An Improved Discrete Artificial Bee Colony Algorithm for Hybrid Flow Shop Problems

Zhe Cui and Xingsheng Gu*

Key Laboratory of Advanced Control and Optimization for Chemical Process of Ministry of Education, East China University of Science and Technology, Shanghai 200237, China
xsgu@ecust.edu.cn

Abstract. Being a typical NP-hard combinatorial optimization problem, the hybrid flow shop (HFS) problem widely exists in manufacturing systems. In this paper, we firstly establish the model of the HFS problem by employing the vector representation. Then an improved discrete artificial bee colony (IDABC) algorithm is proposed for this problem to minimize the makespan. In the IDABC algorithm, a novel differential evolution and a modified variable neighborhood search are studied to generating new solutions for the employed and onlooker bees. The destruction and construction procedures are utilized to obtain solutions for the scout bees. The simulation results clearly imply that the proposed IDABC algorithm is highly effective and efficient as compared to six state-of-the-art algorithms on the same benchmark instances.

Keywords: hybrid flow shop problem, mathematical model, artificial bee colony, differential evolution, scheduling.

1 Introduction

Production scheduling is a decision-making process that plays a crucial role in manufacturing and service industries [1]. As industries are facing increasingly competitive situations, the classical flow shop model is not applicable to some practical industry processes. As a result, the hybrid flow shop (HFS) problem in which a combination of flow shop and parallel machines operate together arises. Although the HFS problem, also called multi-processor or flexible flow shop, widely exists in real manufacturing environments, e.g., in chemical, oil, food, tobacco, textile, paper, and pharmaceutical industries, there is no effective method to solve this problem.

As to the computational complexity, since even the two-stage HFS problem is strongly NP-hard [2] by minimizing the maximum completion time (makespan), the multi-stage HFS problem is at least that difficult. Despite of the intractability, the HFS problem has great significance in both engineering and theoretical fields. Thus, it is meaningful to develop effective and efficient approaches for such the problem considered.

* Corresponding author.

K. Li et al. (Eds.): ICSEE 2013, CCIS 355, pp. 294–302, 2013.

Compared with a large number of literatures on the classic flow shop scheduling problem, the HFS problem has not been well studied. Santos et al. [3] presented a global lower bound for makespan minimization that has been used to analyze the performance of other algorithms. Neron et al. [4] used the satisfiability tests and time-bound adjustments based on the energetic reasoning and global operations to enhance the efficiency of another kind of B&B method proposed in Carlier and Neron [5]. With advanced statistical tools, Ruiz et al. [6] tested several heuristics in the realistic HFS problem and suggested that the modified NEH heuristic [7] outperformed the other dispatching rules. The genetic algorithm (GA) was applied by researchers to solve the HFS problem under the criterion of makespan minimization [8]. On the basis of vertebrate immune system, Engin and Doyen [9] proposed the artificial immune system (AIS) technique that incorporated the clonal selection principle and affinity maturation mechanism. Inspired by the natural mechanism of the ant colony, Alaykyran et al. [10] introduced an improved ant colony optimization (ACO) algorithm. Niu et al. [11] presented a quantum-inspired immune algorithm (QIA) for the HFS problem to minimize makespan. Liao et al. [12] developed a particle swarm optimization (PSO) algorithm. This algorithm hybridized the PSO and bottleneck heuristic to fully exploit the bottleneck stage, and further introduced simulated annealing to help escape from local optima. In addition, a local search is embedded to further improve its performance.

The artificial bee colony (ABC) algorithm, simulating the intelligent foraging behaviors of honey bee colonies, is one of the latest population-based evolutionary meta-heuristics [13]. Basturk and Karaboga suggested that the ABC algorithm has a better performance than the other population-based algorithms for solving continuous problems [14], [15]. Nevertheless, on account of its continuous nature, the studies on the ABC algorithm for combinatorial optimization problems is very limited.

