KKU Engineering Journal

https://www.tci-thaijo.org/index.php/easr/index

Published by the Faculty of Engineering, Khon Kaen University, Thailand

A comparison of genetic algorithm and artificial bee colony approaches in solving blocking hybrid flowshop scheduling problems with sequence dependent setup/changeover times

Pongpan Nakkaew*[1], Nantachai Kantanantha[1] and Wuthichai Wongthatsanekorn [2]

[1]Industrial Engineering Department, Kasetsart University, Bangkok 10900, Thailand
[2]Industrial Engineering Department, Thammasat University, Patumtani 12120, Thailand

Received June 2015
Accepted September 2015

Abstract

In manufacturing processes where efficiency is crucial to remain competitive, the flowshop is a common configuration in which machines are arranged in series and products are produced individually in stages. In certain production processes, machines are frequently configured in a way that each production stage may contain multiple processing units in parallel or a hybrid configuration. Moreover, along with precedent conditions, sequence dependent setup times may exist. Finally, in the case where there is no buffer, a machine is said to be blocked if the next stage is occupied. In NP-Hard problems, referred to as Blocking Hybrid Flowshop Scheduling Problems with Sequence Dependent Setup/Changeover Times, it is usually not possible to find an exact best solution to satisfy optimization objectives. Thus, it is usually solved by approximate algorithms such as metaheuristics. In this paper, we comparatively investigate the effectiveness of two approaches: a genetic algorithm (GA) and an artificial bee colony (ABC) algorithm. Additionally, we applied an algorithm to improve the GA and ABC algorithms so that they can take advantage of parallel processing resources of modern multiple core processors, while eliminating the need for advance screening of the algorithm optimal parameters. These techniques were applied to solve three problems. In small-sized problems, GA outperformed ABC. For medium-sized problems, the two algorithms appeared to have similar performance. For large-sized problem, ABC performed better than GA because GA is held back by crossover operations. Furthermore, enhancements helped increase the performance of both GA and ABC algorithms.

Keywords: Genetic algorithm, Artificial bee colony, Sequence, Blocking, Flowshop

1. Introduction

Production planning, in general, may be described as finding favorable procedures or schedules to manufacture goods, given required quantities, available resources, conditions, constraints, and objectives. In most cases, scheduling problem may be categorized into four types depending on machine environments: flowshop, openshop, jobshop, and parallel-machine shop. In a flowshop, all jobs follow the same machine sequence and each job has exactly one operation on each machine. An openshop is similar to a flowshop, with the exception that the operation of a job may be performed in any order. In a jobshop, the process may follow different machine sequences and may use the same machine more than once. In parallel-machine shop, a shop consists of a number of identical machines and a job can be processed on any machine.

An n-job m-machine flowshop scheduling problem is an example of combinatorial optimization problems, dealing with locating an optimal object from a finite set of objects. The scheduling problem is to specify the resources and the times to process the jobs, depending to optimization objectives. The problem was originally brought to attentions of the researchers when [1] developed a simple decision rule to achieve the optimal scheduling of the items minimizing the makespan for two-stage problems. For three-stage problems, the solution can be obtained only for a restricted case. For such problem, there are generally m(n!) alternative sequences for the jobs. However, with the assumption that all machines process the jobs in the same order with no limitation of the buffers, the search space reduces to the n!.

When there are parallel machines allowing alternative routes, the assumption that all machines process the jobs in the same order is no longer valid. Consequently, the search space becomes larger. Because of the large search space, exhaustive search is usually not feasible. This type of problems are often addressed by heuristic methods and approximation algorithms. Despite the challenge, the problem continues to attract researchers due to its diverse applications and approches as reviewed by [2].

Other researchers, such as [3], [4], and [5] have attempted MILP-based approaches while [6] has employed an ant colony system approach. For TSP based approaches, a review is provided by [7].

*Corresponding author. Tel.: +6684 098 8838
Email address: pongpan@gmail.com
doi: 10.14456/kkuenj.2016.10

In this paper, we address three-stage blocking hybrid flowshop scheduling problem with sequence dependent setup/changeover times. The objective is to minimize the makespan, the time required to complete all the jobs. Since makespan is a crucial factor contributing to the operation cost, the ability to reduce makespan means a greater chance to achieve operational profitability.

1.1. Hybrid flowshop scheduling problem

In certain manufacturing processes, for instance, make and pack plants, the machines are usually configured in a way that each production stage may contain multiple processing units in parallel. Such configuration is known as flexible or hybrid flowshop.

In similar fashion to the mathematical formulation of a standard hybrid flowshop found in [8], with the objective to minimize the makespan, a model may be formulated as follows.

