Computers and Operations Research 100 (2018) 230–243

Contents lists available at ScienceDirect

Computers and Operations Research

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

Fast heuristics for minimizing the makespan in non-permutation flow shops

Alexander J. Benavides [a], Marcus Ritt [b,*]

[a] Escuela Profesional de Ciencia de la Computación, Universidad Nacional de San Agustín de Arequipa, Av. Venezuela S/N Área de Ingenierías, Arequipa, Peru
[b] Instituto de Informática, Universidade Federal do Rio Grande do Sul, Av. Bento Gonalves, 9500, Porto Alegre 91501-970, Brazil

ARTICLE INFO

Article history:
Received 12 March 2018
Revised 9 July 2018
Accepted 18 July 2018
Available online 20 July 2018

Keywords:
Scheduling
Metaheuristics
Flow shop
Non-permutation schedules
Makespan
Iterated greedy algorithm

ABSTRACT

We introduce a new permutation representation for non-permutation schedules, and show how the acceleration technique of Taillard can be extended to it. We propose three new heuristics for the non-permutation flow shop scheduling problem with makespan minimization: a constructive heuristic with the same time complexity as the well-known NEH heuristic, a non-permutation insertion local search with a time complexity $O(n^2m)$ for evaluating a neighbourhood on n jobs and m machines, the same as for the permutation insertion local search, and a reduced-neighbourhood best-improvement non-permutation local search with a time complexity of $O(nm)$ per neighbourhood. These heuristics are combined into iterated greedy algorithms for the permutation and non-permutation flow shop scheduling problem. In extensive computational experiments we find that our iterated greedy algorithms produce better non-permutation schedules than other methods for the permutation and non-permutation flow shop scheduling problem, in the same computational time.

1. Introduction

Flow shop scheduling is a very common manufacturing layout, with a wide range of applications in production and other areas, e.g. in database applications (Allahverdi and Al-Anzi, 2002) or parallel computing (Sahni, 1995). Flow shop scheduling is also the basis for scheduling robotic cell flowshops, which are widely used in modern manufacturing systems (Dawande et al., 2007). In a flow shop, a collection of jobs $J_1, \ldots, J_n$ must be processed on machines $M_1, \ldots, M_m$ following the same fixed machine sequence. The processing of job J_j on machine M_i is called the *operation* o_{ij}, and this operation must be processed without interruption for a time p_{ij}. At any given instant, each job can be processed on at most one machine, and no machine can process more than one job. A *schedule* is an allocation of the jobs to the machines over time. The flow shop scheduling problem (FSSP) consists in finding a schedule optimizing some objective function. Minimizing the makespan is the most common optimization criterion in the literature, and the resulting problem is NP-complete. The makespan is defined as the maximum completion time, i.e. the completion time of the latest job. Table 1 shows the processing times of an instance of the FSSP with six jobs and six machines.

A *permutation schedule* processes the jobs in the same order on all machines. For this reason it can be represented by a single permutation of the jobs. A *non-permutation schedule* may have a different processing order of jobs for some machines. There are $n!^{\max\{m-2,1\}}$ candidates for the optimal schedule, since there always exists an optimal schedule with the same job sequence on the first two and on the last two machines (Conway et al., 1967). Fig. 1 shows the best possible permutation schedule for the instance given in Table 1, of makespan 43. The best non-permutation schedule shown in Fig. 2 has makespan 40. Note that the order of J_6 and J_4 changes between M_3 and M_4.

Since the permutation flow shop scheduling problem (PFSSP) is a simplification of the FSSP that only considers $n!$ candidate permutations, it can lead to inferior solutions. Non-permutation schedules can be up to a factor $O(\sqrt{m})$ shorter than permutation schedules, and are often significantly shorter in practice (Liao et al., 2006; Potts et al., 1991; Tandon et al., 1991). Nevertheless, almost all the literature focuses on the PFSSP since it is easier to solve.