As we know, there is no published work using the ABC-based algorithm for the HFS problem. In this paper, we firstly establish the model of the HFS problem by employing the vector representation, and then an improved discrete artificial bee colony (IDABC) algorithm is proposed for the problem to minimize the makespan. The rest of the paper is organized as follows. In Section 2, the model of the HFS problem is formulated. Section 3 presents the details of the proposed IDABC algorithm. The simulation results are provided in Section 4. Finally, conclusions are drawn in Section 5.

2 Problem Statement

2.1 Description of the Problem

The HFS problem can be described as follows. There are n jobs J={1,2,...,i,...,n-1,n} that have to be performed on s stages S={1,2,...,j,...,s-1,s}, and each stage s has m_s identical machines. These identical machines are continuously available from time zero and have the same effect. At least one stage j must has more than one machine. Every job has to visit all of the stages in the same order

string from stage 1 through stage s and is processed by exactly one machine at every stage. A machine can process at most one job at a time and a job can be processed by at most one machine at a time. The processing time $p_{i,j}$ is given for each job at each stage. The scheduling problem is to choose a machine at each stage for each job and determine the sequence of jobs on each machine so as to minimize the makespan.

2.2 Mathematical Model

As we know, on the research of the HFS problem, there are two formats to represent a solution, namely the matrix representation and the vector representation. In this paper, we employ the vector representation [16], which considers the sequence of jobs only at the stage one. This subset sequence should contain the collection of all potentially good solutions for the problem and is a one-to-one correspondence. Most importantly, it is very convenient to design and operate by using this format. A subset sequence is decoded to a complete schedule by employing a generalization of the List Scheduling (LS) algorithm to incorporate the jobs at other stages [17], [18]. For scheduling jobs at each stage, the LS algorithm is based on the first-come-first-service rule, in which the jobs with the shortest completion time from the previous stage should be scheduled as early as possible. It could result in a non-permutation schedule, that is, the sequence of jobs at each stage may be different. The model of the HFS problem can be formulated as follows in terms of this representation:

Minimize: $$C_{\max}(\pi_1) = \max_{i=1,2,...n}\{C_{\pi_s(i),s}\}$$

Subject to:

$$\begin{cases} C_{\pi_1(i),1} = p_{\pi_1(i),1} \\ IM_{i,1} = C_{\pi_1(i),1} \end{cases} i = 1, 2, ...m_1$$

$$\begin{cases} C_{\pi_1(i),1} = \min\limits_{k=1,2,...m_1}\{IM_{k,1}\} + p_{\pi_1(i),1} \\ NM_1 = \arg\min\limits_{k=1,2,...m_1}\{IM_{k,1}\} \\ IM_{NM_1,1} = C_{\pi_1(i),1} \end{cases} i = m_1 + 1, m_1 + 2, ...n$$

$$\pi_j(i) = g(C_{\pi_{j-1}(i),j-1})\ i = 1, 2, ...n;\ j = 2, 3, ...s$$

$$\begin{cases} C_{\pi_j(i),j} = C_{\pi_j(i),j-1} + p_{\pi_j(i),j} \\ IM_{i,j} = C_{\pi_j(i),j} \end{cases} i = 1, 2, ...m_1;\ j = 2, 3, ...s$$

$$\begin{cases} C_{\pi_j(i),j} = \max\{C_{\pi_j(i),j-1}, \min\limits_{k=1,2,...m_j}\{IM_{k,j}\}\} + p_{\pi_j(i),j} \\ NM_j = \arg\min\limits_{k=1,2,...m_j}\{IM_{k,j}\} \\ IM_{NM_j,j} = C_{\pi_j(i),j} \end{cases} i = m_j + 1, m_j + 2, ...n;\ j = 2, 3, ...s$$

where π_j is the job permutation at the stage j; $\pi_k(i)$ is the ith job in the π_k; $C_{\pi_k(i),j}$ is the completion time of job $\pi_k(i)$ at the stage j; $IM_{i,j}$ represents the idle moment of machine i at the stage j; NM_j denotes the serial number of earliest available machine at the moment at the stage j; the function

$S_1(i) = g(S_2(i))$ (i=1,2,...n) means that $S_1(i)$ is the permutation of (1,2,...n) based on the ascending order of $S_2(i)$ (i=1,2,...n); and $\arg\min_k\{IM_k\}$ stands for the argument of the minimum, i.e. the set of points of the given argument for which the given function attains its minimum value.

