Expert Systems With Applications 79 (2017) 20–32

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Scatter search for mixed blocking flowshop scheduling

Vahid Riahi[a,*], Mostafa Khorramizadeh[b], M.A. Hakim Newton[a], Abdul Sattar[a]

[a] Institute for Integrated and Intelligent Systems (IIIS), Griffith University, 170 Kessels Rd, Nathan QLD 4111, Australia
[b] Department of Mathematical Sciences, Shiraz University of Technology, Shiraz, Iran

ARTICLE INFO

Article history:
Received 28 September 2016
Revised 15 February 2017
Accepted 16 February 2017
Available online 20 February 2017

Keywords:
Flowshop scheduling
Mixed blocking constraints
Heuristic algorithms
Meta-heuristic algorithms
Scatter search

ABSTRACT

Empty or limited storage capacities between machines introduce various types of blocking constraint in the industries with flowshop environment. While large applications demand flowshop scheduling with a mix of different types of blocking, research in this area mainly focuses on using only one kind of blocking in a given problem instance. In this paper, using makespan as a criterion, we study permutation flowshops with zero capacity buffers operating under mixed blocking conditions. We present a very effective scatter search (SS) algorithm for this. At the initialisation phase of SS, we use a modified version of the well-known Nawaz, Enscore and Ham (NEH) heuristic. For the improvement method in SS, we use an Iterated Local Search (ILS) algorithm that adopts a greedy job selection and a powerful NEH-based perturbation procedure. Moreover, in the reference set update phase of SS, with small probabilities, we accept worse solutions so as to increase the search diversity. On standard benchmark problems of varying sizes, our algorithm very significantly outperforms well-known existing algorithms in terms of both the solution quality and the computing time. Moreover, our algorithm has found new upper bounds for 314 out of 360 benchmark problem instances.

1. Introduction

Permutation flowshop scheduling has been a very active research area after the seminal work by Johnson (1954). It has its significance both in theory and industry (Pan et al., 2011; Riahi & Kazemi, 2016; Safari & Sadjadi, 2011; Shao & Pi, 2016). Permutation flowshops involve processing a number of jobs by a number of machines. The machines are in a fixed order with buffers of different sizes between successive machines. Each job needs processing at each machine and in the order of the machines. However, once an ordering of the jobs is determined, jobs are processed in the same order on all machines. The *permutation flowshop* problem is to find a permutation of the jobs such that a given objective (e.g. the makespan) is minimised.

Permutation flowshops could run with buffers having unlimited capacities. However, the buffer capacities in the real-world industries are limited or even zero. Flowshop scheduling problems with zero capacity buffers are called *blocking flowshop scheduling problems* (BFSP) (Lovner, 1969). In the literature, different types of blocking are introduced and the most traditional and common one is Release when Starting Blocking (RSb). In the RSb, a completed job has to stay (blocked) in the current machine until the next machine is available for processing (Ribas et al., 2015). This type of blocking can be observed in chemical industry as an example. In this industry, partially completed jobs sometimes stay in the machines due to lack of intermediate buffers. Several types of algorithm have been proposed to solve the BFSP. These include genetic algorithms (Caraffa et al., 2001), tabu search (Grabowski & Pempera, 2007), hybrid discrete differential evolution ()Wang, Pan, Suganthan et al., 2010), harmony search algorithms (Wang, Pan, & Tasgetiren, 2011), iterated greedy algorithms (Ribas et al., 2011), artificial immune system algorithms (Lin & Ying, 2013), and bee colony algorithms (Ribas et al., 2015).

