$

| | DABC | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 618.60 | 583.10 | 709.70 | 53.58 | 58.58 |
| 200 | 3910.30 | 3536.40 | 4618.20 | 439.53 | 200.50 |
| 300 | 10519.50 | 9380.70 | 12188.10 | 1139.34 | 420.61 |
| 400 | 20306.70 | 18302.30 | 23086.50 | 1906.69 | 719.32 |
| 500 | 34060.50 | 30994.70 | 38144.20 | 2871.37 | 1185.57 |

TABLE II

COMPUTATIONAL RESULTS OF GA FOR $\alpha = 0.0$

| | GA | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 639.80 | 594.60 | 763.10 | 72.58 | 95.81 |
| 200 | 4397.60 | 3951.00 | 5081.90 | 462.92 | 328.09 |
| 300 | 11724.30 | 10771.60 | 13014.80 | 940.51 | 694.73 |
| 400 | 22840.40 | 21503.40 | 24252.10 | 1136.55 | 1186.21 |
| 500 | 37387.30 | 35755.00 | 39614.00 | 1538.86 | 1812.71 |

As seen in the tables above, in DABC algorithm, the results of Avg, Min, Max values are superior to GA. For example, the deviation for the largest problem size was 37387/34060=1.1 for Avg. value. It indicates that there is 10% improvement over GA. However, for larger size of problems, GA generated lower STD values, hence it was more robust than DABC.

TABLE III

COMPUTATIONAL RESULTS OF DABC FOR $\alpha = 0.1$

| | DABC | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 955,7 | 916,10 | 1060,10 | 62,45 | 59,59 |
| 200 | 6718,4 | 6042,80 | 7904,50 | 760,95 | 207,83 |
| 300 | 18386,2 | 16519,10 | 21395,90 | 1959,25 | 433,94 |
| 400 | 35886,9 | 32441,90 | 40375,50 | 3166,65 | 730,14 |
| 500 | 59532,4 | 53967,30 | 67117,70 | 5190,48 | 1125,24 |

TABLE IV

COMPUTATIONAL RESULTS OF GA FOR $\alpha = 0.1$

| | GA | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 1059,9 | 939,90 | 1312,90 | 159,65 | 96,27 |
| 200 | 7500,9 | 6743,70 | 8527,50 | 765,14 | 330,33 |
| 300 | 20691,8 | 18899,40 | 23154,30 | 1725,54 | 683,64 |
| 400 | 40149,9 | 37961,10 | 43288,60 | 2158,85 | 1160,89 |
| 500 | 67293,3 | 64563,10 | 70654,00 | 2538,63 | 1779,56 |

When the DABC and GA algorithms was compared according to $\alpha = 0.1$, results of the DABC algorithm yielded much better results in terms of Avg, Min, Max. For 500 jobs, the deviation was 67293/59532=1.13 meaning that it is a 13% improvement over GA. However, GA was more robust.

TABLE V

COMPUTATIONAL RESULTS OF DABC FOR $\alpha = 0.3$

| | DABC | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 794,4 | 746,50 | 908,70 | 66,72 | 68,60 |
| 200 | 5319,9 | 4797,30 | 6330,30 | 628,37 | 228,74 |
| 300 | 14354,9 | 12837,80 | 16580,50 | 1486,34 | 432,50 |
| 400 | 28428,6 | 25566,40 | 32250,20 | 2687,41 | 740,31 |
| 500 | 46493,5 | 42562,00 | 51923,90 | 3739,78 | 1132,64 |

TABLE VI

COMPUTATIONAL RESULTS OF GA FOR $\alpha = 0.3$

| | GA | | | | |
|---|---|---|---|---|---|
| **Jobs** | Avg | Min | Max | Std | CPU |
| 100 | 844,4 | 762,70 | 1014,50 | 107,17 | 98,94 |
| 200 | 6120 | 5477,80 | 6951,50 | 606,31 | 345,72 |
| 300 | 16268,4 | 14969,30 | 17859,10 | 1174,93 | 720,17 |
| 400 | 31762,8 | 30081,40 | 33760,00 | 1524,95 | 1177,26 |
| 500 | 51632,1 | 48675,20 | 54943,50 | 2540,89 | 1806,78 |

According to $\alpha = 0.3$, while DABC algorithm having better results than GA algorithm in terms of Avg, Min, Max values, GA was robust than DABC. The deviation for the largest problem was 51632/46493=1.11 indicating 11% improvement over GA.

