Knowledge-Based Systems 105 (2016) 68–82

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

A branch-and-bound algorithm for minimizing the total weighted completion time on parallel identical machines with two competing agents

Wen-Chiung Lee[a], Jen-Ya Wang[b,*], Mei-Chun Lin[a]

[a] Department of Statistics, Feng Chia University, Taichung, Taiwan, ROC
[b] Department of Computer Science and Information Management, Hungkuang University, No. 1018, Sec. 6, Taiwan Boulevard, Shalu District, Taichung 43302, Taiwan, ROC

ARTICLE INFO

Article history:
Received 15 August 2015
Revised 22 April 2016
Accepted 7 May 2016
Available online 10 May 2016

Keywords:
Discrete optimization
Branch-and-bound algorithm
Two-agent scheduling
Multi-machine scheduling
Lower bound

ABSTRACT

Scheduling with two competing agents has drawn a lot of attention lately. However, most studies focused only on single-machine problems. In reality, there are many machines or assembly lines to process jobs. This study explores a parallel-machine scheduling problem. The objective is to minimize the total weighted completion time of jobs from agent 1 given a bound of the maximum completion time of jobs from agent 2. We develop a branch-and-bound algorithm to solve the problems with fewer jobs. In addition, we propose genetic algorithms to obtain the approximate solutions. Computational results are given to evaluate the performance of the proposed algorithms.

1. Introduction

To survive today's intense competition and meet challenges that develop rapidly, industries must reduce cost, shorten waiting times, maximize profits, and give satisfaction. To achieve the above goals, many industries use parallel-machine production. Some issues with this type of production need to be managed. For example, the orders may not come from a sole source. Some orders need to be completed as soon as possible, and some do not, or some orders are of different importance and incur different penalties for late completion. In light of these observations, jobs having various time priorities need to be carefully scheduled on parallel machines.

An example of a tea shop is given as follows. Several servers deal with daily orders. Some bulk orders may be placed a week ahead of time via online shopping, and some orders are placed verbally in front of a brick-and-mortar bar. Some orders may involve an important international conference, and heavy tardiness fines may be imposed for these orders. Meanwhile, the face-to-face customers should not be kept waiting for a long time. Clearly, traditional FCFS (First Come First Serve) rules are not good approaches to this type of production.

Multi-agent scheduling is a new concept that can be applied to many fields. The concept was first introduced in [1,2]. Baker and Smith [1] introduced a real-world instance commonly seen in R&D and production departments. An R&D department is concerned about meeting deadlines, but a production department is more concerned about a short response time. Agnetis et al. [2] considered some two-agent problems. Each agent deals with a set of non-preemptive jobs and wants to minimize a certain objective function, e.g., number of tardy jobs or total weighted completion time. Wan et al. [3] provided an example in which jobs are divided into two parts: regular jobs and maintenance activities. Meiners et al. [4] showed an example in the field of telecommunication. To improve network efficiency, they viewed a packet as two distinct sets of jobs: deadline jobs, which represent real-time packets in a network, and flow jobs, which represent other packets in the network. Leung et al. [5] discussed some two-agent scheduling problems and developed several mathematical properties. Each of the researchers above proposed a novel mathematical model, but neither provided experimental analyses. For more research into multi-agent scheduling problems, please refer to [6–14].

Most two-agent scheduling problems are focused on single-machine scheduling. Liu et al. [15] considered the processing time of a job as an increasing linear function of its starting time. Two agents compete to perform their respective jobs on a common single machine, and each agent has its own criterion to optimize. Mor and Mosheiov [16] investigated batch scheduling on a single

* Corresponding author. Fax: +886 4 26521921.
E-mail address: jywang@sunrise.hk.edu.tw (J.-Y. Wang).

http://dx.doi.org/10.1016/j.knosys.2016.05.012
0950-7051/© 2016 Elsevier B.V. All rights reserved.

machine. One agent minimizes the total completion time, and the other must complete all its jobs before a given time limit. Since this problem is not NP-hard, a linear-lime algorithm was proposed. Nong et al. [17] considered agents having their own jobs and competing for a single machine. The first agent minimizes the maximum weighted completion time, while the second minimizes the total weighted completion time. Due to the NP-hardness, a polynomial-time algorithm was proposed for providing a near-optimal schedule. Wu et al. [18] considered the learning effect in a single-machine two-agent scheduling problem. The first agent minimizes the total tardiness, and the second demands zero tardiness. A branch-and-bound (B&B) algorithm was proposed for scheduling jobs on a single machine. Lee et al. [19] considered a single-machine two-agent scheduling problem. The first agent minimizes a linear combination of the total completion time and the maximum tardiness given that no tardy jobs are allowed for the second agent. They developed another B&B algorithm for the single-machine scheduling problem.

However, Pinedo [20] indicated that multi-machine scheduling is important. First, in reality, few real-world applications involve jobs done by a single machine. Second, in theory, the combination of each locally optimal schedule for a single machine does not mean a globally optimal schedule. Therefore, two-agent scheduling on parallel machines merits further study.