In the above recursive equations, we firstly calculate the completion time of the jobs at the stage one, then that of the stage two, until the last stage.

3 The Improved Discrete Artificial Bee Colony (IDABC) Algorithm for the HFS Problem

3.1 Individual Representation and Initialization

Owing to the continuous nature of the ABC algorithm, it can not be directly used for the HFS problem. So it is important to find a suitable mapping which can conveniently convert a harmony to a solution. The model of the HFS problem is formulated by using the vector representation in last section. As a result, we adopt this representation in the proposed IDABC algorithm. The individual in the IDABC algorithm is represented by a permutation of jobs at the stage one $= \{\pi(1), \pi(2), ..., \pi(n)\}$.

To guarantee the initial population with a certain quality and diversity, it is constructed randomly except that one is established by the aforementioned NEH heuristic [7]. According to [6], the NEH heuristic, a typical constructive method for the permutation flow shop scheduling problem, is also very robust and well performing for the HFS problem.

3.2 Employed Bee Phase

In the original ABC algorithm, the employed bees exploit the given food sources in their neighborhood. Here we propose a novel differential evolution scheme for the employed bees to generate neighboring food sources. The differential evolution scheme consists of three steps: mutation, crossover, and selection.

In the mutation part, two parameters mutation rate (MR), insert times (IT) are introduced. For each incumbent individual, a uniformly random number is generated in the range of [0,1]. If it is less than MR, the mutant individual is obtained by operating the insert operation on the best individual π best in the population IT times; otherwise, the mutant individual is gained by operating the insert operation on a randomly selected individual IT times.

Next, partially mapped crossover (PMX) [19], a widely used crossover operator for permutation-based, is used in the crossover part. The incumbent individual and the mutated individual will undergo the PMX operation with a crossover rate (CR) to obtain the two crossed individuals. On the other hand, the two crossed individuals are the same as the mutated individual with about a probability of (1-CR).

Following the crossover operation, the selection is conducted. The one with the lowest value of the objective function among the two crossed individuals and

the incumbent individual will be accepted. In other words, if either of these two crossed individuals yields a better makespan than the incumbent individual, then the better individual will replace the incumbent one and become a new member in the population; otherwise, the old individual is retained.

3.3 Onlooker Bee Phase

There are NP onlooker bees and each communicates with its corresponding employed bee. After the probability selection, also called the wheel selection, a modified variable neighborhood search (VNS) [20] is incorporated into our algorithm as a hybrid strategy to further improve the performance. We use two structures of neighborhoods, which are referred to as the insert local search and the swap local search. The procedures of the insert local search and the swap local search are given in Fig. 1, where u and v are two positive integers chosen randomly in the range of $[1,n]$. The local search combining both the insert local search and the swap local search is illustrated as follows:

Step1. Perform the insert local search. If the individual is improved, go back to step 2; otherwise end the procedure.
Step2. Perform the swap local search. If the individual is improved, go back to step 1; otherwise end the procedure.

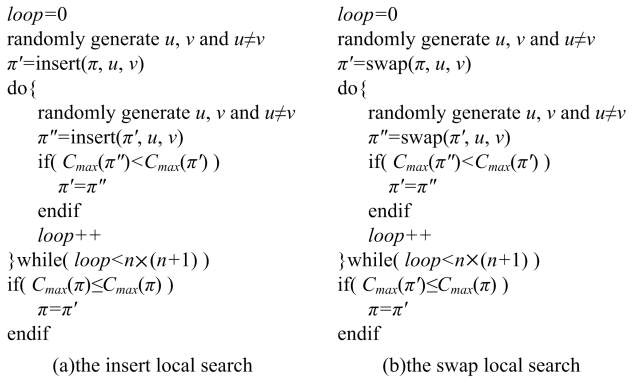

Fig. 1. The procedure of modified variable neighborhood search

If the new food source obtained is better than or equal to the incumbent one, the new food source will be memorized in the population. The onlooker bee phase in the IDABC algorithm provides the intensification of the local search on the relatively promising solutions.

