Contents lists available at ScienceDirect

Computers & Operations Research

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

A hybrid variable neighborhood search algorithm for solving the limited-buffer permutation flow shop scheduling problem with the makespan criterion

Ghasem Moslehi *, Danial Khorasanian

Department of Industrial and Systems Engineering, Isfahan University of Technology, Isfahan 84156-83111, Iran

ARTICLE INFO

Available online 5 October 2013

Keywords:
Scheduling
Limited-buffer permutation flow shop
Makespan
Variable neighborhood search
Simulated annealing algorithm
Blocking flow shop

ABSTRACT

This paper investigates the limited-buffer permutation flow shop scheduling problem (LBPFSP) with the makespan criterion. A hybrid variable neighborhood search (HVNS) algorithm hybridized with the simulated annealing algorithm is used to solve the problem. A method is also developed to decrease the computational effort needed to implement different types of local search approaches used in the HVNS algorithm. Computational results show the higher efficiency of the HVNS algorithm as compared with the state-of-the-art algorithms. In addition, the HVNS algorithm is competitive with the algorithms proposed in the literature for solving the blocking flow shop scheduling problem (i.e., LBPFSP with zero-capacity buffers), and finds 54 new upper bounds for the Taillard's benchmark instances.

1. Introduction

The flow shop scheduling problem is one of the most popular cases of machine scheduling problems. Unlike the general assumptions of the problem, its intermediate buffers may have limited capacity due to either technological considerations or process characteristics [1]. The advents of just-in-time manufacturing and Kanban systems which maintain a limited in-process inventory have attracted the attention of most researchers to the limited-buffer permutation flow shop scheduling problem (LBPFSP) [2]. In the case of LBPFSP with non-zero capacity buffers, a completed job on a machine may block it until the next intermediate buffer has a free space. Also, in the case with zero capacity buffers, called the 'blocking flow shop scheduling problem' (BFSP), a completed job may block the machine until the next downstream machine is free. The process of no jobs can be started on a machine, until a job blocks it. Hall and Sriskandarajah [1] presented a good review of the studies and applications of LBPFSPs.

In this paper, the LBPFSP with the makespan criterion is investigated. Papadimitriou and Kanellakis [3] proved the strong NP-hardness of the two-machine LBPFSP with a one-capacity buffer and the makespan criterion. Also, Hall and Sriskandarajah [1] showed that the three-machine BFSP with the makespan criterion is strongly NP-hard.

Due to the high complexity of the LBPFSP with the makespan criterion, attempts at its solution have invested more on the use of heuristics than other methods. Leisten [4] compared some constructive heuristics to arrive at the conclusion that the well-known Nawaz–Enscore–Ham (NEH) heuristic outperforms all others. Smutnicki [5] developed a tabu search (TS) algorithm for the two-machine case using H_block and S_block properties. The use of H_block and S_block properties may accelerate the local search by initially eliminating the neighbors that do not improve the current solution. Later, Nowicki [6] generalized Smutnicki's algorithm [5] to the m-machine case. Wang et al. [7] presented a hybrid genetic algorithm (HGA) and showed its superiority over the TS algorithm. Further, Liu et al. [2] developed a hybrid particle swarm optimization (HPSO) algorithm which obtained better solutions for most benchmark instances than HGA did. Recently, Pan et al. [8] proposed a chaotic harmony search (CHS) algorithm and obtained better results compared to HGA or HPSO algorithms. Also, Qian [9] presented a hybrid differential evolution (HDE) algorithm for the multi-objective LBPFSP.

A greater number of approaches have been proposed for solving the BFSP with the makespan criterion than those developed for solving the general case of the problem, i.e. the LBPFSP with the makespan criterion. Except for the branch and bound algorithm proposed by Ronconi [10] and that by Companys and Mateo [11], almost all other approaches are inexact. McCormick et al. [12] proposed a profile fitting (PF) heuristic. Later, Ronconi [13] presented a minmax (MM) heuristic, a combination of NEH and MM (MME), and a combination of NEH and PF (PFE). Computational results showed the superiority of MME and PFE over NEH. Caraffa et al. [14]

* Corresponding author. Tel.: +98 311 391 5522; fax: +98 31 139 15 526.
E-mail addresses: moslehi@cc.iut.ac.ir, moslehi@istt.ir (G. Moslehi), d.khorasanian@in.iut.ac.ir (D. Khorasanian).

