Available online at www.sciencedirect.com
ScienceDirect
Omega 37 (2009) 155 – 164
omega
The International Journal of Management Science
www.elsevier.com/locate/omega

Efficient composite heuristics for total flowtime minimization in permutation flow shops☆

Xiaoping Li[a, b, *], Qian Wang[a, b], Cheng Wu[c]

[a]School of Computer Science & Engineering, Southeast University, 210096 Nanjing, PR China
[b]Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, PR China
[c]Department of Automation, Tsinghua University, 100084, Beijing, PR China

Received 8 November 2006; accepted 10 November 2006
Available online 12 January 2007

Abstract

In this paper, permutation flow shops with total flowtime minimization are considered. General flowtime computing (GFC) is presented to accelerate flowtime computation. A newly generated schedule is divided into an unchanged subsequence and a changed part. GFC computes total flowtime of a schedule by inheriting temporal parameters from its parent in the unchanged part and computes only those of the changed part. Iterative methods and LR (developed by Liu J, Reeves, CR. Constructive and composite heuristic solutions to the $P\|\Sigma C_i$ scheduling problem, European Journal of Operational Research 2001; 132:439–52) are evaluated and compared as solution improvement phase and index development phase. Three composite heuristics are proposed in this paper by integrating forward pair-wise exchange-restart (FPE-R) and FPE with an effective iterative method. Computational results show that the proposed three outperform the best existing three composite heuristics in effectiveness and two of them are much faster than the existing ones.

Keywords: Composite heuristic; Flow shop; Scheduling; Flowtime

1. Introduction

Flow shop scheduling is an important manufacturing system widely existing in industrial environments. A flow shop can be described as n jobs being processed on m machines and each job having the same machine order [1]. There are many kinds of flow shops, such as blocking flow shops [2], no-wait flow shops [3], and so on. Nearly all of them are derived from permutation flow shop, the job-order on each machine follows the same order. Makespan and total flowtime (or an equivalent mean flowtime if all machines are available at time zero) are two important performance measures in permutation flow shops. In practice, total flowtime is usually considered because it can lead to stable or uniform utilization of resources, rapid turn-around of jobs, and minimization of in-process inventory [4,5].

Although the makespan and total flowtime minimization in flow shops are NP-complete [6], it seems that NEH (introduced by Nawaz, Enscore and Ham, Omega [7]) is the best heuristic for makespan minimization [8].

☆ This manuscript was processed by Associate Editor Semple.

* Corresponding author. School of Computer Science & Engineering, Southeast University, 210096 Nanjing, PR China.
Tel.: +86 25 83790901.
E-mail addresses: xpli@seu.edu.cn (X. Li), qwang@seu.edu.cn (Q. Wang), wuc@tsinghua.edu.cn (C. Wu).

0305-0483/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.omega.2006.11.003

However, so far no method seems to be the best for total flowtime minimization. Heuristics and meta-heuristics are commonly studied for flow shops. A meta-heuristic method always obtains better solution than a heuristic does but needs much more CPU-time, which cannot meet the real time requirements in practice, especially for large-scale cases. Thus, it is essential to develop effective heuristics to find good solutions in limited CPU-time for practical manufacturing systems.

For decades, many heuristics have been developed for the problems considered. Though NEH is claimed the best for makespan minimization in flow shops, it is not effective even for total flowtime ones [9]. Heuristics presented by Gupta [10], Rajendran and Chaudhuri [11], Rajendran [12], Ho [13] and Wang et al. [14] were efficient algorithms. Some efficient constructive heuristics are FL (proposed by Framinan and Leisten [15]), WY (presented by Woo and Yim [16]) and RZ (developed by Rajendran and Ziegler [17]). FL integrates NEH insertion with pair-wise exchange. In NEH insertion, an unscheduled job of the seed generated by some rule is inserted into every possible slot of the current solution (a schedule/partial schedule) and the best one is selected as the new current solution. A pair-wise exchange generates solutions by exchanging positions of every pair of jobs to improve the current solution. If the current solution is worse than the best of the generated ones, it is replaced with the best. WY is also derived from NEH but no seed is predetermined and all unscheduled jobs perform job-insertion. RZ is founded on a different job-insertion from NEH, in which a seed is a result of sorting jobs with weighted processing times and it is set as the current solution. Every job of the seed performs job-insertion subsequent to the current solution without the inserted job. By comparing WY against RZ [16,18], it is found that RZ outperforms WY for small instances but the relative performance of WY improves with the number of jobs and finally WY outperforms RZ. Computational results of Framinan and Leisten [15] show that FL outperforms both WY and RZ for majority of the randomly generated instances. The temporal complexities of FL and WY are $O(n^4m)$, whereas RZ is $O(n^3m)$.

