ABSTRACT

We consider the problem of makespan minimization on a flexible flow shop with k stages and m_s machines at any stage. We propose a heuristic algorithm based on the identification and exploitation of the bottleneck stage, which is simple to use and to understand by practitioners. A computational experiment is conducted to evaluate the performance of the proposed method. The study shows that our method is, in average, comparable with other bottleneck-based algorithms, but with smaller variance, and that it requires less computational effort.

Keywords. Flexible flow-shop, heuristic, computational experiments.

1 INTRODUCTION

We consider the flexible flow shop (FFS) machine environment with k stages in series; at stage s, s = 1, ..., k, there are m_s identical machines in parallel. There is unlimited intermediate storage between two successive stages. Job j, j = 1, ..., n has to be processed at each stage on any one machine. The processing times of job j at the various stages are p_{1j}, p_{2j}, ..., p_{kj}. The goal is to find a schedule without preemption that minimizes the maximum completion time of all jobs (makespan).

Figure 1. Flexible (hybrid) flow-shop
based heuristic is presented in section 2. The computational study is presented in section 3, and final conclusions are given in section 4.

2 THE HEURISTIC ALGORITHM

The proposed method is inspired from the Theory of Constraints (TOC), which is an easy-to-understand manufacturing philosophy already implemented in various manufacturing environments. TOC states that a local optimum is not an optimum at all, and that the overall system performance is governed by the bottleneck resource (Goldratt and Cox, 1992). The idea is thus to find the bottleneck stage and to optimize the whole system performance by exploiting it. The basic approach of our heuristic consists on three steps: (i) bottleneck identification, (ii) times windows and scheduling of jobs at the bottleneck stage, and (iii) scheduling of jobs at nonbottleneck stages. The algorithm is now described more in detail.

2.1 The TOC-based heuristic

1. Identifying the bottleneck stage
   • For each stage $s$, compute the flow ratio
     \[ FR_s = \frac{\sum_{j=1}^{n} \frac{p_j}{m_s}}{\text{total available capacity}} \]
   • Select the stage with max FRs. Let denote $S^B$ such stage.
   • Compute release times $R^B_j$ for each job $j$ from the bottleneck stage $S^B$ as the sum of processing times in the stations before $S^B$.
   • Compute the estimated flow for each job $j$ as the sum of flows for all the stages.
   • Compute deadlines to the bottleneck station $D^B_j$ for each job $j$ as the difference between the estimated flow and the sum of processing times in the stations before $S^B$.

2. Sequencing on the bottleneck stage $S^B$
   • Schedule jobs in a list by increasing order of $R^B_j$. If there is more than one job with the same $R^B_j$, then rank them by increasing value of $D^B_j$. If there is more than one job satisfying the criterion, then prioritize them by processing times.
   • Schedule jobs on the machines according to the precedent ranking. If there is more than one machine available at time $t$, then assign the next job to the machine with less workload until time $t$. Com-

3. Sequencing on non-bottleneck stages
   • Stages after the bottleneck station: proceed in a similar way as for $S^B$, but job $j$ may be scheduled as soon as it is completed by the precedent stage.
   • Stages before the bottleneck station: in order to respect the delivery time to the bottleneck station, jobs are scheduled according to values of $D^B_j$, $R^B_j$ and processing times criteria.

3 COMPUTATIONAL STUDY

A computational experiment was conducted in order to evaluate the performance of the TOC-based method, and to compare with both the famous shifting bottleneck heuristic (SBG), and a hybrid shifting bottleneck-local search (SB-LS) heuristic (Pinedo, 1994). Algorithms were implemented in C++ language, and tests were performed on a PC Pentium 4 (750 MHz). Over 450 instances, with number of jobs ranging from 5 to 9, number of stages ranging from 3 to 5, and number of machines within stages ranging from 2 to 4, have been generated and solved by the three heuristics algorithms as well as an optimal one. The processing times of the jobs were randomly generated from a uniform distribution from 1 to 100. These algorithms were evaluated on the basis of the deviation from the optimal solution using the formula:

\[ \text{dev} = \left( \frac{C^{\text{heur}}_{\text{max}} - C^{\text{opt}}_{\text{max}}}{C^{\text{opt}}_{\text{max}}} \right) \times 100 \] (1)

We were interested in looking into the mean and the interval of deviation from the optimal solution. These results are shown in table 1, together with the worstcase obtained for the tested instances. The average value of dev for the proposed heuristic is a little higher than to the one of the shifting bottleneck algorithm (SBG), but it outperforms the hybrid SB-LS heuristic. In addition, the proposed method has a smaller interval of deviation from the optimum over the total set of instances tested. This allows us to conclude that the proposed algorithm produces results comparable with those given by other bottleneck-based methods proposed by other researchers. Besides, the worst-case value for our algorithm was neatly smaller than others.

<table>
<thead>
<tr>
<th>Table 1. Mean and worst-case values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed</td>
</tr>
<tr>
<td>Mean</td>
</tr>
<tr>
<td>Interval</td>
</tr>
<tr>
<td>Worst case</td>
</tr>
</tbody>
</table>
An evaluation of the CPU time requirements of these algorithms was also carried out. The highest value obtained for our heuristic was 0.23 seconds, while those for the SBG and SB-LS heuristics were 11 and 45 seconds, respectively. It is to note that the computational requirement of our algorithm increases with the number of stages, but it always takes less time than the others (with a factor between 17.4 and 256.4 times lower).

4 CONCLUSION

A new TOC-based algorithm to minimizing makespan in a flexible (hybrid) flow-shop has been introduced. The proposed algorithm produces results comparable with those proposed by other researchers. Since it uses a strategy based on well-known principles of the Theory of Constraints (TOC), the proposed heuristic is simple to use and simple to understand by practitioners. In particular, by exploiting the bottleneck stage, our algorithm looks at a global system optimization driving to a better utilization of production resources, instead to use ranking rules normally employed from the parallel machine environment, as suggested by other researchers (i.e. LPT-based). The work now is to relax some constraints of the conventional FFS problem (i.e. limited inter-stage storage) in order to adapt the algorithm to more real-life production scheduling problems.

It is noted that the algorithm also yields competitive results on other performance measure, i.e. flowtime. This issue should be further studied so as to determine the effect of the TOC-based procedure on other performance measures.

Further research should also focus on resource load balancing on proportional machines (not identical resources) since it is a more realistic problem.

REFERENCES


AUTHOR BIOGRAPHIES

MILTON J. ACERO-DOMÍNGUEZ graduated as an industrial engineer from Universidad del Norte, Barranquilla, Colombia. He is currently working at the Department of Industrial Engineering at the Universidad del Norte under a young-researcher grant from Colciencias (colombian institute for science and technology development).

CARLOS D. PATERNINA-ARBOLEDA is an assistant professor in the Industrial Engineering Department at Universidad del Norte, Barranquilla, Colombia. He obtained his Ph.D. degree from University of South Florida (Tampa, FL) in May 2000. His research interests include simulation.
modeling and optimization, Ant Colony Systems and heuristic optimization of large scale industry problems. Professor Paterna can be reached by e-mail at <cpaterni@uninorte.edu.co>.