TABLE VII

COMPUTATIONAL RESULTS OF DABC FOR $\alpha = 0.5$

| | DABC | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 618,6 | 583,10 | 709,70 | 53,58 | 58,58 |
| 200 | 3910,3 | 3536,40 | 4618,20 | 439,53 | 200,50 |
| 300 | 10519,5 | 9380,70 | 12188,10 | 1139,34 | 420,61 |
| 400 | 20306,7 | 18302,30 | 23086,50 | 1906,69 | 719,32 |
| 500 | 34060,5 | 30994,70 | 38144,20 | 2871,37 | 1185,57 |

TABLE VIII

COMPUTATIONAL RESULTS OF GA FOR $\alpha = 0.5$

| | GA | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 639,8 | 594,60 | 763,10 | 72,58 | 95,81 |
| 200 | 4397,6 | 3951,00 | 5081,90 | 462,92 | 328,09 |
| 300 | 11724,3 | 10771,60 | 13014,80 | 940,51 | 694,73 |
| 400 | 22840,4 | 21503,40 | 24252,10 | 1136,55 | 1186,21 |
| 500 | 37387,3 | 35755,00 | 39614,00 | 1538,86 | 1812,71 |

According to the tables for $\alpha = 0.5$, similar results are obtained. the DABC algorithm yielded approximately 10% improvement over GA again. But, GA was more robust than DABC because it generated lower STD values.

TABLE IX

COMPUTATIONAL RESULTS OF DABC FOR $\alpha = 0.7$

| | DABC | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 450,8 | 421,30 | 523,30 | 42,51 | 67,54 |
| 200 | 2483,1 | 2245,30 | 2887,10 | 264,60 | 222,30 |
| 300 | 6541,3 | 5835,50 | 7670,90 | 735,04 | 466,08 |
| 400 | 12460,8 | 11255,70 | 14195,30 | 1186,71 | 723,92 |
| 500 | 20559,5 | 18745,40 | 23133,90 | 1751,52 | 1125,24 |

TABLE X

COMPUTATIONAL RESULTS OF GA FOR $\alpha = 0.7$

| | GA | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 473,3 | 426,60 | 561,20 | 56,51 | 102,58 |
| 200 | 2769,5 | 2482,00 | 3188,90 | 291,11 | 340,53 |
| 300 | 7390,3 | 6892,70 | 8147,30 | 521,17 | 741,58 |
| 400 | 13999,5 | 13330,90 | 14884,40 | 642,58 | 1152,57 |
| 500 | 22875,4 | 21982,20 | 24100,50 | 888,76 | 1767,12 |

When the results in the tables IX and X are compared for the Avg, Min, Max values, again the DABC algorithm outperformed GA. However, GA algorithm is better than DABC algorithm in terms of STD values.

TABLE XI

COMPUTATIONAL RESULTS OF DABC FOR $\alpha = 0.9$

| | DABC | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 268,8 | 250,30 | 304,60 | 22,23 | 59,66 |
| 200 | 1070,1 | 965,50 | 1251,70 | 117,25 | 202,20 |
| 300 | 2540,6 | 2273,40 | 2938,20 | 267,37 | 421,99 |
| 400 | 4766,5 | 4321,30 | 5377,00 | 422,73 | 723,43 |
| 500 | 7584,7 | 6915,20 | 8466,90 | 620,10 | 1142,90 |

TABLE XII

COMPUTATIONAL RESULTS OF GA FOR $\alpha = 0.9$

| | GA | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 280,7 | 255,30 | 334,80 | 33,68 | 95,07 |
| 200 | 1199,5 | 1088,50 | 1354,20 | 110,01 | 308,85 |
| 300 | 2879,7 | 2701,90 | 3116,60 | 168,48 | 594,28 |
| 400 | 5322,1 | 5019,20 | 5612,40 | 233,62 | 1054,16 |
| 500 | 8452,9 | 8186,80 | 8831,20 | 265,56 | 1654,96 |

The results are given for $\alpha = 0.9$ in the tables above. The DABC algorithm again produced better results than GA. However, GA was more robust than DABC since it generated lower STD values.

TABLE XII

COMPUTATIONAL RESULTS OF DABC FOR $\alpha = 1.0$

| | DABC | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 179,9 | 165,90 | 202,60 | 15,42 | 40,75 |
| 200 | 377,6 | 344,30 | 422,50 | 32,44 | 120,28 |
| 300 | 583,1 | 537,50 | 644,00 | 42,84 | 263,48 |
| 400 | 798 | 740,60 | 864,70 | 49,05 | 436,75 |
| 500 | 1027,5 | 973,00 | 1085,60 | 45,03 | 579,37 |

TABLE XIV

COMPUTATIONAL RESULTS OF GA FOR $\alpha = 1.0$

| | GA | | | | |
|---|---|---|---|---|---|
| Jobs | Avg | Min | Max | Std | CPU |
| 100 | 193,7 | 181,00 | 213,00 | 13,30 | 67,63 |
| 200 | 413,3 | 395,70 | 436,60 | 17,67 | 192,59 |
| 300 | 639,1 | 622,60 | 663,60 | 17,25 | 370,26 |
| 400 | 869,3 | 857,90 | 885,90 | 11,92 | 621,52 |
| 500 | 1091,9 | 1083,90 | 1105,60 | 9,13 | 903,04 |

As seen in the tables above, DABC algorithm generated slightly better results in terms of Avg, Min, Max values. In other words, the improvement over GA was 1091/1027=1.06. However, GA was more robust than DABC.