3.4 Scout Bee Phase

As it has been stated in the basic ABC algorithm, the scout bees search randomly in the predefined space. This procedure will increase the population diversity and

avoid getting trapped in local optima, whereas this will also decrease the search efficacy. The new food source of a scout bee is produced as follows. Firstly, a tournament selection with the size of two is applied due to its simplicity and efficiency. That is, a scout bee selects two individuals π_a and π_b randomly from the population, and compares them with each other. If the makespan of π_a is smaller than that of π_b, π_a wins the tournament and π_b loses. Then, the scout bee generates a new solution new by employing the destruction and construction procedures of the iterated greedy (IG) algorithm [21]. The destruction and construction procedures are performed on the better individual π_a in the tournament selection and it has one parameter: destruction size (d). After that, the new solution π_{new} becomes a new member in the population and the worse one π_b is discarded. In this phase, the number of scout bees is ten percent of that of food sources.

4 Simulation Results and Comparisons

4.1 Experimental Setup

To fully examine the performance of the IDABC algorithm, an extensive experimental comparison with other powerful methods are provided. The IDABC algorithm was coded in Visual C++ and run on an Intel Pentium 3.06 GHz PC with 2 GB RAM under Windows 7 operating system.

The test problems used in experiments are the 98 different benchmark problems which are presented in [5]. The sizes of these problems vary from 10 jobs and 5 stages to 15 jobs and 10 stages. The processing times of the operations in these 98 instances are uniformly distributed between 3 and 20. Three characteristics that define a problem are the number of jobs, the number of stages and the number of identical machines at each stage. Therefore, we use the notation of j10c5b1 for instance, which means a 10-job, 5-stage problem. There are 55 easy problems and 43 hard problems. The problems with a and b machine layouts are easy problems. The problems with c, d, and e machine layouts are relatively harder to solve, so they are mostly grouped as hard problems.

The benchmark problems taken from Carlier and Neron [5] are relative simple. Thus, another 10 benchmark problems generated by Liao et al. [12] recently are also utilized in this section. In each problem, there are 30 jobs and 5 stages. At each stage, the machine number has a uniform distribution in the range of [3,5]. The processing times in these problems are within [1,100].

In the proposed IDABC algorithm, there are five main parameters: NP, MR, CR, IT, and d. We set the parameters $NP{=}n$ (the number of jobs), $MR{=}0.8$, $CR{=}0.8$, $IT{=}4$, and $d{=}4$ in the following experiments.

4.2 Computational Results

Comparison of Carlier and Neron's Benchmarks. Several meta-heuristics have been applied to Carlier and Neron's benchmark problems. To evaluate the

performance of the proposed IDABC algorithm in solving the HFS problem under the criterion of makespan minimization, the IDABC algorithm was compared with a B&B method [4], an AIS [9], an ACO [10], a GA [8], a QIA [11], and a PSO algorithm [12]. The maximum run time of the algorithm was set at 1600s or until the lower bound (LB) [3], [4] was reached. If the LB was not found within this time limit, the search was stopped and the best solution was accepted as the final solution.

In order to establish more accurate and objective comparisons, the computational results of these compared algorithms are obtained from their original papers. For each test problem, the proposed IDABC algorithm was run independently twenty times and the performance of all the compared algorithms was summarized in Table 1. In Table 1, Solved means the number of problems which the algorithm can solve, and Deviation denotes the average relative percentage error to LB.

