Int. J. Production Economics 141 (2013) 137–145

Contents lists available at SciVerse ScienceDirect

Int. J. Production Economics

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

Particle swarm optimization with cocktail decoding method for hybrid flow shop scheduling problems with multiprocessor tasks

Fuh-Der Chou*

Department of Industrial Management, Ching Yun University, Jung-Li, Tao Yuan, Taiwan, ROC

ARTICLE INFO

Article history:
Received 14 September 2011
Accepted 14 May 2012
Available online 23 May 2012

Keywords:
Multiprocessor task
Hybrid flow shop
Particle swarm optimization
Lower bound

ABSTRACT

This paper addresses the problem of multiprocessor task-scheduling in a hybrid flow shop (HFS) problem to minimize the makespan. Due to the complex nature of an HFS problem, it is decomposed into the following two sequential decision problems: determining the job permutation in stage 1, followed by a decoding method to assign jobs into each machine in subsequent stages when designing a heuristic algorithm. The decoding method plays a pivotal role for improving the solution quality of any algorithm for the HFS problem. However, the majority of existing algorithms ignores the problem and is only concerned with the first decision problem. This study emphasizes the importance of the decoding method via a small test, and searches for a number of solid decoding methods that can be incorporated into the cocktail decoding method. Then, this study develops a particle swarm optimization (PSO) algorithm that can be combined with the cocktail decoding method. In the PSO, a variety of job sequences are generated using the PSO procedure in stage 1, and the cocktail decoding method is used to assign the jobs to machines in sequential stages. Moreover, a modified lower bound is introduced. Computational results show that the proposed lower bound is competitive, and with the help of the cocktail decoding method, the proposed PSO, and even the adoption of a standard PSO framework, significantly outperforms the majority of existing algorithms in terms of quality of solutions, especially for large problems.

1. Introduction

Hybrid flow shop (HFS) problems have recently attracted considerable attention from researchers. HFS is an extension model of the flow shop system that relaxes the assumption of the one-stage-one-machine pattern based on the entire issue that machines operate in parallel in at least one stage to prevent the whole system from being blocked (e.g., a breakdown) by a single machine in many industrial environments. Thus, the HFS problem has two basic characteristics as follows: (1) a set of n jobs is sequentially processed in a series of m stages; and (2) at least one of the stages has m_i machines in parallel. Rao (1970) conducted the early work on HFS scheduling, and variants of HFS problems with different constraints and criteria have been studied by many researchers since then. Gupta (1988) considered the two-stage special case with a single machine in the first stage and two identical machines in the second stage, and showed that the case was NP-hard. Brah and Hunsucker (1991) proposed a branch and bound (B&B) algorithm to minimize the makespan for HFS problems. Portmann et al. (1998) applied a genetic algorithm (GA) within a B&B algorithm to further improve the performance of the B&B algorithm. With the computational complexity of HFS problems, a number of heuristics and metaheuristic algorithms have also been developed to obtain good enough solutions in a short time for medium-to-large problems (Haouari and M'Hallah, 1997; Brah and Loo, 1999; Lin and Liao, 2003; Jin et al., 2006; Janiak et al., 2007; Alaykiran et al., 2007). The studies mentioned above focused on the model of the one-job-one-machine pattern, in which each job is processed on only one machine at a given time. Ruiz and Vazquez-Rodriguez (2010) conducted a comprehensive survey of relevant research on HFS problems from 1970 to 2009, clearly revealing that among the majority of HFS problems that were concerned with the one-job-one-machine model, only a few studies discussed multiprocessor tasks where each job is processed on a number of identical machines simultaneously at each stage.

This study focuses on multiprocessor task-scheduling in a HFS system. The problem can be encountered in a number of industrial environments, such as berth allocation of container terminals, real-time machine-vision systems, and work force management. In the considered problem, a given set of n jobs, each with k tasks, is processed sequentially from stage 1 to stage k. Each stage consists of Pm_i $(i=1,\dots,k)$ identical parallel machines. Each job j is characterized by p_{ij} and $size_{ij}$, indicating that the tasks of job j are processed by

* Tel.: +886 3 4581196; fax: +886 3 4683298.
E-mail address: fdchou@tpts7.seed.net.tw

0925-5273/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ijpe.2012.05.015

$size_{ij}$ parallel machines simultaneously for p_{ij} time period without interruption at stage i. Any machine can only process one task at a time. All jobs and machines are available at time zero, setup times are negligible, and preemption is not allowed. The problem aims to minimize the maximum completion time of all jobs, i.e., the makespan. Using the standard three-field notation, the problem can be described as follows: $Fk(Pm_1,\ldots,Pm_k)|size_{ij}|C_{\max}$.

In literature, several heuristic algorithms have been developed to solve multiprocessor task HFS problems. Oguz and Ercan (1997) and Oğuz et al. (2003) examined the multiprocessor task in a two-stage HFS problem motivated by a computer vision system. The two studies presented different constructive heuristic algorithms based on several dispatching rules wherein Oğuz and Ercan assumed a unit-processing time for all jobs. Ying and Lin (2009) developed the first simple constructive heuristic algorithm with a computational complexity of $O(n^2)$ to solve a general $Fk(Pm_1,\ldots,Pm_k)|size_{ij}|C_{\max}$ problem. Kahraman et al. (2010) developed a parallel greedy algorithm, which is an effective stochastic local search algorithm that is easy to implement and has great potential to provide good solutions, for the HFS problem with multiprocessor tasks. Their results are competitive compared with the results of Oğuz and Ercan (2005). Lahimer et al. (2011) developed the climbing depth-bounded adjacent discrepancy search (CDADS) based on a limited discrepancy search, which was effective for both small and large problems.

