Computers & Operations Research 74 (2016) 165–174

Contents lists available at ScienceDirect

Computers & Operations Research

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

New simple constructive heuristic algorithms for minimizing total flow-time in the permutation flowshop scheduling problem

Hamid Abedinnia, Christoph H. Glock*, Andreas Brill

Institute of Production and Supply Chain Management, Department of Law and Economics, Technische Universität Darmstadt, Hochschulstr. 1, 64289 Darmstadt, Germany

ARTICLE INFO

Article history:
Received 7 February 2015
Received in revised form
19 March 2016
Accepted 8 April 2016
Available online 13 April 2016

Keywords:
Flowshop scheduling problem
Permutation flowshop
Total flow-time
Heuristics

ABSTRACT

This paper develops a set of new simple constructive heuristic algorithms to minimize total flow-time for an n-jobs $\times$ m-machines permutation flowshop scheduling problem. We first propose a new iterative algorithm based on the best existing simple heuristic algorithm, and then integrate new indicator variables for weighting jobs into this algorithm. We also propose new decision criteria to select the best partial sequence in each iteration of our algorithm. A comprehensive numerical experiment reveals that our modifications and extensions improve the effectiveness of the best existing simple heuristic without affecting its computational efficiency.

1. Introduction

A flowshop production system is commonly defined as a production system in which a set of n jobs undergoes a series of operations in the same order [23]. Determining optimal job sequences for flowshop scheduling problems can be based on various objectives; minimizing makespan and minimizing total flow-time have, however, most often been considered as objectives for flowshop scheduling problems in the past. The first objective refers to the minimization of the last job's completion time, while the second one aims on minimizing the total in-process time, which reduces work-in-progress inventory [6]. For makespan minimization, problems with more than two machines have been shown to be strongly NP hard [23]; this is even the case for Permutation Flowshop Scheduling Problems, i.e. for flowshop scheduling problems with the same job order on all machines. Garey et al. [11] showed that the problem of minimizing total flow-time with more than one machine belongs to the category of NP complete problems. Accordingly, for large-size problems, heuristic procedures have to be used to find solutions in reasonable computational time. A comprehensive review of research on flowshop scheduling that appeared during the last 50 years is the one of Gupta and Stafford [14]. A review of scheduling problems that aim on minimizing makespan can be found in Ruiz and Maroto [28] and Gupta et al. [13]. The Permutation Flowshop Scheduling Problem with the objective of flow-time minimization was reviewed by Pan and Ruiz [22] and Framinan et al. [8], where the latter also reviewed works that consider makespan minimization. Mutlu and Yagmahan [18] recently reviewed multi-objective flowshop scheduling problems.

Framinan et al. [10] provided a framework to categorize heuristic algorithms for the Permutation Flowshop Scheduling Problem (PFSP) according to their structure. This framework distinguished between the phases of (a) index development, (b) solution construction, and (c) solution improvement. Framinan et al. [9] categorized existing heuristics, which can address one or more of these phases, into two classes: simple and composite heuristics. An algorithm was categorized as a simple heuristic if it does not include another heuristic. Composite heuristics are heuristics that contain at least one simple heuristic for conducting one or more of the three above-mentioned phases. Pan and Ruiz [22] showed that composite heuristics outperform simple heuristics in minimizing flow-time. Yet, as simple heuristics are the basic building blocks of composite heuristics, improving their performance is still of interest for the research community, as this improvement can boost the performance of composite heuristics as well. The aim of this paper is to propose a set of new simple heuristics to improve the performance of the best existing simple heuristic algorithm for minimizing total flow-time in the PFSP.

A popular simple heuristic for minimizing makespan in the general PFSP was presented by Nawaz et al. [19] (we refer to this heuristic as NEH in the following), which outperformed other

* Corresponding author.
E-mail addresses: hamid@im.wi.tu-darmstadt.de (H. Abedinnia), glock@bwl.tu-darmstadt.de (C.H. Glock), Andreas.brill@gmx.com (A. Brill).

http://dx.doi.org/10.1016/j.cor.2016.04.007
0305-0548/© 2016 Elsevier Ltd. All rights reserved.

algorithms developed earlier, such as the heuristics of Palmer [20], Gupta [12], or Campbell et al. [2]. Despite its good performance for makespan-related PFSPs, another advantage of NEH is that it leads to good solutions for other objectives as well, such as minimizing total flow-time (as was shown, for example, by Allahverdi and Aldowaisan [1]). The NEH heuristic consists of two phases, namely (I) the sorting (prioritizing) phase and (II) the insertion phase. In the sorting phase, jobs are sorted in descending order of their total processing time. This sorted list is used in the insertion phase to determine the sequence in which jobs are added to an existing partial sequence. For an n-job PFSP, the insertion phase consists of n iterations. In step k ($1 \le k \le n$) of the insertion phase, the kth job on the sorted list is successively assigned to the k possible slots in the current partial sequence that was obtained in the previous iteration, which consists of $k-1$ jobs. The partial sequence that leads to the best value for the objective function (minimum partial makespan) is used as the current k-jobs partial sequence for the next iteration.

Since 1983, many researchers have tried to improve NEH for different objective functions by modifying either its sorting or its insertion phase. One example is the work of Framinan et al. [7], which tried to improve the performance of NEH for three objectives (i.e., makespan, idle time and total flow-time minimization) by applying 177 new ordering policies to the sorting phase of NEH. These policies are combinations of different indicator values and sorting criteria. Most extensions of NEH are more effective than the original version (i.e., they lead to better solutions), but they are usually less efficient (i.e., they are usually more complex and require more computational time than the original NEH).

