Neurocomputing 148 (2015) 248–259

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems

Zhe Cui, Xingsheng Gu *

Key Laboratory of Advanced Control and Optimization for Chemical Process, East China University of Science and Technology, Ministry of Education, Shanghai 200237, China

ARTICLE INFO

Article history:
Received 3 January 2013
Received in revised form
8 July 2013
Accepted 19 July 2013
Available online 25 June 2014

Keywords:
Hybrid flow shop problem
Scheduling
Mathematical model
Artificial bee colony
Orthogonal test

ABSTRACT

As a typical NP-hard combinatorial optimization problem, the hybrid flow shop (HFS) problem is widely existing in manufacturing systems. In this article, the HFS problem is modeled by vector representation, and then an improved discrete artificial bee colony (IDABC) algorithm is proposed for this problem to minimize the makespan. The proposed IDABC algorithm combines a novel differential evolution and a modified variable neighborhood search to generate new solutions for the employed and onlooker bees, and the destruction and construction procedures are used to obtain solutions for the scout bees. Moreover, an orthogonal test is applied to efficiently configure the system parameters, after a small number of training trials. The simulation results demonstrate that the proposed IDABC algorithm is effective and efficient comparing with several state-of-the-art algorithms on the same benchmark instances.

1. Introduction

Production scheduling is a decision-making process that plays a crucial role in manufacturing and service industries. It concerns how to allocate available production resources to tasks over given time periods, aiming at optimizing one or more objectives [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. The HFS problem, also called multi-processor or flexible flow shop, widely exists in real manufacturing environments, such as chemical, oil, food, tobacco, textile, paper, and pharmaceutical industries. In the HFS problem, it is assumed that the jobs have to pass through all stages in the same order and that there are at least one stage having multiple machines. Additionally, one machine can only process one job at a time. As to one single job, it can be processed by only one machine at a time. Preemption of processing is not allowed. The problem consists of assigning the jobs to machines at each stage and ordering the jobs assigned to the same machine in order to minimize some criteria.

As to the computational complexity, since even the two-stage HFS problem is strongly NP-hard [2,3] on minimizing the maximum completion time (makespan), the multi-stage HFS problem is at least as difficult as the two-stage one. Despite of the intractability, the HFS problem has great significance in both engineering and theoretical fields. Therefore, it is meaningful to develop an effective and efficient approach for this problem.

Among a large number of algorithms on this problem, the authors opt for a simple classification with three different classes [4,5]: exact algorithms, heuristics, and meta-heuristics. The branch and bound (B&B) method is the preferred exact algorithm when solving the HFS problem. Arthanary and Ramaswamy [6] developed a B&B method for a simple HFS problem with only two stages in the 1970s. Santos et al. [7] presented a global lower bound for makespan minimization that has been used to analyze the performance of other algorithms. Neron et al. [8] used the satisfiability tests and time-bound adjustments based on the energetic reasoning and global operations, in order to enhance the efficiency of another kind of B&B method proposed in Carlier and Neron [9].

Several heuristics have also been extensively studied. One efficient heuristic algorithm is developed by Gupta [2] to find a minimum makespan schedule for the two-stage HFS problem in which there is only one machine at the stage two. And the similar problem was studied later with the objective of minimizing the total number of tardy jobs in [10]. Brah and Loo [11] investigated some excellent flow shop heuristics for the HFS problem with the criteria makespan and mean flow time. They also examined the effects of the problem characteristics and the performance of the heuristics by using regression analysis. With advanced

* Corresponding author. Tel.: +86 2164253463; fax: +86 2164253121.
E-mail address: xsgu@ecust.edu.cn (X. Gu).

http://dx.doi.org/10.1016/j.neucom.2013.07.056
0925-2312/

statistical tools, Ruiz et al. [12] tested several heuristics in the realistic HFS problem and suggested that the modified NEH heuristic [13] outperformed other dispatching rules. Ying and Lin [14] proposed an effective and efficient heuristic with a simple conception to solve the multistage hybrid flow shop with multiprocessor tasks.

