Computers & Operations Research 66 (2016) 160–169

Contents lists available at ScienceDirect

Computers & Operations Research

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

Two simple and effective heuristics for minimizing the makespan in non-permutation flow shops

Alexander J. Benavides, Marcus Ritt *

Instituto de Informática, Universidade Federal do Rio Grande do Sul, Brazil

ARTICLE INFO

Available online 13 August 2015

Keywords:
Scheduling
Metaheuristics
Flow shop
Non-permutation schedules

ABSTRACT

We propose a constructive and an iterated local search heuristic for minimizing the makespan in the non-permutation flow shop scheduling problem. Both heuristics are based on the observation that optimal non-permutation schedules often exhibit a permutation structure with a few local job inversions. In computational experiments we compare our heuristics to the best heuristics for finding non-permutation and permutation flow shop schedules, and evaluate the reduction in makespan and buffer size that can be achieved by non-permutation schedules.

1. Introduction

In the flow shop scheduling problem (FSSP) we have to find a schedule for jobs $J_1,\ldots,J_n$ on machines $M_1,\ldots,M_m$. Each job J_j must be processed on all machines in the given order. On each machine M_i it must be processed without interruption for a time p_{ij}, and can be processed on at most one machine at a given instant. Likewise, each machine can process at most one job at a given instant. The *makespan* of a given schedule is the completion time of the last job on the last machine, and its minimization is the most common objective function in the FSSP, and also our concern here. This problem is also denoted by $F||C_{\max}$ and is NP-hard for $\min\{n,m\}\geq 3$.

Formally, let x_{ij} be the starting time of job J_j on machine M_i and let variable $y_{ijj'}\in\{0,1\}$ indicate that job J_j precedes job $J_{j'}$ on machine M_i, then an integer linear program for the FSSP is

$$\textbf{min}\quad C_{\max}, \tag{1}$$

$$\textbf{s.t.}\quad x_{mj}+p_{mj}\leq C_{\max}, \qquad \forall j\in[n], \tag{2}$$

$$x_{ij}+p_{ij}\leq x_{i+1,j}, \qquad \forall i\in[m-1],\ j\in[n], \tag{3}$$

$$x_{ij}+p_{ij}\leq x_{ij'}+M(1-y_{ijj'}), \qquad \forall i\in[m],\ j\neq j'\in[n], \tag{4}$$

$$y_{ijj'}+y_{ij'j}=1, \qquad \forall i\in[m],\ j\neq j'\in[n], \tag{5}$$

$$x_{ij}\geq 0, \qquad \forall i\in[m],\ j\in[n], \tag{6}$$

$$y_{ijj'}\in\{0,1\}, \qquad \forall i\in[m],\ j\neq j'\in[n]. \tag{7}$$

In this formulation constraints (2) define the makespan, constraints (3) require a job to terminate on a machine before starting on the following one, constraints (4) guarantees that the jobs are processed in the order defined by the variables y, and constraint (5) forces a linear ordering of the jobs on all machines. The constant M has to be sufficiently large, e.g. $M=\sum_{i\in[m]}\sum_{j\in[n]}p_{ij}$.

Johnson [12] has shown that there always exists an optimal schedule such that the processing order of the jobs on the first two and the last two machines is the same. Thus, the flow shop scheduling problem (FSSP) has up to $(n!)^{max\{m-2,1\}}$ different candidate schedules. This may have contributed to the fact that most of the literature focuses on the permutation flow shop scheduling (PFSSP), which permits only the $n!$ schedules which have an identical job order on all machines. Another reason for preferring the PFSSP may be that studies show that the gain of non-permutation schedules is limited, so that the increased problem complexity and the corresponding increased solution time seems not to justify the improvement in the makespan [20,28,40].