It is worth noting that for $\alpha = 0.0\ and\ \alpha = 1.0$, better results are achieved by the DABC algorithm. However, $\alpha = 0.0$ indicates only the total weighted tardiness and $\alpha = 1.0$ indicates only the makespan criterion. When analyzing these results, it can be seen that the DABC results was much more better when considering only the total weighted while DABC and GA performed almost similar when considering only makespan criterion.

TABLE XIV

COMPUTATIONAL RESULTS OF DABC AND GA FOR $\alpha = 0.9$

| $\alpha = 0,9$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | DABC_VNS | | | | GA_VNS | | | |
| Jobs | Ins | Min | Max | Avg | Std | Min | Max | Avg | Std |
| 100 | 1 | 248,00 | 371,00 | 289,00 | 52,21 | 255,00 | 501,00 | 330,00 | 103,85 |
| | 2 | 260,00 | 280,00 | 267,00 | 7,68 | 244,00 | 310,00 | 265,00 | 27,27 |
| | 3 | 248,00 | 302,00 | 265,00 | 22,69 | 251,00 | 322,00 | 269,00 | 30,07 |
| | 4 | 250,00 | 277,00 | 258,00 | 10,98 | 240,00 | 275,00 | 253,00 | 14,47 |
| | 5 | 237,00 | 263,00 | 244,00 | 11,54 | 262,00 | 273,00 | 265,00 | 5,00 |
| | 6 | 257,00 | 331,00 | 283,00 | 29,72 | 253,00 | 334,00 | 285,00 | 30,46 |
| | 7 | 253,00 | 296,00 | 266,00 | 17,74 | 264,00 | 315,00 | 289,00 | 21,32 |
| | 8 | 243,00 | 305,00 | 270,00 | 23,69 | 266,00 | 358,00 | 286,00 | 40,14 |
| | 9 | 257,00 | 310,00 | 271,00 | 22,33 | 252,00 | 304,00 | 273,00 | 28,12 |
| | 10 | 250,00 | 311,00 | 275,00 | 23,76 | 266,00 | 356,00 | 292,00 | 36,10 |

In the Table XIV, results are shown by taking the average of 5 replications of 100 jobs for each instance for $\alpha = 0.9$. According to the results, it can be seen that DABC algorithms yielded better results than GA for all statistics (Min, Max, Avg, Std).

## VI. CONCLUSION

In this paper, we presented a DABC and GA to solve a problem from the real-life. We developed DBAC and GA algorithms to assign the filling production orders to machine groups, then schedule them on each identical parallel machine groups. We also presented a unique solution representation inspired from general assignment problem. In addition, we developed a VNS local search to further improve the solution quality. We also devised benchmark instances to test the performance of the algorithms proposed. The computational results show that the DABC algorithm outperforms the GA on set of benchmark problems we generated.

As a future work, we will apply these algorithms to real-life data from DYO painting company in order to develop a decision support system for them.

## REFERENCES


[1] Du, J. and Leung, J.Y.T., (1990), Minimizing total tardiness on one machine is NP-hard, Mathematics of Operations Research, 15(3), 483-494.

[2] M. Pfund, J.W. Fowler, J.N.D. Gupta, (2004). A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling. Chinese Journal of Industrial Engineers 21, 230–241.

[3] E.L. Lawler, J.K. Lenstra, Kan Rinnoy, A.H.J., D.B. Shmoys, (1993). Sequencing and scheduling: algorithm and complexity. In: Handbooks in Operations Research and Management Science, Logistic of

Production and Inventory, vol. 4. North-Holland, Amsterdam, pp. 445–524.

[4] B. Chen, C.N. Potts, G.J. Woeginger, (1998). A review of machine scheduling: complexity, algorithms and applications. In: Du, D.-Z., Pardalos, P.M. (Eds.), Handbook of Combinatorial Optimization. Dordrecht, Netherlands, Kluwer, pp. 21–169.

[5] M. Haouari, and M. Jemmali. (2008). Tight Bounds for the Identical Parallel Machine Scheduling Problem: Part II. International Transactions in Operations Research 15 (1): 19–34.

[6] M. Skutella, and G. Woeginger. (2000). A PTAS for Minimizing the Total Weighted Completion Time on Identical Parallel Machines. Mathematics of Operations Research 25 (1): 63–75.

[7] McNaughton, R., "Scheduling with deadlines and loss functions. Management Science," 6(1):1–12., 1959.