Recently, metaheuristic algorithms have been developed for multiprocessor task-scheduling in an HFS system, including simulated annealing (SA) by Wang et al. (2011), tabu search (TS) by Oğuz et al. (2004), GA (Sivrikaya-Serifoglu and Ulusoy, 2004; Oğuz and Ercan, 2005) particle swarm optimization (PSO) algorithm by Tseng and Liao (2008), and ant colony optimization (ACO) by Ying and Lin (2006). These studies decomposed the problem into two decision problems, namely, the determination of job permutation in stage 1 and the assignment of jobs to each machine in the subsequent stages. For two sub-problems, the metaheuristic algorithms mentioned above usually follow a two-phase structure in which different permutations of the jobs are generated in the first phase, and then a decoding method is used to assign the jobs to machines for the remaining stages according to the first come, first served (FCFS) manner (i.e., list scheduling). Despite the relative popularity of list scheduling for HFS problems, it still has a significant drawback. Suppose the case of the processor requirement $size_{ij}$ of the selected job with an earlier completion time at the previous stage exceeds the number of available machines, and thus, the job must be postponed for processing to satisfy the processor requirement constraint. Consequently, unnecessary idle times in the machines at the current stage occur and longer makespan is obtained. In addition, Liao et al. (2006) proposed a non-permutation procedure incorporated in the GA and TS algorithms and showed that the non-permutation schedule could be superior to the permutation schedule in a traditional flow shop, especially for due-date based criteria, through a comprehensive computational experiment. Therefore, the optimal solution may not be able to produce if only the list schedule (LS) is used for subsequent stages when the permutations of jobs are exhausted for stage 1. For this drawback, Jouglet et al. (2009) proposed a new memetic algorithm (MA), in which the permutation of jobs is generated by a GA operator, and then a constraint programming (CP)-based B&B algorithm is used to schedule the unscheduled jobs in the remaining stages and obtain an optimal/feasible solution for each given permutation of jobs. Wang et al. (2011) developed a first-fit method to reduce the idle time caused by the LS and combined it with SA for the HFS problem with multiprocessor tasks.

The common features of most existing metaheuristic algorithms for solving multiprocessor task scheduling in an HFS environment are based on the two-phase structure, in which the permutation of jobs is stochastically generated at stage 1, and then only a decoding method, namely, the LS method, is used to assign the jobs to the machines for sequel stages. A combined method with a variety of decoding methods is not frequently found in the HFS problem with multiprocessor tasks. As a result, this paper intends to develop a cocktail decoding method that can be combined with several popular priority rules in literature, and embedded in a standard PSO to solve the HFS problem with multiprocessor tasks. Moreover, a modified lower bound is also proposed based on existing lower bounds in the literature to obtain a tighter lower bound.

The remainder of this paper is organized as follows. A number of relevant simple priority rules are first reviewed to identify important issues in the rules to be investigated, and then a cocktail decoding method is proposed. Section 3 discussed the methodology of the PSO algorithm. The cocktail decoding method is embedded in the PSO to evaluate the fitness value for each particle. Section 4 discusses various lower bound formulations. Section 5 evaluates the proposed lower bound and PSO algorithm using a computational experiment. Finally, concluding remarks and directions for future research are summarized in Section 6.

2. Cocktail decoding method

As mentioned above, the performance of a heuristic algorithm in solving the HFS problem with multiprocessor tasks is influenced by two factors, namely, the determination of job sequence at stage 1 and the assignment of jobs to each machine for the subsequent stages. Jouglet et al. (2009) named the method to assign unscheduled jobs on each machine for the subsequent stages decoding method. Wang et al. (2011) stated that the decoding method exhibits certain influences on the solution quality for the considered problem. Moreover, priority rules are popular for assigning the unscheduled jobs to machines in the scheduling area, and thus, a variety of famous priority rules are adopted as decoding methods, which are described as follows:

- Shortest processing time (SPT), where the jobs are sorted by the ascending order of their processing times p_{kj} for stage k.
- Longest processing time (LPT), where the jobs are sorted by the descending order of their processing times p_{kj} for stage k.
- Shortest total remaining processing time (STRPT), where the jobs are sorted by the ascending order of their total processing times $\sum_{i=k}^{m} p_{ij}$ from stage k to last stage m.
- LTRPT (largest total remaining processing time) where the jobs are sorted by the descending order of their total processing times $\sum_{i=k}^{m} p_{ij}$ from stage k to last stage m.
- Smallest processor requirement (SPR), where the jobs are sorted by the ascending order of their processor requirements $size_{kj}$ for stage k.
- Largest processor requirement (LPR), where the jobs are sorted by the descending order of their processor requirements $size_{kj}$ for stage k.
- Smallest occupied capacity (SOC), where the jobs are sorted by the ascending order of their processing time multiplied by processor requirement $p_{kj} \times size_{kj}$ for stage k.
- Largest occupied capacity (LOC), where the jobs are sorted by the descending order of their processing time multiplied by processor requirement $p_{kj} \times size_{kj}$ for stage k.
- Smallest total remaining occupied capacity (STROC), where the jobs are sorted by the descending order of a total of their processing time multiplied by processor requirement $\sum_{i=k}^{m} p_{ij} \times size_{ij}$ from stage k to the last stage m.
- Largest total remaining occupied capacity (LTROC), where the jobs are sorted by the descending order of a total of their