Let's first define the following parameters and indexes.

n	Number of jobs
m	Number of stages
j	Index for the jobs
i	Index for stages, {1, 2,..., m}
l	Index for machines in stage i
k	Index for the positions of the jobs in a particula r stage
$p_{j,i}$	Processing time of job j at stage i
C_{max}	Makespan
M	A large positive number

The following variables are defined.

$X_{j,i,k}$	Binary variable taking value 1 if job j occupies position k at stage i, and 0 otherwise
$Y_{i,k,l}$	Binary variable taking value 1 if the job in posi tion k of stage i is processed on machine l, and 0 otherwise
$S_{i,k}$	The starting time of the job in position k at stag e i
$F_{j,i}$	The starting time of job j at stage i

The model formulates the problem as follows.
Minimize

$$C_{max} = max\{C_j | j = 1, \dots, n\} \qquad (1)$$

where $$C_j = F_{j,m} + p_{j,m} \qquad (2)$$

Subject to:

$$\sum_{k=1}^{n} X_{j,i,k} = 1 \qquad \forall j, i \qquad (3)$$
$$\sum_{j=1}^{n} X_{j,i,k} = 1 \qquad \forall i, k \qquad (4)$$
$$\sum_{l=1}^{n} X_{i,k,l} = 1 \qquad \forall j, i \qquad (5)$$
$$F_{j,i+1} \geq F_{j,i} + p_{j,i} \qquad \forall j, i \leq m \qquad (6)$$
$$S_{i,k} \geq F_{j,i} - M(1 - X_{j,i,k}) \qquad \forall i, k, j \qquad (7)$$
$$S_{i,k} \geq F_{j,i} + M(1 - X_{j,i,k}) \qquad \forall i, k, j \qquad (8)$$
$$S_{i,k}, F_{j,i} \geq 0 \qquad (9)$$
$$X_{j,i,k}, Y_{i,k,l} \in \{0,1\} \qquad (10)$$

The constraints according to (3), (4), and (5) ensure that that every job occupies each stage exactly once without overlapping. (6) ensures that the operation of a job in the next stage only occurs after it has been completed in previous stage. (7) and (8) relate the jobs to the positions in the stages. Lastly, (9) and (10) define decision variables.

The standard model assumes that the processing times for each jobs are known in advance. The buffer capacities are also assumed to be unlimited.

In reality, in many processes, overall processing times of each stage are not known in advance. For example, in addition to the fixed processing times, there might be variable sequence dependent setup times. This increases the problem complexity because the total processing times for the jobs, in this case, cannot be determined in advance. Finally, when there are no buffer, the problem is even more difficult because the machines between succeeding stages are no longer independent. A machine is said to be blocked if the next stage to handle its output is being occupied. Such problem is referred here as Blocking Hybrid Flowshop Scheduling Problem with Sequence Dependent Setup/Changeover Times.

Because of the difficulty of the problem as pointed out by [9] and [10] and the lack of computational resources, publications addressing this problem only started to appear after the year 2000 when the advances in computer technologies make it more practical to address this problem heuristically without the need to formulate and solve a proper mathematical model.

1.2. The example problem

For investigation of the performance and behavior of a Genetic Algorithm and Artificial Bee Colony Algorithm, we implement the method to solve a three stage blocking hybrid flowshop problem. Each stage contains a number of identical processing units (PU) or processors as shown in Figure 1. These identical processors in the same stage perform the same operation. The processing units belonging to different stages, however, are usually different.

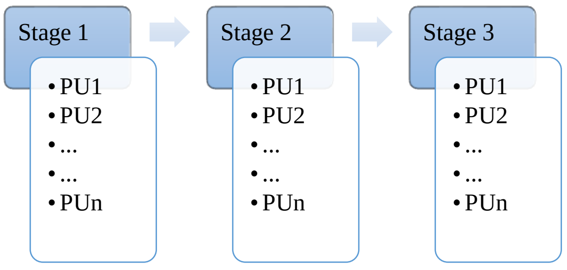

Figure 1 A typical setup of a hybrid flowshop problem

For each job to produce a product, the raw materials first enter stage 1 where they can be processed by any of the processing units, as long as the unit is free.

The minimum time the processing unit remains occupied can be obtained as the sum of the fixed processing time and the sequent dependent setup time. Upon completion after the time has elapsed, the product will be transferred to the subsequent stage as soon as there is an available free processing unit downstream. In the case that none of the processors in the subsequent stage is free, the product will continue to keep the current processor occupied after the task has finished. In other words, the current processing unit will continue to be unavailable or blocked, waiting for an

Table 1 The number of processors in each stage for each case.

cases	Number of PU		
	1st stage	2nd stage	3rd stage
small	3	3	3
medium	5	5	5
large	25	25	25

available processing unit downstream because there is no buffer between stages. Nevertheless, the buffers preceding the first stage and following the last stage are assumed to be unlimited.