The relatively high efficiency of NEH is primarily due to the idea of keeping an established partial sequence of a set of jobs unchanged from one iteration until the algorithm terminates. This idea, however, also restricts the effectiveness of NEH, as it does not search for potentially better local solutions once a partial sequence has been established. One option to improve the insertion phase of NEH is to optimize partial sequences by testing alternative positions for jobs at the end of each iteration, i.e. to evaluate the neighborhood of each partial sequence. A similar idea was presented by Rajendran [24], who optimized partial sequences by exchanging adjacent jobs pairwise with the objective to minimize total flow-time. Framinan and Leisten [6] combined this idea with NEH and performed pairwise exchanges at the end of each iteration to improve partial sequences. The authors showed that their algorithm (to which we refer as FL hereafter) outperformed other constructive algorithms for the total flow-time criterion. Framinan et al. [9] evaluated different heuristic algorithms for the PFSP and concluded that the FL heuristic led to better solutions for the total flow-time criterion. Laha and Sarin [17] extended FL by allowing all jobs assigned to a partial sequence to change their respective position by checking all other $k-1$ slots at the end of each iteration. They showed that their algorithm (to which we refer as LS in the following) leads to a better performance, in terms of the quality of the solutions, and only a small loss in efficiency as compared to the FL heuristic. Pan and Ruiz [22] reviewed the most promising constructive heuristics and indicated that LS is the best existing simple heuristic to minimize total flow-time in general PFSPs in terms of the quality of the results. Since LS is computationally complex, the authors developed some new composite heuristics that outperform LS and at the same time consume about one order of magnitude less CPU time. Recently, Fernandez-Viagas and Framinan [5] proposed a set of new constructive heuristics (we refer to them as FF heuristics in the following) and compared them with some of the heuristics considered in Pan and Ruiz [22]. Although some of their composite algorithms showed a better performance than the ones proposed in Pan and Ruiz [22], their proposed simple heuristics (all pure FF heuristics, i.e. FF(1)–FF(n)) are outperformed by LS.

As mentioned above, having promising composite heuristics does not render efforts to improve simple heuristics worthless. Better simple heuristics may open the gate for the development of even better composite heuristics. Based on LS, this paper proposes several new simple heuristics for the PFSP. Numerical experiments illustrate that our modifications lead to a significant improvement in terms of the quality of the solutions without affecting the computational efficiency as compared to the best existing simple heuristic.

The remaining sections of this paper are organized as follows. Section 2 outlines the heuristic of Laha and Sarin [17] and possible modifications for its extensions. Section 3 describes the proposed heuristics in detail. A comprehensive comparison of the proposed heuristics and LS, together with a detailed evaluation of the effect of the proposed modifications on NEH and LS, are given in Section 4, and Section 5 concludes the paper.

2. The heuristic of Laha and Sarin and its modifications

Both the FL and LS heuristics optimize partial sequences at the end of each iteration of NEH's insertion phase. This paper focuses on improving the LS heuristic, which outperforms all other existing simple heuristics for optimizing total flow-time in a permutation flowshop manufacturing system [22]. The pseudocode of LS is the following:

```
Step 1: $P_i = \sum_{j=1}^{m} p_{ij}$, $i = 1,2, \ldots, n$, where $P_i$ is the indicator value of job $i$ and $p_{ij}$ is the processing time of job $i$ on machine $j$.
Step 2: Sort the jobs in an *ascending* order of their indicator values.
Step 3: Select jobs $k = 1$ and $k = 2$ and keep the partial sequence (i.e. $1-2$ or $2-1$) that results in a shorter total flow-time as the current partial sequence.
Step 4: For $k = 3, \ldots, n$, repeat the following:
    4.1 Insert the $k$th job in all $k$ possible slots in the partial sequence obtained in the last iteration, which consists of $k-1$ jobs.
    4.2 Select the best $k$-job partial sequence that results in the shortest total flow-time as the current partial sequence.
    4.3 For $i = 1, \ldots, k$, remove job $i$ from the current partial sequence and insert it into the $k-1$ positions of the remaining partial sequence. Calculate the corresponding total flow-time for all new combinations.
    4.4 If the best of the new $k(k-1)$ $k$-job partial sequences generated in Step 4.3 is better than the current partial sequence, replace it by the best partial sequence obtained in Step 4.3. Set $k = k + 1$.
```

It is worth noting that the first three steps of LS are almost identical to those of the original NEH heuristic. The only difference is that in LS, unlike in the original NEH, the jobs are sorted in ascending order of their weights (Step 2). Framinan et al. [8] showed that minimizing total flow-time in a PFSP by using a modified version of the NEH heuristic, with jobs sorted in an *ascending* order of their total processing times, performs better than the original NEH. It is also worth noting that partial sequences are optimized starting with Step 4 of LS.

As LS is based on NEH, we have the same options for improving LS as for extending the NEH heuristic. Framinan et al. [7] named six attributes of the NEH heuristic that offer rooms for extensions:

(1) Consider different objective functions, such as makespan, total flow-time or idle-time minimization.
(2) Employ different indicator values in Step 1 (e.g. as by Framinan et al. [7]).