Applied Soft Computing 24 (2014) 277–283

Contents lists available at ScienceDirect

Applied Soft Computing

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

A memetic algorithm for the re-entrant permutation flowshop scheduling problem to minimize the makespan

Jianyou Xu[a,*], Yunqiang Yin[b], T.C.E. Cheng[c], Chin-Chia Wu[d], Shusheng Gu[a]

[a] *College of Information Science and Engineering, Northeastern University, Shenyang 110819, China*
[b] *Faculty of Science, Kunming University of Science and Technology, Kunming 650093, China*
[c] *Department of Logistics and Maritime Studies, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*
[d] *Department of Statistics, Feng Chia University, Taichung, Taiwan*

ARTICLE INFO

Article history:
Received 20 October 2012
Received in revised form 11 May 2013
Accepted 6 July 2014
Available online 18 July 2014

Keywords:
Re-entrant permutation flowshop scheduling
Memetic algorithm
Makespan

ABSTRACT

A common assumption in the classical permutation flowshop scheduling model is that each job is processed on each machine at most once. However, this assumption does not hold for a re-entrant flowshop in which a job may be operated by one or more machines many times. Given that the re-entrant permutation flowshop scheduling problem to minimize the makespan is very complex, we adopt the CPLEX solver and develop a memetic algorithm (MA) to tackle the problem. We conduct computational experiments to test the effectiveness of the proposed algorithm and compare it with two existing heuristics. The results show that CPLEX can solve mid-size problem instances in a reasonable computing time, and the proposed MA is effective in treating the problem and outperforms the two existing heuristics.

Introduction

In classical scheduling model, it is commonly assumed that each job visits each machine only once (see [1]). However, the assumption that each job is processed on each machine at most once may not be valid in some real-life situations. A job may be processed by the same machine twice or more in practice. Such a processing environment is called the "re-entrant flowshop" in the literature. Examples of the re-entrant flowshop can be found in semiconductor wafer manufacturing [2], signal processing [3], and manufacturing of printed circuit boards [4–7].

The defining characteristic of the re-entrant flowshop comprising m machines is that every job in the shop must be processed on the m machines in the following order: M_1, M_2,. . ., M_m; M_1, M_2,. . ., M_m;. . .; and M_1, M_2,. . ., M_m, with L re-entrants (or levels), i.e., starting on machine M_1 and ending on M_m, and the job sequence is the same on any machine at each level. The corresponding scheduling problem is known as the re-entrant flowshop (RFS) scheduling problem. With the imposition of the constraint that no passing of the jobs is allowed, the problem becomes the RPFS scheduling problem.

For the RPFS scheduling problem to minimize the makespan, Pan and Chen [8] proposed three extended mixed binary integer program (BIP) formulations and six heuristics to treat the problem with up to $n = 8$ jobs. They used the Lingo optimizer to solve small-sized instances of the problem. Alfieri [9] studied a multi-objective flowshop scheduling problem arising in the cardboard industry, where the objective is to develop a modular decision support system for daily workload planning. The problem setting includes multi-machine stations, sequence-dependent setup times, and work calendars on resources, re-entrant flows, external operations, and transfer batches between stations. Liu [10] applied a genetic algorithm (GA) based method to solve the RFSP scheduling problem involving multiple orders per job, where the objective is to minimize the total weighted tardiness of all the jobs. Rau and Cho [11] proposed GAs to solve the inspection allocation problem in a re-entrant production system. Lin and Lee [12] presented a GA that encodes the problem as multi-level chromosomes to reflect the dependent relationship between the re-entrant possibility and resource consumption. For more studies of the RPFS scheduling problem in different settings, we refer the reader to Choi et al. [13], Chu et al. [14], and Boudhar and Meziani [15]. For more detailed research results on variants of the reentrant scheduling problem, the reader may refer to the surveys of Bellman and Ernest [16], Uzsoy et al. [7], and Lin and Lee [17], among others.

Except for the few studies cited above, research on the RPFS scheduling problem is relatively unexplored. Moreover, searching

* Corresponding author. Tel.: +86 18602470166.
E-mail addresses: xujianyou@mail.neu.edu.cn, zhyouxu@163.com (J. Xu).

http://dx.doi.org/10.1016/j.asoc.2014.07.002
1568-4946/© 2014 Elsevier B.V. All rights reserved.

for the optimal solution for the RFSP scheduling problem is only viable for instances involving a small number of jobs. For example, Pan and Chen [8] used the Lingo optimizer to solve problem instances with up to $n = 8$ jobs. Chen [18] developed a branch-and-bound algorithm incorporating five lower bounds to solve problem instances with up to $n = 15$ jobs. Chen et al. [19] further proposed a hybrid tabu search and evaluated its performance with solutions obtained from Lingo for instances with only up to $n = 4$ jobs. In view of these observations, we tackle in this paper the RPFS scheduling problem to minimize the makespan by using the CPLEX solver and a memetic algorithm.

The remainder of this paper is organized as follows: In "Notation and problem statement" section we introduce and formulate the problem. In "A memetic algorithm" section we develop a memetic algorithm to obtain near-optimal solutions for the problem. In "Computational experiments" section we provide computational results to assess the performance of the proposed algorithm and compare it with two existing heuristics. We conclude the paper and suggest topics for future research in the last section.

Notation and problem statement

In this section we first introduce the notation to be used throughout the paper, followed by the problem formulation.