The small, medium and large cases of the three-stage problem used for the numerical analysis are generated. In each case, the number of the processors in all stages are the same. As shown in Table 1, the numbers processing units of all stages are varied for the three case as 3, 5, and 25, respectively.

All the three cases consist of 10 product types in various quantities according to Table 2. To produce each product of any type, exactly one job consisting to three tasks is required. Consequently, the required quantities are equal to the number of jobs. In total, there are 10, 100, 500 jobs to be assigned for the small, medium, and large case, respectively. The required quantity for each product type for each case are as follows.

Table 2 Required quantities for each product type for each case.

product type	small case	medium case	large case
1	1	12	60
2	1	11	55
3	1	7	35
4	1	14	70
5	1	5	25
6	1	8	40
7	1	12	60
8	1	13	65
9	1	10	50
10	1	8	40
Total number of jobs	10	100	500

The processing times in hours for each product type at each stage for the three cases are the same as shown in Table 3.

Table 3 Production time in hours for each product type for each stage.

Product Type	Stage 1	Stage 2	Stage 3
1	0.1930	0.0915	0.6423
2	0.3416	0.6146	0.2213
3	0.9329	0.0110	0.8371
4	0.3907	0.5733	0.9711
5	0.2732	0.7897	0.8464
6	0.1519	0.2354	0.5060
7	0.3971	0.4480	0.2789
8	0.3747	0.5694	0.7466
9	0.1311	0.0614	0.2369
10	0.4350	0.4963	0.9573

The setup times in hours for switching product types at stage 1, 2, and 3 are as follows.

$$S_1 = \begin{array}{cccccccccc}
0 & 0.5747 & 0.1170 & 0.5154 & 0.3242 & 0.9969 & 0.8266 & 0.5038 & 0.7202 & 0.3600 \\
0.7167 & 0 & 0.8147 & 0.6575 & 0.3017 & 0.5535 & 0.3945 & 0.6128 & 0.3469 & 0.4542 \\
0.2834 & 0.4564 & 0 & 0.9509 & 0.0117 & 0.5155 & 0.6135 & 0.8194 & 0.5170 & 0.3864 \\
0.8962 & 0.7138 & 0.2462 & 0 & 0.5399 & 0.3307 & 0.8186 & 0.5319 & 0.5567 & 0.7756 \\
0.8266 & 0.8844 & 0.3427 & 0.4001 & 0 & 0.4300 & 0.8862 & 0.2021 & 0.1565 & 0.7343 \\
0.3900 & 0.7209 & 0.3757 & 0.8319 & 0.1465 & 0 & 0.9311 & 0.4539 & 0.5621 & 0.4303 \\
0.4979 & 0.0186 & 0.5466 & 0.1343 & 0.6311 & 0.0710 & 0 & 0.4279 & 0.6948 & 0.6938 \\
0.6948 & 0.6748 & 0.5619 & 0.0605 & 0.8593 & 0.8877 & 0.2586 & 0 & 0.4265 & 0.9452 \\
0.8344 & 0.4385 & 0.3958 & 0.0842 & 0.9742 & 0.0646 & 0.8979 & 0.6201 & 0 & 0.7842 \\
0.6096 & 0.4378 & 0.3981 & 0.1639 & 0.5708 & 0.4362 & 0.5934 & 0.6954 & 0.7314 & 0
\end{array}$$

$$S_2 = \begin{array}{cccccccccc}
0 & 0.6690 & 0.0196 & 0.1432 & 0.5078 & 0.9419 & 0.8669 & 0.1403 & 0.2436 & 0.1749 \\
0.3899 & 0 & 0.4352 & 0.5594 & 0.5856 & 0.6559 & 0.4068 & 0.2601 & 0.7851 & 0.1386 \\
0.5909 & 0.2180 & 0 & 0.0046 & 0.7629 & 0.4519 & 0.1126 & 0.0868 & 0.0741 & 0.5989 \\
0.4594 & 0.5716 & 0.6174 & 0 & 0.0830 & 0.8397 & 0.4438 & 0.4294 & 0.3939 & 0.9011 \\
0.0503 & 0.1222 & 0.5201 & 0.8487 & 0 & 0.5326 & 0.3002 & 0.2573 & 0.0034 & 0.9394 \\
0.2287 & 0.6712 & 0.8639 & 0.9168 & 0.5170 & 0 & 0.4014 & 0.2976 & 0.2207 & 0.2212 \\
0.8342 & 0.5996 & 0.0977 & 0.9870 & 0.1710 & 0.6801 & 0 & 0.4249 & 0.0013 & 0.4827 \\
0.0156 & 0.0560 & 0.9081 & 0.5051 & 0.9386 & 0.3672 & 0.4036 & 0 & 0.1892 & 0.3760 \\
0.8637 & 0.0563 & 0.1080 & 0.2714 & 0.5905 & 0.2393 & 0.3902 & 0.4951 & 0 & 0.5238 \\
0.0781 & 0.1525 & 0.5170 & 0.1008 & 0.4406 & 0.5789 & 0.3604 & 0.7064 & 0.2681 & 0
\end{array}$$