Table 1. Comparison results on Carlier and Neron's benchmark problems

Algorithm	Easy problems		Hard problems	
	Solved	Deviation	Solved	Deviation
B&B	53	2.17%	24	6.88%
AIS	53	0.99%	24	3.13%
ACO	45	0.92%	18	3.88%
GA	53	0.95%	24	3.05%
QIA	29	0	12	5.04%
PSO	53	0.95%	24	2.85%
IDABC	55	0.94%	43	2.82%

As it can be noticed from Table 1, the machine layouts have an important effect on the complexity of problems that affects solution quality. In the 55 easy problems and 43 hard problems, B&B, AIS, GA, and PSO can solve 53 easy problems and 24 hard problems, ACO can solve only 45 easy problems and 18 hard problems, QIA can solve only 29 easy problems and 12 hard problems, while the proposed IDABC algorithm can solve all the 98 problems. The average percentage deviation values of the easy and hard problems generated by IDABC are equal to 0.94% and 2.82%. For the 55 easy problems, QIA has a zero deviation value but it can solve only 29 of the 55 easy problems. The performance of PSO is comparable with the proposed IDABC, but it still cannot solve problems as many as the proposed IDABC. Thus, it is concluded that the IDABC algorithm is more effective and efficient in comparison with other algorithms for Carlier and Neron's benchmark problems.

Comparison of Liao's Benchmarks. For Liao's benchmark problems, two meta-heuristics: AIS and PSO, have been applied to the problems in [12]. The computational results are shown in Table 2, where the experimental data of AIS and PSO were obtained from their original papers, and the IDABC algorithm was run twenty independent replications for each problem. The execution time

for each problem is limited to 200 seconds. In Table 2, AVE, MIN, and STD indicate the values of average, minimum, and standard deviation, respectively. T presents the average computation time (given in seconds) that the solution converges to the final solution.

Table 2. Comparison results on Liao's benchmark problems

Problem	AIS				PSO				IDABC			
	AVE	MIN	STD	T(s)	AVE	MIN	STD	T(s)	AVE	MIN	STD	T(s)
j30c5e1	485.35	479	2.58	99.44	474.70	471	**1.42**	96.16	**465.15**	**463**	1.50	**56.81**
j30c5e2	620.70	619	1.63	80.24	616.25	**616**	0.44	55.28	**616**	**616**	**0**	**1.51**
j30c5e3	625.70	614	4.81	116.70	610.25	602	4.70	64.56	**596.4**	**593**	**1.70**	**49.14**
j30c5e4	588.55	582	3.38	108.63	577.10	575	1.52	86.98	**566.2**	**565**	**1.20**	**39.29**
j30c5e5	618.75	610	3.42	101.19	606.80	605	**1.11**	79.84	**602**	**600**	1.56	**57.67**
j30c5e6	625.75	620	3.01	100.47	612.50	605	3.49	67.99	**603.05**	**601**	**1.47**	**55.02**
j30c5e7	641.30	635	4.67	93.56	630.60	629	0.75	87.18	**626**	**626**	**0**	**18.68**
j30c5e8	697.50	686	5.14	100.68	684.20	678	2.50	97.67	**674.65**	**674**	**0.88**	**55.18**
j30c5e9	670.20	662	3.85	100.75	654.65	651	1.87	83.80	**643.65**	**642**	**1.04**	**67.49**
j30c5e10	613.45	604	5.33	89.29	599.75	594	5.28	77.46	**576.25**	**573**	**1.52**	**76.05**
Average	618.73	611.10	3.78	99.09	606.68	602.60	2.31	79.69	**596.94**	**595.3**	**1.08**	**47.69**

In Table 2, the smallest values of AVE, MIN, STD, and T in the rows are shown in bold, respectively. It can be noted that the overall mean values of AVE, MIN, and STD yielded by the IDABC algorithm are equal to 596.94, 595.3, and 1.08, respectively, which are much better than those generated by AIS and PSO. Besides, the average computation time T of IDABC: 47.69 seconds is much shorter than that generated by AIS and PSO. From these observations, it is shown that the IDABC algorithm can obtain a better solution than AIS and PSO in an obviously shorter computational time. This means that the IDABC algorithm can converge to the good solutions faster than AIS and PSO. Also, it can be seen that the IDABC algorithm is more robust than both AIS and PSO for Liao's benchmark problems.