0305-0548/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2013.09.014

presented a genetic algorithm (GA) for this problem while Grabowski and Pempera [15] developed a TS algorithm and one with multi-moves (TS+M), both of which outperformed the GA. Wang et al. [16] developed a hybrid discrete differential evolution (HDDE) algorithm that outperformed the TS and TS+M proposed by Grabowski and Pempera [15]. Ribas et al. [17] presented an iterated greedy (IG) algorithm that outperformed the HDDE algorithm. Recently, Wang et al. [18] have proposed a hybrid modified global-best harmony search (hmgHS) algorithm which outperforms the IG algorithm. For an initial solution generation, they used the NEH–Wang–Pan–Tasgetiren (NEH–WPT) heuristic which is a modified version of the NEH and showed its superior performance over NEH. In another study, Wang et al. [19] developed a three-phase algorithm (TPA) which outperformed the IG algorithm with lower CPU-times. Finally, Lin and Ying [20] proposed a revised artificial immune system (RAIS) algorithm that outperformed the IG algorithm. No comparison has been made among the hmgHS, TPA, and RAIS algorithms in the past studies.

This paper aims to solve the LBPFSP with the makespan criterion using a variable neighborhood search (VNS) algorithm hybridized with the simulated annealing (SA) algorithm. The VNS algorithm, a metaheuristic initially proposed by Mladenović and Hansen [21], draws upon the idea of systematically changing the neighborhood structure. The algorithm and its variants have been successfully used to solve some combinatorial optimization problems [22–24], among others. According to this algorithm, first an initial solution is generated. In a second stage, the three steps of 'shaking', 'local search', and 'neighborhood change' are repeated in this order until a stopping criterion is met. Algorithm 1 illustrates these steps. In the shaking step, a neighbor x' is generated for the current solution (x) using the k-th pre-defined neighborhood structure (N_k) where $k \in \{1, \ldots, k_{max}\}$. In Line 6, N_k (x) represents a set including all neighbors of x generated by N_k. After the shaking step, a local search is performed on x'. Finally, in the neighborhood change step, if the solution obtained from the local search step (x'') is better than x, it will be considered as the new current solution, and k will be equal to 1; otherwise, $k=k+1$. The hybrid VNS algorithm (HVNS), presented in Section 4, uses the idea of the SA algorithm in the local search step.

Algorithm 1. VNS algorithm

```
1    Generate an initial solution $x$;
2    **Repeat**
3        $k=1$;
4        **Repeat**
5            %Shaking
6            Generate a neighbor $x' \in N_k(x)$;
7            %Local search
8            Obtain $x''$ by a local search for $x'$;
9            %Neighborhood change
10           **If** $x''$ is better than $x$
11               $x=x''$; $k=1$;
12           **Else** $k=k+1$;
13           **Endif**
14       **Until** $k=k_{max}$
15   **Until** stopping criterion is met
```

The rest of the paper is organized as follows. Following this Introduction, the LBPFSP with the makespan criterion is formulated in Section 2. Section 3 provides the provisions needed for speeding up the local search approaches. Section 4 develops the HVNS algorithm for the problem. Algorithm settings and computational results are presented in Section 5. Finally, conclusions and suggestions for future studies are presented in Section 6.

2. Problem formulation

In LBPFSP with the makespan criterion, there are n jobs that should be sequentially processed on a series of machines m_1, m_2, …, and m_m. The operation $O_{i,r}$ corresponds to the processing of job r, $r \in \{1, \ldots, n\}$, on machine i, $\{i=1,\ldots, m\}$, the processing time of which is equal to $p_{i,r}$. Between each two consecutive machines i and $i+1$, $i=1, \ldots, m-1$, there exists a buffer with the size equal to $B_i \geq 0$. The jobs obey the first-in-first-out (FIFO) rule in the buffers, and the sequence in which the jobs are processed on each machine is consequently identical. In the case with non-zero size buffers, a completed job on a machine may block it until a free space is available in the intermediate buffer. Also, in BFSP, the completed job on the machine may block it until the next downstream machine is free. At any time, each machine can process at most one job, and each job can be processed on at most one machine. The release time of each job is equal to zero, and the set-up times are included in the processing times. Also, each job is processed without preemption on each machine. The objective of the problem is to find a sequence that minimizes the makespan.