Inspired by real-world industries, another model of blocking flowshop, named Release when Completing blocking (RCb), was introduced by Dauzère-Pérès et al. (2000). In RCb, a machine can start the next available job if the previous job not only does finish, but also leaves the next machine. RCb is seen in real-life industries such as waste treatment and aeronautics parts fabrication industries (Martinez de La Piedra, 2005). For more details, consider waste treatment industries as an example. There are two main steps named tanks and blender. Different types of waste (industrial and farm) when received are transferred and unloaded into tanks by trucks. Then, each cargo is treated by only one blender. In this industry, when wastes flow from the tank to the blender, the tank is not available until the product is finished completely in the blender (RCb constraint). To deal with RCb, an integer

* Corresponding author.
E-mail addresses: vahid.riahi@griffithuni.edu.au (V. Riahi), m.khorrami@sutech.ac.ir (M. Khorramizadeh), mahakim.newton@griffith.edu.au (M.A. Hakim Newton), a.sattar@griffith.edu.au (A. Sattar).

http://dx.doi.org/10.1016/j.eswa.2017.02.027
0957-4174/© 2017 Elsevier Ltd. All rights reserved.

linear programming (ILP) model along with a meta-heuristic algorithm was proposed by Martinez de La Piedra (2005) and also an electromagnetism-like (EM) algorithm was used by Yuan and Sauer (2007). Later, a hybridisation of the EM algorithm and a mixed-integer linear programming model appeared in Yuan et al. (2009). Lately, a new blocking constraint, Release when Completing Blocking* (RCb*), has been proposed in Trabelsi et al. (2010). RCb* is a special type of RCb where a machine cannot start processing an available job until the next machine finishes the previous job regardless of whether the previous job leaves the machine or not. RCb* is seen at production factories where two consecutive machines use the same resource (tools or operators) (Trabelsi et al., 2010).

Despite the work mentioned above on various types of blocking, research on using a number of different types of blocking in a mixed setting–hence called a *mixed blocking permutation flowshop scheduling problem* (MBPFSP)–is rare. This is the case even though large-scale production industries, due to technical limitation or scarcity of intermediate buffer space, often face different types of blocking simultaneously. We will later show an example of mixed blocking situation in detail in Section 2.4 on cider production process. Nevertheless, mixing of RSb and RCb are studied in Martinez et al. (2006). Later, mixing of RSb, RCb and RCb* are studied in Trabelsi et al. (2012) using genetic algorithms and then in Khorramizadeh and Riahi (2015) using Taguchi arrays and local search within a bee colony optimisation algorithm. Overall, the aforementioned population-based algorithms used for MBPFSP suffer from the lack of diversity in the exploration. As a result, the individual solutions in the population soon become very close to each other; which needs to be dealt with more carefully.

In this paper, we present a very effective scatter search algorithm for the MBPFSP. In our MBPFSP, we use different types of blocking that include RSb, RCb and RCb*, and we consider minimising makespan as an objective criterion. Scatter search algorithms (Glover, 1977) keep a reference set of solutions and combine them intelligently with each other to incorporate both quality and diversity, and finally to reach better solutions. Scatter search has been successfully applied in many scheduling problems. At the initialisation phase of our scatter search algorithm for MBPFSP, we use a modified version of the well-known Nawaz, Enscore and Ham (NEH) heuristic (Nawaz et al., 1983). We also use an iterated local search along with an NEH-based perturbation. In the local search, jobs are selected greedily based on two differential blocking times created by RCb and RCb* constraints. Moreover, we accept worse solutions with small probabilities to increase diversity of the exploration in the reference set updating step. The experimental and statistical results on standard benchmark problems show that the presented heuristic algorithm is better than the best-performing existing heuristic algorithms. Finally, our SS algorithm significantly outperforms the state-of-the-art MBPFSB algorithms on both solution quality and computing time. Moreover, we have obtained new upper bounds for 314 out of 360 standard benchmark instances of the MBPFSP.

The rest of the paper is organised as follows: Section 2 introduces the mixed blocking permutation flowshop scheduling problem; Section 3 details our scatter search algorithm for MBPFSP; Sections 4 and 5 provide our experimental results on different benchmark problems; Section 6 discusses the best solutions found by our algorithm; and finally, Section 7 presents our conclusions.