However, a large fraction of the search space of the FSSP usually is uninteresting. It is, for example, very unlikely that the schedules on two subsequent machines are reversed. We can observe in practice that good or optimal solutions of the FSSP exhibit only a few "strategical" inversions of jobs. For example, Fig. 1 shows Gantt charts of the optimal permutation and non-permutation schedules of an instance of the FSSP. Observe that the non-permutation schedule has almost a permutation structure with a few inversions, namely the 4th and the 5th job on the last two machines and the 13th and the 14th job on the last two machines. Thus it seems to be reasonable to limit heuristic searches to such cases. The main contribution of this paper is to propose a constructive and a search heuristic based on this observation.

* Corresponding author. Tel.: +55 5133086818; fax: +55 5133087308.
E-mail addresses: ajbenavides@inf.ufrgs.br (A.J. Benavides), marcus.ritt@inf.ufrgs.br (M. Ritt).

http://dx.doi.org/10.1016/j.cor.2015.08.001
0305-0548/

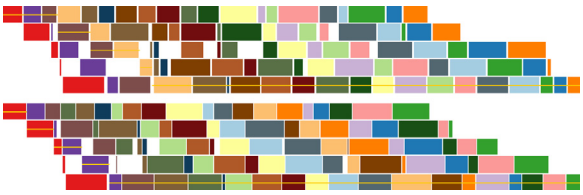

Fig. 1. Gantt chart of an optimal permutation schedule (above, makespan 1359) and an optimal non-permutation schedule (below, makespan 1358) for instance ta002.

Current state-of-the-art heuristics for the PFSSP produce nearly optimal solutions. For the instances proposed by Taillard [38], for example, such solutions deviate by about 0.5% from the best known values [7,33,35,45]. The best known values, in turn, are mostly optimal, and deviate in average by 0.2% from the best known lower bounds [39]. This limits the possible improvement of new heuristics for the PFSSP.

The worst case gap between permutation and non-permutation schedules is $\Theta(\sqrt{m})$ [22,27]. For practical instances, studies of Tandon et al. [40] and Liao et al. [20] show that the makespan of non-permutation schedules is about 1–3% shorter compared to permutation schedules. Given the above limits for improving permutation schedules, a gain of 1–3%, within reasonable time, can be interesting. Observe also that permutation schedules already can require a buffer space of up to $n-1$ items between the machines, such that a practical implementation of non-permutation schedules does not increase the worst-case buffer space.

We explore the non-permutation FSSP in the remainder of this paper. In the next subsection we discuss related work. In Sections 2 and 3 we propose a constructive and a search heuristic based on the idea of limited "strategical" job inversions. Results of three experiments with these heuristics are reported in Section 4, and we conclude in Section 5.

1.1. Related work

Most of the literature focuses on the permutation flow shop. Since some of its solutions techniques can be successfully applied to produce non-permutation schedules as well, we first give a very brief overview over methods for solving the PFSSP, and then present heuristics for the general case.

The best constructive heuristics for the PFSSP are based on repeated insertion of jobs into a partial schedule at the position which maintains the makespan shortest. In their seminal paper Nawaz et al. [23] propose the constructive heuristic NEH which inserts the jobs in order of non-increasing total processing time. Since then, NEH-like algorithms have been studied intensively [6,7,13–16] and are currently the best constructive heuristics.

Various meta-heuristics have been applied to the PFSSP, including simulated annealing [25], genetic algorithms [30,33], ant colony optimization [29], particle swarm optimization [41], variable neighborhood search [45], and tabu search [10,24]. Among the best performing heuristics, iterated greedy algorithms stand out for their simplicity and performance [7,26,35]. The currently best performing method is the iterated greedy algorithm proposed by Fernandez-Viagas and Framinan [7].

Several authors have proposed constructive heuristics for the non-permutation case. Krone and Steiglitz [18] propose a two-phase heuristic. In the first phase they improve a random permutation schedule by a first improvement local search in a shift neighborhood that reinserts a job in an earlier position. The second phase optimizes the schedule by allowing job-passing. For a fixed "machine tail" k, a swap of adjacent jobs on machines $k, k+1, \dots, m$ which reduces the makespan is accepted. If no such swap exists, k is increased, and the process repeats, until k exceeds m.