$$S_3 = \begin{array}{cccccccccc}
0 & 0.7060 & 0.3180 & 0.5324 & 0.2748 & 0.1888 & 0.7624 & 0.6723 & 0.0249 & 0.7269 \\
0.4363 & 0 & 0.6086 & 0.7165 & 0.2415 & 0.0012 & 0.5761 & 0.4315 & 0.6714 & 0.3738 \\
0.1739 & 0.5523 & 0 & 0.1793 & 0.2431 & 0.3164 & 0.7477 & 0.6944 & 0.8372 & 0.5816 \\
0.0261 & 0.2181 & 0.9091 & 0 & 0.1542 & 0.6996 & 0.6455 & 0.2568 & 0.9715 & 0.1161 \\
0.9547 & 0.7724 & 0.5916 & 0.1877 & 0 & 0.6253 & 0.1232 & 0.0098 & 0.0569 & 0.0577 \\
0.4306 & 0.2280 & 0.3326 & 0.3219 & 0.9357 & 0 & 0.5044 & 0.5323 & 0.4503 & 0.9798 \\
0.9616 & 0.3709 & 0.8531 & 0.4039 & 0.8187 & 0.4390 & 0 & 0.2794 & 0.5825 & 0.2848 \\
0.7624 & 0.8909 & 0.4424 & 0.5486 & 0.7283 & 0.2874 & 0.0921 & 0 & 0.6866 & 0.5950 \\
0.0073 & 0.8564 & 0.9044 & 0.0487 & 0.1758 & 0.5017 & 0.1478 & 0.9064 & 0 & 0.9622 \\
0.6800 & 0.4024 & 0.0332 & 0.5527 & 0.3604 & 0.7615 & 0.1982 & 0.3927 & 0.6500 & 0
\end{array}$$

The S_1, S_2, and S_3 are for the stage 1, 2, and 3, respectively. The row position represents the product that has been or are currently produced. The column position represents the next product to produce. As an example, if you want to know the setup time for switching from product type 3 to product type 2 for stage 1, you look at S1(3, 2), which is 0.4564 hours.

2. Methodology

Evolutionary computation is a subfield of artificial intelligence that involves continuous optimization and combinatorial optimization problems. Two examples of evolutionary computation presented in this paper are Genetic Algorithm (GA) and Artificial Bee Colony (ABC).

2.1 Genetic algorithm

GA is a search heuristic that mimics the process of natural selection according to Darwinian principles. The use of survival of the fittest principles for automated problem solving originated in the 1950s. The examples of the GA-based approaches and applications include [11], [12], and [13].

Typically, a GA proceeds as follows.

1. Generate the initial population of chromosomes.
2. Randomly select a chromosome or chromosomes from population to induce either mutation and/or crossover to create a number of offspring.
3. Compare the offspring. The one with the best result is selected to replace a chromosome among the population.
4. Keep repeating step 2 and step 3 until a stopping criterion is met.

2.2 Artificial bee colony algorithm

Let's consider the behavior of honey bees in locating food sources. [14] found that the colony routinely foraged several kilometers from its nest, frequently adjusted its distribution of foragers on its patches, and worked relatively few patches each day. This foraging pattern hints that the foraging strategy of a honey bee colony involves surveying the food source patches within a vast area around its nest, pooling the exploration results of its many foragers, and using this information to focus the forager labors more on a few promising patches within its foraging area.

Such behavior had inspired researchers into developing the Artificial Bee Colony Optimization (ABC) algorithm. [15], for instance, had conducted empirical study of such approach while [16] developed a novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan criterion. [17] is another example where a modified ABC algorithm was used for parameter optimization.

In ABC algorithm, the actual position of a food source corresponds to a possible solution to the optimization problem and the amount of nectar found in a food source represents the quality of the associated solution.

According to the model, the colony consists of three groups of bees: scout bees, employed bees, and onlooker bees. It is assumed that there is only one artificial employed bee tending to each food source. In other words, the number of employed bees in the colony is the same as the number of food sources being explored. The number of employed bees can be chosen freely and will dictate the number of registered food sources. The number of scout and onlooker bee can be chosen without restrictions.

First the scout bees will explore and register the potential food sources. Once scout bees have been successful in identifying food sources, the employed bees will take over and proceed to evaluate the quality of the sources.