In this paper we propose efficient heuristics for finding permutation and non-permutation schedules of short makespan. The paper makes four main contributions. First, we introduce a new representation of non-permutation schedules with split jobs, which greatly simplifies manipulation of and reasoning about non-permutation schedules. Second, we systematically explore local improvement after each insertion in constructive heuristics. Third, we introduce a new local search for non-permutation schedules based

* Corresponding author.
E-mail addresses: ajbenavides@unsa.edu.pe (A.J. Benavides), marcus.ritt@inf.ufrgs.br (M. Ritt).

https://doi.org/10.1016/j.cor.2018.07.017
0305-0548/

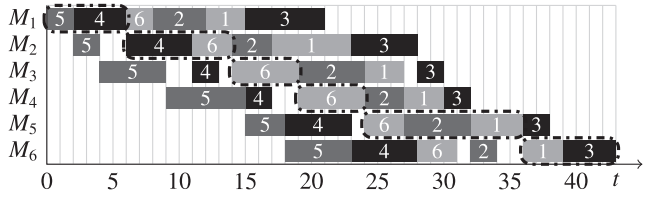

Fig. 1. Gantt chart of the optimal permutation schedule $\pi = (J_5, J_4, J_6, J_2, J_1, J_3)$ for the 6×6 FSSP instance given in Table 1. Blocks of operations on the critical path are drawn dashed.

Table 1
A 6×6 instance of the FSSP.

	Machine					
Job	M_1	M_2	M_3	M_4	M_5	M_6
J_1	3	6	3	3	4	3
J_2	4	3	5	3	5	2
J_3	6	5	2	2	2	4
J_4	4	5	2	2	5	5
J_5	2	2	5	6	3	5
J_6	2	3	5	5	3	3

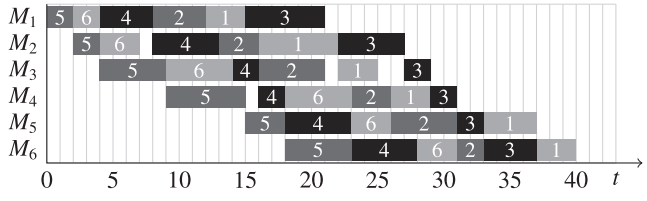

Fig. 2. Gantt chart of the optimal non-permutation schedule for the 6×6 FSSP instance given in Table 1 (Jobs J_1 and J_6 are divided by jobs J_3 and J_4 respectively).

on swapping the order of adjacent jobs, and introduce acceleration techniques to make it efficient. Finally, we provide a comprehensive experimental evaluation, which includes the large, hard instances of Vallada et al. (2015).

In the remainder of this section we review the literature on both problems. We introduce the new representation of non-permutation schedules in Section 2, introduce new constructive heuristics in Section 3, new local search methods in Section 4, an iterated greedy algorithm in Section 5. Acceleration techniques to implement these methods efficiently are explained in Section 6. In Section 7 we present and discuss the results of extensive computational tests and comparisons to existing methods. In Section 8 we offer some concluding remarks.

1.1. Literature review on the PFSSP

The constructive heuristic NEH (Nawaz et al., 1983; Taillard, 1990) is considered to be the best heuristic for the PFSSP (Dong et al., 2008; FarahmandRad et al., 2009; Fernandez-Viagas and Framinan, 2014; Kalczynski and Kamburowski, 2009; Rossi et al., 2016; Vasiljevic and Danilovic, 2015). A pseudo-code of NEH is given in Algorithm 1. First, the jobs are ordered by non-increasing total processing time to create a priority order $\pi_o = (\pi_o(1), \ldots, \pi_o(n))$, and an initial partial solution $\pi = (\pi_o(1))$ containing only the first job is created. Then, NEH inserts the next job $\pi_o(l)$ into the current partial solution π at the position that minimizes its makespan. This is repeated until π is a complete solution. After each insertion, an improvement method may be applied to the partial schedule (line 6). The improvement is not part of the original NEH heuristic. The heuristic processes n jobs, line 4 needs to evaluate the makespan for $O(n)$ possible insertion positions for each job, and calculating the makespan of a schedule needs time $O(nm)$. Thus, the time complexity of NEH is $O(n^3m)$.