Gonzalez and Sahni [9] obtain a $\lceil m/2 \rceil$-approximation in time $O(mn \log n)$ by solving two-machine flow shop problems on subsequent machine pairs optimally and joining the obtained schedules. Their algorithm has been improved by Chen et al. [3] to an $m/2+1/6$-approximation.

Leisten [19] is concerned with limited buffer space and no-wait flow shops. He proposes two heuristics BFPS and BFPSE for generating permutation schedules for the two-machine problem that optimize the buffer usage, from the start, and the start and end, respectively. To optimize the buffer usage, the next job is chosen such that it is the difference between its completion on the first machine and the completion of its $b+1$th predecessor on the second machine, for buffer size b, is minimized. If the buffer is infinite, this is equivalent of scheduling by earliest completion time for BFPS or the smaller of earliest completion time and latest starting time for BFPSE. He extends these heuristics to BFS and BFSE, which generate non-permutation schedules trying to optimize buffer usage.

Tandon et al. [40] propose a generalized RAES heuristic: it uses the RAES heuristic of Dannenbring [4] to produce a permutation schedule, and then applies a local search swapping adjacent jobs of an initial solution where all machines have the same permutation.

Koulamas [17] presents a constructive heuristic for the FSSP. In computational experiments he finds it to be competitive to NEH, and shows that it can outperform NEH when non-permutation schedules are optimal (later studies, however, do not confirm this on a larger set of instances, see Ruiz and Maroto [34]). To generate a permutation schedule, he solves all $\binom{m}{2}$ two-machine problems using the method of Johnson [12]. For each schedule, whenever a job J_i precedes a job J_j (not necessarily immediately), then priority of J_i is decreased by one, and whenever it succeeds a job J_j, its priority is increased by one. Equivalently, for each schedule, a job at rank r contributes $2r-(n+1)$ to its overall priority.

Pugazhendhi et al. [28] propose a heuristic to improve a permutation schedule, with their main interest in schedules with missing operations. The heuristic processes the jobs in a given permutation order, and inserts each job on each machine into an idle interval without increasing the completion time of any other job, if possible, and otherwise appends it at the end. The heuristic runs in time $O(n^2m)$.

The best results for the FSSP have been obtained by the application of meta-heuristics. Yin [43] proposes an iterated greedy algorithm (IGA). It represents a non-permutation schedule by a permutation π_1 for the first two machines, a permutation π_2 for machines $3, \dots, m-2$, and a permutation π_3 for the last two machines. The heuristic has three phases: the first improves π_1, letting $\pi_2 = \pi_3 = \pi_1$, the second fixes π_1 and improves π_2, letting $\pi_3 = \pi_2$, and the last fixes π_1 and π_2 and improves π_3. Each improvement phase applies an IGA to the permutation, removing $d=6$ random jobs and reinserting them at the best position. Lin and Ying [21] use the same representation in a hybridization of tabu search and simulated annealing. The tabu search generates a random neighbor by swapping two jobs in some permutation and accepts the best non-tabu neighbor according to a Metropolis criterion, with a temperature that follows a geometric cooling schedule. When a neighbor has been accepted, a short-term memory prevents the swapped jobs to return to their previous rank during the tabu tenure. This idea can easily be generalized to more than three schedules, and Liao et al. [20] apply the genetic algorithm of Reeves [30] and a tabu search to a representation of all m schedules in the same manner to produce non-permutation schedules.

Ying and Lin [44], Sadjadi et al. [36], Yagmahan and Yenisey [42], and Rossi and Lanzetta [31,32] investigate ant colony optimization (ACO) for the FSSP. All these approaches work on the disjunctive graph model. An ant produces a schedule by traversing the graph from the artificial source to the artificial sink node,