5 Conclusions

This paper establishes the model of the HFS problem by employing the vector representation and presents an improved discrete artificial bee colony (IDABC) algorithm for the HFS problem to minimize the makespan. Our future work is to extend the IDABC algorithm to other kinds of scheduling problems such as stochastic scheduling and multi-objective scheduling.

Acknowledgments. The authors are grateful to Carlier, Neron, and Liao for making the benchmark set available and to the anonymous reviewers for giving us helpful suggestions. This work was supported by National Natural Science Foundation of China (Grant no. 61174040, 61104178), the Fundamental Research Funds for the Central Universities.

References

1. Pinedo, M.: Scheduling: theory algorithms and systems. Prentice-Hall, Englewood Cliffs (2002)
2. Gupta, J.N.D.: Two-stage hybrid flowshop scheduling problem. J. Oper. Res. Soc. 39, 359–364 (1988)
3. Santos, D.L., Hunsucker, J.L., Deal, D.E.: Global lower bounds for flow shops with multiple processors. Eur. J. Oper. Res. 80(1), 112–120 (1995)
4. Neron, E., Baptiste, P., Gupta, J.N.D.: Solving hybrid flow shop problem using energetic reasoning and global operations. Omega 29(6), 501–511 (2001)
5. Carlier, J., Neron, E.: An exact method for solving the multi-processor flow-shop. RAIRO-Oper. Res. 34(1), 1–25 (2000)
6. Ruiz, R., Serifoglu, F.S., Urlings, T.: Modeling realistic hybrid flexible flowshop scheduling problems. Comput. Oper. Res. 35(4), 1151–1175 (2008)
7. Nawaz, M., Enscore, E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow shop sequencing problem. Omega 11, 91–95 (1983)
8. Kahraman, C., Engin, O., Kaya, I., Yilmaz, M.K.: An application of effective genetic algorithms for solving hybrid flow shop scheduling problems. Int. J. Comput. Intell. Sys. 1(2), 134–147 (2008)
9. Engin, O., Doyen, A.: A new approach to solve hybrid flow shop scheduling problems by artificial immune system. Future Gener. Comp. Sys. 20, 1083–1095 (2004)
10. Alaykyran, K., Engin, O., Doyen, A.: Using ant colony optimization to solve hybrid flow shop scheduling problems. Int. J. Adv. Manuf. Tech. 35, 541–550 (2007)
11. Niu, Q., Zhou, T., Ma, S.: A quantum-inspired immune algorithm for hybrid flow shop with makespan criterion. J. Univers. Comput. Sci. 15, 765–785 (2009)
12. Liao, C.J., Tjandradjaja, E., Chung, T.P.: An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. Appl. Soft. Comput. 12(6), 1755–1764 (2012)
13. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Technical report, Computer Engineering Department, Engineering Faculty, Erciyes University (2005)
14. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony algorithm. J. Global. Optim. 39(3), 459–471 (2007)
15. Karaboga, D., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Appl. Soft. Comput. 8(1), 687–697 (2008)
16. Wardono, B., Fathi, Y.: A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities. Eur. J. Oper. Res. 155, 380–401 (2004)
17. Oguz, C., Zinder, Y., Do, V.H., Janiak, A., Lichtenstein, M.: Hybrid flow-shop scheduling problems with multiprocessor task systems. Eur. J. Oper. Res. 152, 115–131 (2004)
18. Oguz, C., Ercan, M.F.: A genetic algorithm for hybrid flow shop scheduling with multiprocessor tasks. J. Scheduling 8, 323–351 (2005)
19. Goldberg, D.E., Lingle, R.J.: Alleles, loci and the traveling salesman problem. In: 1st International Conference on Genetic Algorithms and their Application, pp. 154–159. Lawrence Erlbaum (1985)
20. Mladenovic, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. 24, 1097–1100 (1997)
21. Ruiz, R., Stutzle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur. J. Oper. Res. 177(3), 2033–2049 (2007)