Notation for the problem

n	the number of jobs
m	the number of machines
L	the number of levels for each job
M_i	machine number i, where $i = 1,2,\ldots,m$
J_{ij}^l	the code for job j on machine i at level l, where $i = 1, 2,\ldots, m$; $j = 1, 2,\ldots, n$; and $l = 1, 2,\ldots, L$
O_{ik}^l	denotes the operation of J_{ik}^l on machine k at level l, where $i = 1, 2,\ldots, n$; $k = 1, 2,\ldots, m$; and $l = 1, 2,\ldots, L$
p_{ij}^l	the processing time of O_{ik}^l on machine k at level l, where $i = 1, 2,\ldots, n$; $k = 1, 2,\ldots, m$; and $l = 1, 2,\ldots, L$.
C_{ij}^l	the completion time of J_{ij}^l on machine i at level l, where $i = 1, 2,\ldots, m$; $j = 1, 2,\ldots, n$; and $l = 1, 2,\ldots, L$.
$[j]$	the job scheduled in the jth position of a sequence.
$C_{i[j]}^l$	the completion time of J_{ij}^l scheduled in the jth position of a sequence on machine k at level l, where $i = 1, 2,\ldots, m$; $j = 1, 2,\ldots, n$; and $l = 1, 2,\ldots, L$
$C_{\max}$	the maximum completion time or makespan.

Notation for the MA

π, π_0, π_1, π_2	a sequence of n jobs
$\pi(j)$	the job arranged in position j of a schedule
$d(\pi_1, \pi_2)$	the distance between the solutions of π_1 and π_2
X, X', X_{new}	a population or schedule
$n(X)$	the size of population X

We formally state and formulate the RPFS scheduling problem to minimize the makespan as follows: There are n jobs to be processed on m machines (i.e., $M_1, M_2,\ldots, M_m$) where all the machines are available throughout the working period. All the jobs are available at time zero and preemption is not allowed. Technological constraints are known in advance. There is unlimited waiting space for jobs waiting to be processed. The defining characteristic of the re-entrant flowshop is that every job in the shop must be processed on the m machines in the following order: $M_1, M_2,\ldots, M_m$; $M_1, M_2,\ldots, M_m$;...; and $M_1, M_2,\ldots, M_m$, with L re-entrants (or levels), i.e., starting on machine M_1 and ending on M_m, and the job sequence is the same on any machine at each level. Furthermore, we assume that the machine order is the same for all the jobs and the job sequence is the same on each machine at each level. In addition, let O_{ij}^l be the operation of J_{ij}^l and p_{ij}^l be the processing time of O_{ij}^l, respectively. The objective is to minimize the makespan $C_{\max}$. Pan and Chen [8] extended the model of Wilson [20] by formulating the problem as a mixed binary integer program and solving it using the Lingo optimizer (for more details on the MIP model, please refer to [8]). However, they could solve problem instances with up to eight jobs only. Using the same model, we apply the CPLEX optimizer to generate feasible solutions and develop a memetic algorithm (MA) to obtain near-optimal solutions for the problem. For more details on the MIP model and technical computation, the reader may refer to Pan and Chen [8] and Chen [18].

A memetic algorithm

MA is a well-known population-based meta-heuristic, which can be used to tackle various combinatorial optimization problems, e.g. [21–24,30,31]. In this paper we propose an improved MA for the RPFS scheduling problem to minimize the makespan. We summarize the key steps of the MA as follows:

Proposed Memetic Algorithm

```
1:  Initialize population $X$ with $n(X)$ solutions using the method
    described in "Population initialization" section
2:  while the stopping criterion is not reached do
3:    Update the population using the method described in
    "Population update strategy" section
4:      (1) Generate a temporary population $X'$ with $n(X)$ solutions
    using the crossover and mutation operators based on $X$.
5:      (2) Generate a new population $X_{\text{new}}$ from $X \cup X'$.
        (3) Set $X = X_{\text{new}}$.
7:    Perform the dynamic local search described in "Dynamic local
    search" section to improve the best 10 solutions in the current
    population $X$.
8:    Update the best solution found so far.
9:    Remove the duplicated solutions in $X$ by performing a random
    insertion move (remove a random job from its current position
    and insert it to another position) to each duplicated solution.
10:   if (the diversity index of the population is less than a threshold
    value) then
11:     Perform the population restart strategy on $X$ according to the
    method described in "Population restart strategy" section
12:   end if
13: end while
14: Output the best solution found so far by the algorithm.
```

Solution representation

In the proposed MA, a solution (a schedule or sequence) is represented as a permutation of the n jobs, i.e., $\pi = (\pi(1), \pi(2),\ldots, \pi(n))$, where the jth number in the permutation $\pi(j)$ denotes the job arranged in position j of the schedule (sequence).

Population initialization

We generate the initial population X with $n(X)$ solutions by two kinds of method. We use the generalization of the insertion technique of Nawaz et al. [29] (which is called NEH) to generate the first solution because NEH is considered to be the best among the constructive heuristics for permutation flowshop scheduling problems [32]. In addition, we also adopt a random generation heuristic that iteratively assigns jobs to random positions to generate the other initial solutions to guarantee good diversity of the initial population.

Population diversity maintenance

In recent years, evolutionary algorithms such as genetic algorithm, particle swarm optimization, and differential evolution, have been widely used to deal with various kinds of combinatorial optimization problems. One major issue of using evolutionary