The employed bees usually stray from their memorized positions and discover new neighbor food source locations. Provided that the nectar amount of the new one is higher than that of the previous source, the bees memorize the new source positions and forget the old ones. Otherwise, they will keep the positions of the one in their memories. Back in the hive, the employed bees that complete the search will share the position information of the sources with the onlookers that are observing the employed bees.

Based on the behaviors of the employed bees, the onlooker bees will decide which food sources they will explore. The onlooker bees, however, will explore the target that is slightly different than the area previously visited. That is finding a feasible solution in the vicinity of an existing solution.

Each onlooker evaluates the nectar information taken from all employed bees and then chooses a food source depending on the nectar amounts of sources. In similar manner to the case of the employed bee, it produces a modification on the original source position and checks its nectar amount. Given that its nectar is higher than that of the previous one, the bee registered the new position.

In deciding which food sources to explore, these onlooker bees will take into account of the information from the scout bees. The realization of this behavior in a scheduling problem with the objective is to minimize makespan is as follows. Naturally, given that m is the makespan, we define q, the food quality as 1/m. p_i, the probability that the area i is selected, is then defined as:

$$p_i = \frac{q_i}{\sum_{i=1}^{n} q_i} \tag{11}$$

where $$q_i = \frac{1}{m_i} \tag{12}$$

After the flight, if the quality of the source is better than the quality of previously found sources, the new one will be registered and one of the existing food sources is dismissed. Meanwhile, the employed bee whose source is abandoned will be relocated to work on a new source. Figure 2 illustrates such behavior of the bees which is realized as an artificial bee colony optimization routine.

2.3 Solution representation

For flexible flowshop, we must first consider the order of the jobs. Then, additionally we must consider which of the processer in each stage will handle a particular job. Thus for 3 stage flowshop, each job will compose of three tasks, each of which for one of the processing unit for a particular stage. Let's consider the following solution to produce just three products.

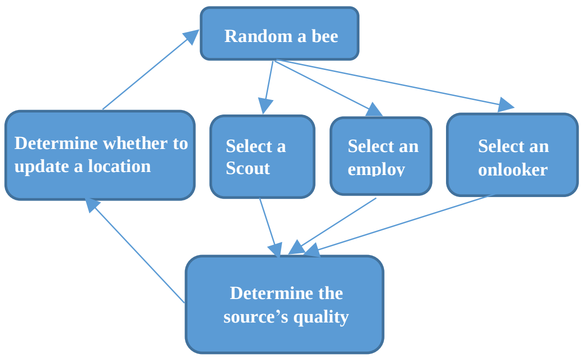
Random a bee
Determine whether to update a location
Select a Scout
Select an employ
Select an onlooker
Determine the source's quality

Figure 2 Artificial bee colony algorithm routine

$$C = \begin{matrix} 1 & 2 & 1 & 4 & 3 \\ 2 & 1 & 2 & 3 & 2 \\ 3 & 3 & 2 & 2 & 2 \end{matrix}$$

In this example, the row order represents the order in which the jobs enter the machines. The first column represent job indexes. The second column represents the product types or formulas. Third, forth, and fifth column represents the machine assignments for a given job for stage 1, stage 2, and stage 3, in that order.

Given this solution C, product 2 will be processed first by CU# 1, CU# 4, and CU# 3 of the stage 1, stage 2, and stage 3, respectively. Product 1 and 3 are then processed sequentially in similar manner.

Once, the processing steps are defined, what left to be determined are timings in order to obtain a solution to the scheduling problem. This can be constructed heuristically following a prescribed procedure as follows. The job being processed, once completed, will be transferred downstream as soon as the designated resource in the next stage becomes available. In the case of the job waiting in the queue, however, they must enter the first stage in accordance with the order specified by the solution.

2.4 The algorithm enhancements

Typically, prior to performing optimization process using algorithms such as the Genetic Algorithm or Artificial Bee Colony optimization, suitable parameters must be evaluated. Without the assumption that these parameters are independent, the process of finding the optimal parameters, by itself, is an combinational optimization problem. To eliminated the need for optimization of the parameters, the following procedure, inspired by League Championship Algorithm (LCA), inspired by [18], is introduced.

1. A number of optimization routines are executed in parallel with different sets of random parameters and population pools for a certain period. In this paper, the number of parallel jobs chosen is four to fully utilize a quad core CPU.

2. At the end of the period, the performance of different pools are evaluated. The population and the parameters of the best performer are kept the same. The parameters of the weaker performers are adjusted stochastically in such a way that they are closer to the parameters of the best performer. This is inspired by the Firefly algorithm. Some population from the best performance group are also introduced into the weaker groups.

3. Repeat step 1 and 2 until a stopping criterion such as time limit is met.

3. Numerical results and discussion

The results presented in this literature are obtained using an AMD Phenom II X4 965 3.4 GHz Processor based PC with 8 GB 1333MHz DDR3 SDRAM. The script is implemented using Matlab.