Algorithm 1 Constructive heuristic NEH for the PFSSP.

Input: The processing times p_{ij} for each job J_j on each machine M_m.
Output: A permutation schedule π.
1: $\pi_o :=$ jobs ordered by non-increasing total processing time
2: $\pi := (\pi_o(1))$
3: **for** $l \in [2, n]$ **do**
4: Evaluate all possible insertions of job $\pi_o(l)$ into π
5: Insert job $\pi_o(l)$ into π at the best position
6: Optionally, improve the partial solution
7: **end for**
8: **return** π

Taillard (1990) has shown that the makespan of all $O(n)$ possible insertions in line 4 can be computed in time $O(nm)$, reducing the time of the NEH heuristic to $O(n^2m)$. To see this, assume that job J_l is going to be inserted into the partial permutation schedule $\pi = (\pi(1), \ldots, \pi(n'))$ to produce the permutation $\pi' = (\pi'(1), \ldots, \pi'(n'+1))$. The makespan values $MC_1, \ldots, MC_{k+1}$ after inserting job J_l at position $k \in [n'+1]$ are calculated as follows. The *head* or earliest completion time e_{ik} of job $\pi(k) = J_j$ on machine M_i is

$$e_{ik} = \max\{e_{i,k-1}, e_{i-1,k}\} + p_{ij}, \qquad i \in [m], k \in [n'], \tag{1}$$

where $e_{0,k} = e_{i,0} = 0$. Similarly, the *tail* q_{ik}, i.e. the time between the end of processing and the latest possible starting time of each job $\pi(k) = J_j$ on each machine i is

$$q_{ik} = \max\{q_{i,k+1}, q_{i+1,k}\} + p_{ij}, \qquad i \in [m], k \in [n'], \tag{2}$$

where $q_{m+1,k} = q_{i,k+1} = 0$. The earliest relative completion time e'_{ik} of job J_l when inserted at position k on each machine M_i then is

$$e'_{ik} = \max\{e'_{i-1,k}, e_{i,k-1}\} + p_{il}, \qquad i \in [m], k \in [n'+1], \tag{3}$$

where $e'_{0,k} = 0$. The earliest completion time $e_{i,k-1}$ before the insertion position k remains unchanged after the insertion of job J_l, and thus can be used to find e'_{ik}. Finally, the makespan MC_k of the permutation schedule π' after the insertion of job J_l at position $k \in [n'+1]$ is

$$MC_k = \max_{i \in [m]} \{e'_{ik} + q_{ik}\}. \tag{4}$$

(The notation used here and in the remainder of the paper is summarized in Table 2.)

Many researchers have proposed improvements of the NEH heuristic (Dong et al., 2008; Fernandez-Viagas and Framinan, 2014; Kalczynski and Kamburowski, 2009; Vasiljevic and Danilovic, 2015). The improvements add a very small computational cost by evaluating different initial priority orders, and by breaking ties in the initial priority order and in the selection of the insertion position. The current best constructive heuristic with complexity $O(n^2m)$ was proposed by Fernandez-Viagas and Framinan (2014). They report an average relative deviation (ARD) of 2.897% from the best known values for the benchmark proposed by Taillard (1993).

Other researchers have proposed an improvement phase after each insertion (line 6 in Algorithm 1). FarahmandRad et al. (2009) proposed NEH-based constructive methods that evaluate the relocation of a previously inserted job. Their FRB3 heuristic with complexity $O(n^3m)$ tries to relocate each job and presents an ARD of 1.63%, and their FRB4 heuristic with complexity $O(kn^2m)$ relocates only k jobs around the last inserted job and presents an ARD of 1.87% for $k = 12$. Their most expensive heuristic, FRB5, performs an insertion local search after each new job insertion, and leads to an ARD of 1.44%. Rossi et al. (2016) proposed a similar approach of complexity $O(n^3m)$. Their improvement phase evaluates the reinsertion of pairs of jobs. They