Computers & Operations Research 38 (2011) 267–276

Contents lists available at ScienceDirect

Computers & Operations Research

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

An efficient heuristic for adaptive production scheduling and control in one-of-a-kind production

Wei Li [a], Barrie R. Nault [b], Deyi Xue [a], Yiliu Tu [a,*]

[a] Department of Mechanical and Manufacturing Engineering, University of Calgary, Calgary, Alberta, Canada T2N 1N4
[b] Management Information Systems Area, Haskayne School of Business, University of Calgary, Calgary, Alberta, Canada T2N 1N4

ARTICLE INFO

Available online 8 May 2010

Keywords:
Flow shop scheduling
Adaptive production control
Petri nets
Simulation

ABSTRACT

Even though research in flow shop production scheduling has been carried out for many decades, there is still a gap between research and application—especially in manufacturing paradigms such as one-of-a-kind production (OKP) that intensely challenges real time adaptive production scheduling and control. Indeed, many of the most popular heuristics continue to use Johnson's algorithm (1954) as their core. This paper presents a state space (SS) heuristic, integrated with a closed-loop feedback control structure, to achieve adaptive production scheduling and control in OKP. Our SS heuristic, because of its simplicity and computational efficiency, has the potential to become a core heuristic. Through a series of case studies, including an industrial implementation in OKP, our SS-based production scheduling and control system demonstrates significant potential to improve production efficiency.

1. Introduction

As a typical manufacturing paradigm, one-of-a-kind production (OKP) challenges production scheduling and control differently than mass production. High throughput in OKP is an extreme example of mass customization, which is one of the important strategies in the current economy [1] where the objective is to maximize the customer satisfaction by producing highly customized products with near mass production efficiency. OKP is intensely customer focused such that every product is based on specific customer requirements, and products differ on matters of colors, shapes, dimensions, functionalities, materials, processing times, and so on. Consequently, a product that is produced on an OKP flow line is rarely repeated [2], although some processes in the production of similar kinds of products can be repeated. Moreover, unexpected disturbances frequently and randomly happen to affect the daily production on OKP shop floors, such as job insertion/cancellation, machine breakdown/operator absence, and variations in processing times. Thus, OKP companies use mixed-product production on a flow line to improve production efficiency [3,4], and they have to adaptively schedule and control production online.

If a customer order in OKP is viewed as a project, to schedule the production of products in the customer order in OKP is very similar to scheduling a project. In fact, to schedule production in OKP is concurrent engineering, including product design, process planning, resource allocation, and finally the production schedule. Through this concurrent engineering effort, a combined engineering file for a product, including bill of materials (BOMs), bill of operations (BOOs), and resource constraints, is generated. This file is referred to as a product production structure (PPS) [5]. After the generation of a PPS for an OKP product, the processing times of each operation in the PPS are quoted based on the previous production of similar products. After all the PPSs for a batch of OKP products are determined and the processing times of all the operations are quoted, heuristics are needed to finally sequence the products in the batch to minimize the makespan. The state space (SS) heuristic that we present in this paper is typically for flow shop scheduling. In project scheduling, the emphasis is placed on how to allocate scarce resources to dependent activities or operations of a project to control the budget or minimize the duration of the project [6,7]. These dependent activities or operations are normally arranged in a hierarchy, commonly known as a precedence diagram. If the sequence of a series of projects needs to be adaptively adjusted in order to minimize the duration to complete all projects, the SS heuristic may be applied.

Currently, OKP management primarily uses priority dispatching rules (PDRs) to deal with disturbances. It is fast and simple to use PDRs to control production online, but PDRs depend heavily on the configuration of shop floors, characteristics of jobs, and scheduling objectives [8], and there is no specific PDR that clearly dominates the others [9]. Moreover, the performance of PDRs is poor on some scheduling objectives [10], and it is especially inconsistent when a processing constraint changes [11]. Consequently, there is a considerable difference between the scheduled and actual production progress [12]: when unexpected changes

* Corresponding author. Tel.: +1 403 220 4142; fax: +1 403 282 8406.
E-mail address: paultu@ucalgary.ca (Y. Tu).

0305-0548/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2010.05.002

occur and PDRs are used to adaptively control production, production often runs in a chaotic or an "ad hoc fire fighting" manner [13,14].

Indeed, due to dynamic disturbances, OKP has to be adaptively scheduled and controlled [2,14]. When adaptive production control is taken into consideration, a closed-loop control structure is necessary and an efficient heuristic is critical. We propose a state space (SS) heuristic to support a computer-aided production scheduling and control system. We compare the optimality of our SS heuristic in terms of minimizing the maximum completion time, to the CDS heuristic [15] and to the NIS heuristic [16] using case studies based on well-accepted benchmarks, for both traditional flow shop (TFS) and hybrid flow shop (HFS) problems under no pre-emption or no wait processing constraints. Both the CDS and NIS heuristics use Johnson's algorithm as their core. In addition to its self-contained performance, we believe that our SS heuristic – because of its simplicity and computational efficiency – has the potential to become a core heuristic.