Given that the control parameters are the population size, number of offspring produced through mutation, and the number of children produced by crossover processes, we first determine the suitable ranges of these parameters.

The suitable value of a parameter is determined by varying its value while keeping other parameters constant to identify the value that yield best result. For example, the population size is determined by varying the population size while keeping the number of offspring from mutation, the number of children from crossover processes, and other parameters the same. An example this approach is shown as Figure 3.

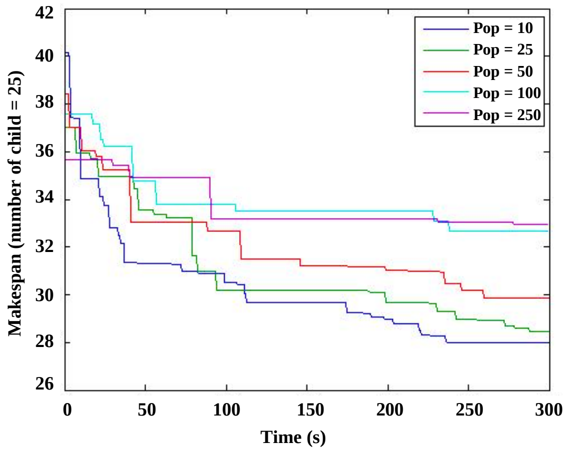

Figure 3 Illustrate the process to determine a parameter

In the end, we obtain the suitable parameters for Genetic algorithm as follows. Population size, the number children from the crossover process, and the number of offspring resulting from mutation are 10, 10, and 100, respectively.

The parameters for the Artificial Bee Colony Algorithm are also obtained in the same manner. With such procedure, the three parameters are obtained as follows. The number of employed bees, scout bees, and onlooker bees are 10, 25, and 50, respectively.

To evaluate the performance of a Genetic algorithm and an Artificial Bee Colony Algorithm for comparison, we conducted a number of experiments with different time settings. The runtimes are varied between 10 to 1000 seconds. For each case, 30 trials were conducted. The result are summarized in Table 4, 5, and 6. In addition to the means, standard deviations (SD) are also provided as an information concerning the distribution and the consistency of the data.

For discussion, the results are shown in Figure 4 - 6. In all cases, the results are relatively consistent as the SD values are within a few percent of the results.

For the small case, ABC appeared to be trapped by local optimums. GA, however, seemed to be able to escape local optimums thanks to the crossover operations. The benefit of enhancements is obvious, especially in the case of GA.

For the medium case, even though the ABC seems to outperform at the beginning, in the end similar results are achieved by GA and ABC. Both Enh. GA and Enh. ABC perform considerably better.

For the large case, the ABC algorithm outperform the GA approach due to the fact that ABC demands less computing resources. Again, the benefit of the enhancements are obvious.

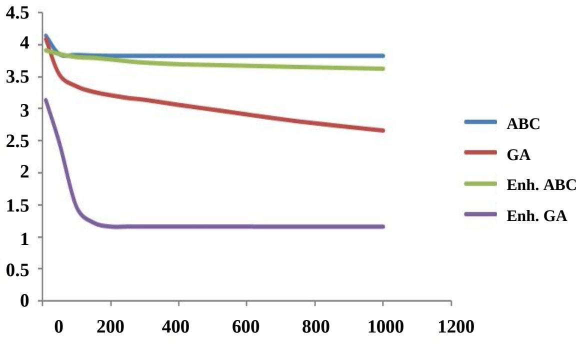

Figure 4 Comparison of the results between the different approaches for the small case

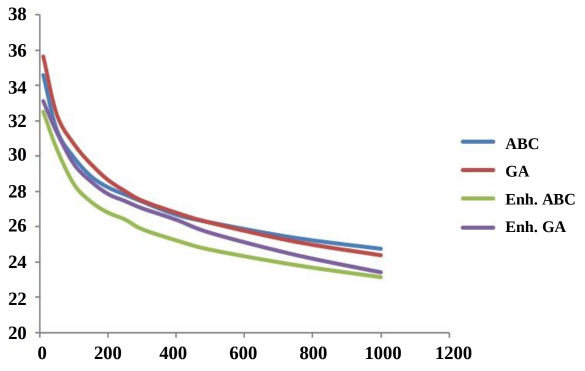

Figure 5 Comparison of the results between the different approaches for the medium case

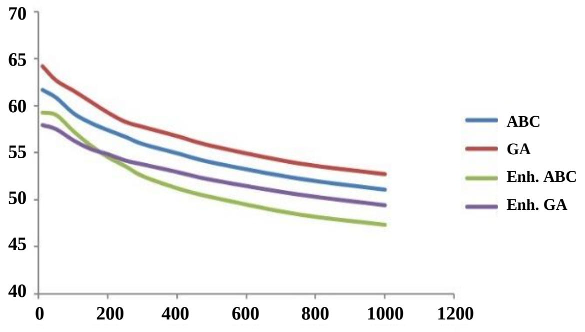

Figure 6 Comparison of the results between the different approaches for the large case