Let us consider the sequence $\pi=(\pi(1), \pi(2), \ldots, \pi(n))$ where $\pi(j)$ represents the j-th job in the sequence, further, define $S_{i,j}(\pi)$, $i=1, \ldots, m$, $j=1, \ldots, n$, as the earliest starting time of job $\pi(j)$ on machine i calculated from the following recursive formula [2]:

$$S_{i,j}(\pi) = \max(S_{i,j-1}(\pi)+p_{i,\pi(j-1)},\ S_{i-1,j}(\pi)+p_{i-1,\pi(j)},\ S_{i+1,j-B_i-1}(\pi)),\quad i=1,\ldots,m,\ j=1,\ldots,n \tag{1}$$

where, $p_{i,\pi(j-1)}=0$ and $S_{i,j-1}(\pi)=0$ for $j=1$, $p_{i-1,\pi(j)}=0$ and $S_{i-1,j}(\pi)=0$ for $i=1$, and $S_{i+1,j-B_i-1}(\pi)=0$ for $i=m$ or $j \leq B_i+1$. The makespan of the sequence π designated by $C_{max}(\pi)$ is equal to $S_{m,n}(\pi)+p_{m,\pi(n)}$.

3. Preparations to speed up local search approaches

In LBPFSP, the precedence relationships existing between the operations of jobs in a given sequence can be depicted in a directed graph. In the graph for the sequence $\pi=(\pi(1), \pi(2), \ldots, \pi(n))$, there are $m \times n$ nodes, where the node (i, j), $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ represents the operation $O_{i,\pi(j)}$ and has a weight equal to $p_{i,\pi(j)}$. The graph has a horizontal arc between the nodes (i, j) and $(i,j+1)$, $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n-1\}$, a vertical arc between the nodes (i, j) and $(i+1, j)$, $i \in \{1, \ldots, m-1\}$, $j \in \{1, \ldots, n\}$, and a skew arc between the nodes (i, j) and $(i-1, j+B_{i-1}+1)$, $i \in \{2, \ldots, m\}$, $j \in \{1, \ldots, n-B_{i-1}-1\}$. The skew arc between the nodes (i, j) and $(i-1, j+B_{i-1}+1)$, $i \in \{2, \ldots, m\}$, $j \in \{1, \ldots, n-B_{i-1}-1\}$ has a weight equal to $(-p_{i,\pi(j)})$ while those of the other arcs are equal to zero. The length of the longest path in the graph is equal to $C_{max}(\pi)$. The longest path, called the 'critical path', contains some nodes with zero floating times, called 'critical nodes'. The values for the earliest and latest starting times are the same for each critical node [25]. Fig. 1 shows the graph of a sequence for a problem with $n=7$, $m=4$, $B_1=B_3=1$, and $B_2=0$. The weight of each node is placed on it. Also, the critical nodes are shown in dark and bold typeface.

Define $T_{i,j}(\pi)$, $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$ as the duration between the latest loading time of $\pi(j)$ on machine i and $C_{max}(\pi)$, calculated as follows based on the graph:

$$T_{i,j}(\pi) = \max(T_{i+1,j}(\pi)+p_{i,\pi(j)},\ T_{i,j+1}(\pi)+p_{i,\pi(j)},\ T_{i-1,j+B_{i-1}+1}(\pi)),\quad i=1,\ldots,m,\ j=1,\ldots,n \tag{2}$$

where, $T_{i+1,j}(\pi)=0$ for $i=m$, $T_{i,j+1}(\pi)=0$ for $j=n$, and $T_{i-1,j+B_{i-1}+1}(\pi)=0$ for $i=1$ or $j>n - B_{i-1}$ 1. It is straightforward