2. Mixed blocking permutation flowshop scheduling

In a permutation flowshop scheduling problem (shown in Fig. 1), there are m machines and n jobs. The machines are in a given fixed order. Without loss of generality, assume the machines are in the order $1, 2, \ldots, m$. There is a buffer i of a given capacity between each two successive machines i and $i+1$. Each job $1 \leq j \leq n$ needs processing at each machine. A machine $i+1$ can process a job j only after machine i has processed it. Once an ordering of the jobs is determined, jobs are processed in the same order on all machines. We are to find a permutation of the jobs such that all jobs one by one in the permutation order will pass through all machines and the objective (e.g. the makespan) will be minimised.

Assume a given ordering of the jobs where $[k]$ denotes the job at position $k \in [1, n]$. We use $S_{i,\,[k]}$, $C_{i,\,[k]}$ and $L_{i,\,[k]}$ to denote the starting time, completion time, and leave time of job $[k]$ at machine i. Machine i starts processing a job $[k]$ as soon as it arrives to the machine, and spends $P_{i,\,[k]}$ time to process the job. When the job $[k]$ is completed at machine i at $C_{i,\,[k]}$ time, it might be blocked for a while depending on the blocking constraint and would leave the machine i at time $L_{i,\,[k]}$, which is also its arrival and starting time $S_{i+1,[k]}$ at machine $i+1$ (for $i < m$). From these, we clearly get three relations $S_{i,[k]} + P_{i,[k]} = C_{i,[k]}$, $C_{i,\,[k]} \leq L_{i,\,[k]}$, and $L_{i,[k]} = S_{i+1,[k]}$ where $i < m$. Then, combining the three relations, we obtain another relation $S_{i+1,[k]} > S_{i,[k]}$, for $i < m$. Moreover, we have the relation $S_{i,[k]} \geq C_{i,[k-1]} > S_{i,[k-1]}$ for $k > 1$, since machine i always starts processing job $[k]$ after processing job $[k-1]$.

2.1. Mixed blocking

In this paper, we study the mixed blocking flowshop scheduling, where the buffer capacities are either zero (blocking) or unlimited (no-blocking). Three different types of blocking are found in the literature. These blocking types and also the no-blocking type are formally described below.

1. *Wb* (*Without Blocking*): a machine i can immediately start processing an available job $[k]$ after finishing job $[k-1]$.
2. *RSb* (*Release when Starting Blocking*): a machine $i < m$ cannot start processing an available job $[k]$ until machine $i+1$ starts processing job $[k-1]$.

   $S_{i,[k]} \geq S_{i+1,[k-1]}$
3. *RCb** (*Release when Completing Blocking**): a machine $i < m$ cannot start processing an available job $[k]$ until machine $i+1$ finishes job $[k-1]$.

   $S_{i,[k]} \geq C_{i+1,[k-1]}$
4. *RCb* (*Release when Completing Blocking*): a machine $i < m$ cannot start an available job $[k]$ until job $[k-1]$ leaves machine $i+1$.

   $S_{i,[k]} \geq L_{i+1,[k-1]}$

One important issue with the MBPFSP is the order of the blocking types to be applied on successive machines. In Trabelsi et al. (2012), a repeated sequence of (*RCb, RSb, RCb**, *Wb*) is used for ordering the blocking types of successive machines. This order is also used exactly by Khorramizadeh and Riahi (2015). However, there is an important drawback of this blocking sequence and standard benchmark MBPFSP problems should take the drawback into account. The RSb constraint applied to a machine that is immediately after another machine running under the RCb constraint cannot make any difference in the makespan. The following lemma proves this.

Lemma 1. *Applying RSb to machine* $i+1$ *does not create any blocking and has no impact on the makespan, if RCb is applied at machine* i.

Proof. When RCb is applied at machine i, we have $S_{i,[k]} \geq L_{i+1,[k-1]}$. Since $L_{i+1,[k-1]} = S_{i+2,[k-1]}$, we can write $S_{i,[k]} \geq S_{i+2,[k-1]}$. We also have $S_{i+1,[k]} > S_{i,[k]}$. So combining these relations, we obtain $S_{i+1,[k]} > S_{i+2,[k-1]}$, which always satisfies the condition $S_{i+1,[k]} \geq S_{i+2,[k-1]}$ for RSb at machine $i+1$. □