In light of the above observations, it is necessary to develop a new optimization algorithm for scheduling jobs from multiple agents on multiple machines. Though many algorithms are available, the existing algorithms cannot always achieve optimality when the problem size is small. For example, the genetic algorithm in [21] only provides near-optimal solutions because of the nature of random walk. Moreover, past research cannot overcome both multi-machine and multi-agent issues at the same time. For example, the B&B algorithm in [18] was developed only for its dedicated single-machine problem. In addition, we cannot apply some B&B algorithms dedicated to single-machine scheduling, such as [19], to our problem. Therefore, a new B&B algorithm is needed.

In this paper, a parallel-machine two-agent scheduling problem is investigated. The objective is to minimize the total weighted completion time of the jobs from agent 1 given that the maximum completion time of the jobs from agent 2 cannot exceed a predefined limit. To ensure optimality, a new B&B algorithm is proposed for small problem instances. Moreover, this B&B algorithm also helps us to evaluate the performance of a genetic algorithm that is designed for generating approximate schedules. Experimental results show that the two complementary algorithms perform well when the problem size is less than 200.

The rest of the paper is organized as follows. The proposed problem is described in Section 2. In Section 3, a branch-and-bound (B&B) algorithm with several dominance properties and a lower bound is developed. In Section 4, a simple genetic algorithm (GA) is proposed to provide an initial schedule for the B&B algorithm. In Section 5, computational experiments are conducted. The conclusions are drawn in Section 6.

2. Problem definition

All the parameters used in this section are listed in Table 2.1. There are n jobs from two agents to be allocated to m identical machines. Note that we have n_1 jobs from agent 1 and n_2 jobs from agent 2, i.e., $n_1 + n_2 = n$. For each job j, the processing time, weight, and agent number are denoted by p_j, w_j, and I_j, respectively. For a schedule S, let $C_j^1(S)$ be the completion time of job $j \in AG_1$ and $C_{\max}^2(S) = \max\{C_j^2(S)\}$ be the maximum completion time of jobs in AG_2. The objective is to minimize $\sum w_j C_j^1$ subject to $C_{\max}^2 \le aF$, where a is a coefficient. In general, we set $a = 1$. That is, the problem can be denoted by $P_m|C_{\max}^2 \le aF|\sum w_j C_j^1$.

Table 2.1
The parameters used in this section.

Parameter	Description
n	The number of jobs
m	The number of machines
S	A schedule
M_k	Machine k
p_j	The processing time of job j
w_j	The weight of job j
$C_j(S)$	The completion time of job j in schedule S
ag_i	Agent i
AG_i	The set of jobs from ag_i
n_i	The number of jobs in AG_i
F	A constant limit for $C_{j\in AG_2}(S)$
a	The coefficient of F
I_j	$I_j = 1$ if $j \in AG_1$; $I_j = 2$ if $j \in AG_2$

Table 2.2
A problem instance.

Parameter	Value						
j	1	2	3	4	5	6	7
I_j	1	1	1	1	2	2	2
p_j	2	2	5	4	1	2	3
w_j	6	2	3	2			
$C_j(S)$	2	2	10	9	4	5	5
k	1	2	2	1	2	2	1

An example is shown in Table 2.2 and Fig. 2.1. Moreover, we have $n = 7$, $m = 2$, $a = 1$, and $F = 5$. Consider a schedule $S = (1, 7, 4|2, 5, 6, 3)$, where the vertical bar means a separator dividing jobs among machines. It is easy to check $C_{\max}^2 \le 5$ and obtain the objective cost, i.e., $\sum w_j C_j^1 = 6 \cdot 2 + 2 \cdot 2 + 3 \cdot 10 + 2 \cdot 9 = 64$. To meet the requirement of F, jobs from ag_2 are scheduled first and thus result in fragments, such as [0, 2] on M_1. In this instance, the remaining jobs from ag_1 perfectly utilize the fragments on the two machines. However, for other larger instances, to determine such optimal permutations requires many trials and errors. It is time-consuming, no matter what optimization algorithms we use. For more information about designing optimization algorithms, please refer to B&B for multiple agents [19,22,23] or B&B for multiple machines [24,25].

3. Branch-and-bound algorithm

In this section, some dominance properties are developed and a lower bound is proposed. With these properties, a branch-and-bound (B&B) algorithm is developed. For a path in the search tree (i.e., schedule), the nodes are visited in a depth-first-search (DFS) manner.

3.1. Dominance properties

Assume that S and S' are two schedules and that S' is obtained by interchanging two pairwise jobs (i.e., i and j) in S. Then we have $S = (\alpha, i, j, \beta)$ and $S' = (\alpha, j, i, \beta)$, where α is a partial determined sequence and β is a partial undetermined sequence. Let M_k be the current machine, J_k be the set of jobs allocated to M_k, and $t = \max\{C_{j\in\alpha\cap J_k}(S)\} = \max\{C_{j\in\alpha\cap J_k}(S')\}$. Then we have $C_i(S) = t + p_i$, $C_j(S) = t + p_i + p_j$, $C_j(S') = t + p_j$, and $C_i(S') = t + p_i + p_j$. The dominance properties are listed below. Since their proofs are similar, only the proof of the first dominance rule is given below.

Property 1. If $i \in AG_1$, $j \in AG_2$, and $t + p_i + p_j \le F$, then S' is dominated.