Table 4 The numerical results of 30 trials for the small case

Algorithms	**ABC**		**GA**		**Enh. ABC**		**Enh. GA**	
time (sec)	mean	SD	mean	SD	mean	SD	mean	SD
10	4.14389	0.163403	4.08601	0.222627	3.906367	0.13157	3.129203	0.315978
50	3.849547	0.151429	3.53205	0.234919	3.85119	0.154398	2.45707	0.515802
100	3.838013	0.16156	3.350547	0.322659	3.802787	0.135232	1.464347	0.33819
150	3.827827	0.166299	3.263283	0.377353	3.7907	0.118028	1.218723	0.212358
200	3.823367	0.17445	3.2118	0.369527	3.766677	0.121614	1.15751	0.060316
250	3.823367	0.17445	3.169167	0.385666	3.738207	0.127581	1.15751	0.060316
300	3.823367	0.17445	3.140503	0.383236	3.71711	0.133639	1.15751	0.060316
400	3.822187	0.174585	3.06159	0.361665	3.692243	0.138983	1.15751	0.060316
500	3.822187	0.174585	2.98824	0.384473	3.679937	0.136181	1.15751	0.060316
750	3.822187	0.174585	2.805243	0.395647	3.649587	0.125987	1.155893	0.060103
1000	3.822187	0.174585	2.661827	0.367806	3.622253	0.127456	1.155893	0.060103

Table 5 The numerical results of 30 trials for the medium case

Algorithms	**ABC**		**GA**		**Enh. ABC**		**Enh. GA**	
time (sec)	mean	SD	mean	SD	mean	SD	mean	SD
10	34.57868	1.024731	35.66021	1.51599	32.5194	0.885741	33.129	0.801592
50	31.4616	1.015471	32.34472	1.261498	30.38272	0.763346	31.33969	0.749133
100	29.91343	0.99702	30.68524	1.041599	28.40118	0.857697	29.54316	0.854553
150	28.84818	0.913967	29.54219	0.986282	27.40336	0.933412	28.55985	0.828626
200	28.2207	0.932094	28.64059	0.889142	26.78798	0.906867	27.84742	0.858498
250	27.81355	0.930573	28.00692	0.770565	26.39883	0.836791	27.44687	0.819769
300	27.41356	0.859827	27.46584	0.87944	25.84964	0.755968	27.03758	0.763417
400	26.66976	0.910056	26.78215	0.855234	25.21382	0.814836	26.38419	0.821173
500	26.23299	0.807977	26.21324	0.904327	24.68466	0.733921	25.63889	0.947584
750	25.35632	0.695284	25.13215	0.954793	23.81293	0.655	24.38433	0.896579
1000	24.74278	0.819755	24.36496	0.950001	23.12639	0.643166	23.39898	0.845096

Table 6 The numerical results of 30 trials for the large case

Algorithms	**ABC**		**GA**		**Enh. ABC**		**Enh. GA**	
time (sec)	mean	SD	mean	SD	mean	SD	mean	SD
10	61.65559	1.231111	64.15371	2.231368	59.24596	1.042457	57.96896	0.362942
50	60.81363	1.154548	62.62462	1.862528	58.97562	0.954264	57.52176	0.541232
100	59.16124	1.313446	61.54429	2.141688	57.24986	1.13295	56.33678	0.718034
150	58.13117	1.315264	60.37343	1.999046	55.73572	1.39643	55.41366	1.143131
200	57.36039	1.44558	59.20591	2.239597	54.49624	1.549506	54.86504	1.129507
250	56.65401	1.464554	58.24348	2.087927	53.5444	1.498294	54.19965	1.228245
300	55.89002	1.565818	57.70891	2.032492	52.49514	1.525258	53.79082	1.321017
400	54.90753	1.533614	56.72622	1.780611	51.20406	1.776619	52.97451	1.475872
500	53.93014	1.617016	55.68696	1.543101	50.25305	1.748974	52.14233	1.617723
750	52.22034	1.331117	53.84463	1.639141	48.43515	1.565003	50.59592	1.586276
1000	51.03764	1.466034	52.69946	1.493084	47.3278	1.544683	49.45846	1.547568

4. Conclusion

Recent evolutionary computations, including GA and ABC, are mostly swam-based. While GA introduces the concept of population, ABC uses the number of employed bees dictating the number of food sources.

ABC employs scout bee perform random search. In the same manner, random search can be adopted for GA through insertion of random population.