[8] Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. SIAM Journal on Applied Mathematics, 17(2):416–429.

[9] J.G. Root Scheduling with deadlines and loss function on k paralel machines. Management Science 1965;11: 460-75.

[10] E.L. Lawler, A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics 1977;1:331-42.

[11] S.E, Elmaghraby, S.H. Park, Scheduling jobs on a number of identical machines. AIIE Transactions 1974;6:1-13.

[12] M.M. Dessouky, Scheduling identical jobs with unequal ready times on uniform parallel machines to minimize the maximum lateness. Computers and Industrial Engineering 1998; 34(4):793-806.

[13] L.J. Wilkerson, J.D. Irwin, An Improved Algorithm for Scheduling Independent Tasks. AIIE Transactions 1971; 3: 239-245.

[14] A. Dogramaci, J. Surkis, Evaluation of a Heuristic for Scheduling Independent Jobs on Parallel Identical Processors. Management Science 1979; 25: 1208-1216.

[15] J.C. Ho, Y.L. Chang, Heuristics for Minimizing Mean Tardiness for m Parallel Machines. Naval Research Logistics 1991; 38: 367-381.

[16] C.P. Koulamas, The Total Tardiness Problem: Review and Extensions. Operations Research 1994; 42: 1025-1041.

[17] C. Koulamas, Decomposition and Hybrid Simulated Annealing Heuristics for the Parallel-Machine Total Tardiness Problem. Naval Research Logistics 1997; 44: 109-125.

[18] J.C. Bean, Genetic Algorithms and Random Keys for Sequencing and Optimization. ORSA Journal on Computing 1994; 6: 154-160.

[19] E. Nowicki, C. Smutnicki, The Flow Shop with Parallel Machines: A Tabu Search Approach. EJOR 1998; 106: 226-253.

[20] M.W. Park, Y.D. Kim, Search Heuristics for a Parallel Machine Scheduling Problem with Ready Times and Due Dates. Computers and Industrial Engineering 1997; 33 (3-4): 793-796.

[21] D. Anghinolfi and M. Paolucci. Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. Computers and Operations Research, 34:3471/3490, 2007.

[22] U. Bilge, F. Kyra»c, F. Kurtulan, and M. Pekgun. A tabu search algorithm for parallel machine total tardiness problem. Computers and Operations Research, 31:397/414, 2004.

[23] M. Yagiura., T. Yamaguchi, T. Ibaraki,. (1998). A variable depth search algorithm with branching search for the generalized assignment problem. Optimization Methods and Software, 10, 419–441.

[24] M. Yagiura,. T. Ibaraki,. F. Glover, (2004). An ejection chain approach for the generalized assignment problem. INFORMS Journal of Computing, 16 (2), 133–151.

[25] M. Yagiura,. T. Ibaraki,. F. Glover (2006). 2006. A path relinking approach with ejection chains for the generalized assignment problem. European Journal of Operational Research, 169, 548–569.

[26] M.L. Fisher, R., Jaikumar, (1981). A generalized assignment heuristic for vehicle routing. Networks 11, 109–124.

[27] D. Karaboga, (2005) 'An idea based on honey bee swarm for numerical optimization', Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey.

[28] D. Karaboga, B. Basturk, (2007) 'A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm', Journal of Global Optimization pp. 39 459-471.

[29] D. Karaboga, B. Basturk,. (2008) 'On the performance of artificial bee colony (ABC) algorithm', Applied Soft Computing 8 pp. 687-697.

[30] D. Karaboga, (2009) 'A new design method based on artificial bee colony algorithm for digital IIR filters', Journal of The Franklin Institute 346 pp. 328-348.

[31] D. Karaboga, B. Akay, (2009) 'A comparative study of artificial bee colony algorithm', Applied Mathematics and Computation

[32] K. Karabulut, M. F. Tasgetiren, (2012), 'A discrete artificial bee colony algorithm for the travelling salesman problem with time windows', IEEE Congress on Evolutionary Computation , pp. 1-7.

[33] N. Mladenovic, P. Hansen, Variable neighborhood search, Computers and Operations Research 24 (1997) 1097-1100.

[34] M.F. Tasgetiren, Y-C Liang, M. Sevkli, Gencyilmaz G., A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem, European Journal of Operational Research 177 (2007) 1930-1947.

[35] R. Ruiz, and C. Maroto, (2005) A comprehensive review and evaluation of flowshop heuristics. Eur. J. Oper. Res., 165(2), 479–494.

[36] Q.K. Pan, M.F.Tasgetiren, Y.C. Liang, (2008) A Discrete Particle Swarm Optimization Algorithm for the No-Wait Flowshop Scheduling Problem with Makespan and Total Flowtime Criteria, Computers and Operations Research 35(9), 2807-2839.