During the past decades, meta-heuristics, which can generate approximate solutions close to the optimum but with considerably less computational time, has become a new and effective approach to solve the HFS problem. The genetic algorithm (GA) was applied by several researchers to solve the HFS problem with the criterion makespan [15,16]. On the basis of vertebrate immune system, Engin and Doyen [17] 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. [18] introduced an improved ant colony optimization (ACO) algorithm, employing the same formula as the classical ant system algorithm, except using a different starting solution procedure. Niu et al. [19] presented a quantum-inspired immune algorithm (QIA) for the HFS problem to minimize makespan. Liao et al. [20] 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 [21]. Applying the ABC algorithm to some numerical optimization problems, Basturk and Karaboga suggested that the ABC algorithm has a better performance than other population-based algorithms [22–24]. Due to the ABC algorithm's remarkable advantages [25], including simple structure, easy implementation, quick convergence, and fewer control parameters, it has been found an increasingly wide utilization in a variety of fields [26–29]. Nevertheless, on account of its continuous nature, the studies on ABC algorithm for production scheduling problems are still in their infancy. In the latest study, some researchers have successfully applied ABC-based algorithm to a series of scheduling problems, such as permutation flow shop scheduling problem [30], blocking flow shop scheduling problem [31,32], lot-streaming flow shop scheduling problem [33,34], multi-objective flexible job shop scheduling problem[35], and steelmaking process scheduling problem [36].

Although the ABC algorithm has solved some combinatorial optimization problems, there is no published work using the ABC-based algorithm for the HFS problem. In this article, the model of the HFS problem is established by employing the vector representation, and then an improved discrete artificial bee colony (IDABC) algorithm is proposed for the problem to minimize the makespan. In the IDABC algorithm, an efficient population initialization based on the NEH heuristic is incorporated into the random initialization to generate an initial population with certain quality and diversity. Meanwhile, a novel differential evolution operation, which includes mutation, crossover, and selection, is applied to generating new neighbor food sources in the employed bee phase. Moreover, a modified variable neighborhood search is developed to further improve the performance in the onlooker bee phase. Furthermore, inspired by the iterated greedy algorithm, the destruction and construction procedures are employed in the scout bee phase to enrich the population and to avoid premature convergence. Besides, the orthogonal experimental design is used to provide a receipt for turning the adjustable parameters of the IDABC algorithm. Finally, the simulation results on benchmarks demonstrate the effectiveness and efficiency of the proposed IDABC algorithm.

The rest of the article 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 tests on parameter selection and 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 $\boldsymbol{J}=\{1,2,\ldots,i,\ldots,n-1,n\}$ that have to be performed on s stages $S=\{1,2,\ldots,j,\ldots,s-1,s\}$, 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 setup and release times of all jobs are negligible. Preemption is not allowed and intermediate buffer capacities between two successive stages are unlimited. 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 [37]. When using the matrix representation, a solution is represented by an $s \times n$ matrix whose elements are all real numbers [38]. Each job is assigned with a real number, whose integer part is the machine number on which the job is to be processed and whose fractional part is used to sort the jobs on the same machine. The advantages of this representation are that it can avoid the infeasible solution and that it also covers the entire solution space. However, there are two disadvantages associated with this representation. The first disadvantage is that there exist a large number of matrices associated with only one solution for the problem, which means that it is not a one-to-one correspondence. The other one is that it is cumbersome to work with.

In this article, the authors employ the vector representation, which considers the sequence of jobs only at stage one. This subset sequence should contain the collection of all potentially good solutions for the problem and should be a one-to-one correspondence. Most importantly, it is very convenient to design and operate with 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 [39,40]. 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:

$$\text{Minimize } C_{max}(\pi_1) = \max_{i=1,2,\ldots,n}\{C_{\pi_s(i),s}\} \tag{1}$$

Subject to

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