The differences between GA and ABC largely come down to the behaviors the algorithms treat the population. GA treats data equally by randomly select data. ABC, on the other hand, may prioritize certain data with the introduction of onlooker bees. In short term, this give the ABC algorithm the advantage. However, once most data are optimized, the quality gaps of the data are expected to be reduced. Eventually, this advantage yield diminishing return.

The GA algorithm, on the other hand, uniquely introduces crossover operations that help to locate new solution while escaping local optimum. But for certain problems, such operations can be more resource intensive as the problem grows.

Thus, for small problem, GA may be more suitable. For medium size problem, the two algorithms appear to yield similar performances as long as proper parameters are used. For large problem, the GA algorithm is holding back by the crossover operations. Consequently, in such case, ABC algorithm generally performs better.

For regular GA and ABC Algorithm, the optimal parameters are usually screened in advance. In this paper, along with the comparison of GA and ABC Algorithm, we also present an approach to allow automatic adaptation of the key parameters. Consequently, the need to determine optimization parameter in advance is no longer crucial. Furthermore, the enhancement takes advantage to the multiple processing cores of the CPU. As the results, Enh. GA and Enh. ABC are introduced. The numerical results clearly demonstrate the benefit of such enhancement in comparison with the typical GA and ABC Algorithm.

5. Acknowledgements

This research was supported by Thailand Research Fund, and National Research University Project of Thailand, Office of the Higher Education Commission.

6. References

[1] Johnson SM. Optimal two- and three-stage production schedules with setup times included. Santa Monica: The RAND Corporation; 1953.

[2] Hejazi SR, Saghafian S. Flowshop-scheduling problems with makespan criterion: a review. Int J Prod Res. 2005;43(14):2895-929.

[3] Baumann P, Trautmann N. An MILP approach to short-term scheduling of an industrial make-and-pack production facility with batch splitting and quality release times. 2010 IEEE International Conference on Industrial Engineering and Engineering Management; 2010 Dec 7-10; Macao, China. USA: IEEE; 2010. p. 1230-4.

[4] Baumann P, Trautmann N. A continuous-time MILP to compute schedules with minimum changeover times for a make-and-pack production. 21st European Symposium on Computer Aided Process Engineering – ESCAPE 21; 2011 May 29-1 June; Chalkidiki, Greece. Amsterdam: Elsevier Publishing; 2011. p. 1060-4.

[5] Mendez CA, Cerda J. An MILP-based approach to the short-term scheduling of make-and-pack continuous production plants. OR Spectrum. 2002;24:403-29.

[6] Ying KC, Lin SW. Multiprocessor task scheduling in multistage hybrid flow-shops: an ant colony system approach. Int J Prod Res. 2006;44(16):3161-77.

[7] Bagchi TP, Gupta JND, Sriskandarajah C. A review of TSP based approaches for flowshop scheduling. Eur J Oper Res. 2006;169(3):816-54.

[8] Najafi E, Naderi B, Sadeghi H, Yazdani M. A mathematical model and a solution method for hybrid flow shop scheduling. J Optim Ind Eng. 2012;10:65-72.

[9] Xie J, Wang X. Complexity and algorithms for two-stage flexible flow shop scheduling with availability constraints. Comput Math Appl. 2005;50(10-12):1629-38.

[10] Honkomp SJ, Lombardo S, Rosen O, Pekny JF. The curse of reality - why process scheduling optimization problems are difficult in practice. Comput Chem Eng. 2000;24(2-7):323-8.

[11] Larranaga P, Kuijpers CMH, Murga RH, Inza I, Dizdarevic S. Genetic algorithms for the traveling salesman problem: a review of representations and operators. Artif Intell Rev. 1999;13:129-70.

[12] Mirabi M. A hybrid genetic algorithm for the sequence dependent flow-shop scheduling problem. World Acad Sci Eng Tech. 2011;5(7):1364-70.

[13] Zandieh M, Rashidi E. An effective hybrid genetic algorithm for hybrid flow shops with sequence dependent setup times and processor blocking. J Ind Eng. 2009;4:51-8.

[14] Visscher PK, Seeley TD. Foraging strategy of honey bee colonies in a temperate deciduous forest. Ecol. 1982;63(6):1790-801.

[15] Nikolic´ M, Teodorovic D. Empirical study of the Bee Colony Optimization (BCO) algorithm. Expert Syst Appl. 2013;40(11):4609-20.

[16] Pan QK, Wang L, Li JQ, Duan JH. A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimization. Omega. 2014;45:42-56.

[17] Akay B, Karaboga D. A modified Artificial bee colony algorithm for real-parameter optimization. Inform Sci. 2012;192:120-42.

[18] Kashan AH. An efficient algorithm for constrained global optimization and application to mechanical engineering design: League championship algorithm (LCA). Comput Aided Des. 2011;43(12):1769-92.