Recently, many composite heuristics have been proposed, such as IH1–IH7 (described by Allahverdi and Aldowaisan [18]), IH7-FL (given by Framinan and Leisten [15]), FLR1 and FLR2 (presented by Framinan, Leisten and Ruiz-Usano [19]). Of these heuristics, FLR2, IH7-FL, and FLR1 seem to be the most efficient ones, all of which adopt FL to construct a solution or improve the current solution. Most existing heuristics improve their solutions only by some one-pass method. However, most solutions can be greatly improved by an iterative method, which will be shown by the three composite heuristics proposed in this paper.

The paper is organized as follows. The problem is described in Section 2. In Section 3, insertion and pair-wise exchange-based heuristics are improved by general flowtime computing (GFC). Three iterative composite heuristics are proposed in Section 4. Computational results are described in Section 5, followed by conclusions in Section 6.

2. Problem description

A flow shop is a scheduling problem in which each job in $\Omega = \{J_1, \ldots, J_n\}$ is processed sequentially on m machines $M_1, \ldots, M_m$. Schedule π_n is a permutation of the n jobs, which can be denoted as $(J_{[1]}, \ldots, J_{[n]})$ in which $J_{[i]} \in \Omega$ is the ith $(i = 1, \ldots, n)$ job in π_n. To denote the start of any schedule, a dummy job is introduced with zero processing times and explicitly denoted as $J_{[0]}$. Then, π_n can also be represented as $(J_{[0]}, J_{[1]}, \ldots, J_{[n]})$. Traditionally, such a permutation is intuitively depicted by Gantt chart (Fig. 1 shows the Gantt chart of a schedule of permutation flow shop). Let $O_{i,j}$ be the operation of job j $(j=0, 1, \ldots, n)$ processed on machine i $(i = 1, 2, \ldots, m)$ and $t_{i,j}$ the processing time (given in the rectangles in Fig. 1).

Let $C_{i,j}$ $(i=1, 2, \ldots, m, j=1, 2, \ldots, n)$ be the finishing time of $O_{i,[j]}$ in schedule π_n. The starting time of $O_{i,[j]}$ is the maximum of $C_{i-1,j}$ and $C_{i,j-1}$. Either $O_{i-1,[j]}$ or $O_{i,[j-1]}$ is called the immediate predecessor of $O_{i,[j]}$ if the finishing time of the former equals the starting time of the latter. Thus, $O_{i,[j]}$ $(j > 0)$ has at least one immediate predecessor. On the contrary, every operation has no more than two immediate successors. Therefore, the Gantt chart of π_n can be replaced with a weighted binary tree (WBT for short, a specially directed acyclic graph) $G = (V, E)$, of which nodes are operations labeled by corresponding processing times as weights. Edges of WBT can be constructed as follows. Starting from $u_0 = O_{1,[0]}$ as Root, $J_{[j]}$ $(j = 1, \ldots, n)$ can be picked up sequentially from π_n and connected to each $O_{i,[j]}$ with its immediate predecessor (randomly

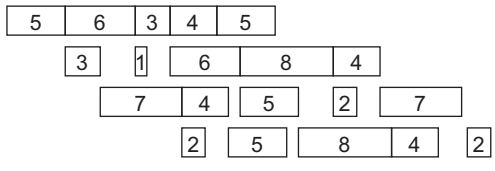

Fig. 1. Gantt chart of an instance.