We find that across our different case studies the SS heuristic outperforms the CDS and NIS heuristics. In addition, in a real industrial application at an OKP company, Gienow Windows and Doors, our production scheduling and control system based on the SS heuristic reduced the company's original scheduling period using PDRs by an order of magnitude from 3 days to 2 h, providing the company significant flexibility and competitiveness.

The rest of this paper is organized as follows: Section 2 gives a brief literature review. Section 3 introduces the SS heuristic. Section 4 presents the scheduling system and a closed-loop control structure for adaptive control in OKP. Section 5 gives the results from our case studies on TFS and HFS problems under no pre-emption or no wait processing constraints, operator absence disturbances, and in an industrial setting. We also provide a possible extension of our SS heuristic. Finally, Section 6 draws conclusions and proposes future work.

2. Literature review

Research in production scheduling has been carried out for many decades, and there are numerous scheduling methods developed in the literature. In this section, we briefly review flow shop production scheduling methods, and discuss the requirements of heuristics for adaptive production control.

Scheduling is a decision making process of allocating resources to jobs over time to optimize one or more objectives. According to [17], one type of flow shop consists of m-machines in series, and each job has to be processed on each one of m-machines in a single direction, which means first on machine 1, then machine 2, and so on. This is typically called a traditional flow shop (TFS). Another type of flow shop where there are S-stages in series with a number of machines/operators in parallel in each stage is a flexible flow shop or hybrid flow shop (HFS). In addition to the difference in flow shop configurations, processing constraints are also different for TFS and HFS. For TFS, if the first in first out (*FIFO*) discipline is applied to jobs in work-in-process (WIP) inventories, then it becomes a no pre-emption or permutation (*prmu*) flow shop problem. For HFS, if the first come first serve (*FCFS*) discipline is applied, then it is still a no pre-emption flow shop problem but the output sequence from each stage may change. Another processing constraint could be no wait (*nwt*), that is, jobs are not allowed to wait between two machines or stages, which also means there is no intermediate storage. The most common objective of flow shop scheduling is to minimize the maximum completion time or makespan, that is, min(C_{max}). Following the popular three parameter notation, $\alpha/\beta/\gamma$, the above problems can be expressed as $Fm/prmu/C_{max}$ for m-machine TFS problems with no pre-emption constraint to minimize makespan, $Fm/nwt/C_{max}$ for m-machine TFS problems with no wait constraint to minimize makespan, $FFs/FCFS/C_{max}$ for S-stage HFS problems with *FCFS* constraint to minimize makespan, or $FFs/nwt/C_{max}$ for S-stage HFS problems with no wait constraint to minimize makespan.

Gupta and Stafford [18] chronologically reviewed flow shop scheduling research in the past five decades since the classic Johnson's algorithm in 1954. They found that the emergence of NP-completeness theory in the third decade (1975–1984) profoundly impacted the direction of research in flow shop scheduling. That is why heuristics are required to solve large problems. HFS problems emerged in the fourth decade (1985–1994), and various artificial intelligence based heuristics were proposed then. The fifth decade (1995–2004) witnessed the proliferation of various flow shop problems, objective functions, and solution approaches. Although flow shop scheduling has been researched for more than 50 years, there remains a large gap between theoretical research and industrial applications [18].

Framinan et al. [19] proposed a general framework for the development of heuristics which consists of three phases: index development, solution construction, and solution improvement. Phase I, index development, means that jobs are arranged according to a certain property based on processing times. For example, Campbell et al. [15] extended Johnson's algorithm and proposed the CDS heuristic for an n-job m-machine TFS problem to min(C_{max}). The CDS heuristic using Johnson's algorithm to arrange jobs is as follows. If there is a counter (*Ctr*) pointing to a machine j, then for each job i ($i=1,\ldots,n$) the sum of processing times on the first *Ctr* machines is regarded as its processing time on virtual machine 1, and the sum of processing times on the rest $m-Ctr$ machines as its processing time on virtual machine 2. Then apply Johnson's algorithm to this virtual 2-machine flow shop problem to get a sequence. As *Ctr* changes from 1 to $m-1$, $m-1$ sequences are generated, and the one with the minimum makespan is the final solution. In phase II, solution construction, a solution is constructed by a recursive procedure, trying to insert one or more unscheduled jobs into a specific position of a partial sequence until the final schedule is completed. NEH [20] is a typical phase II heuristic for an n-job m-machine TFS problem to min(C_{max}). The NEH procedure is as follows. Firstly, for each job, NEH sums the processing times on all of m-machines, and then arranges these sums in a non-ascending order. Secondly, NEH schedules the first two jobs to get a partial sequence, and then inserts the third job into three possible positions to get another partial sequence, and so on. Finally, NEH inserts the last job into n possible positions, and then determines the final schedule. In phase III, solution improvement, there are two main characteristics. The first is that there must be an initial schedule, and the second is, after using artificial intelligence techniques, the quality of solution is better than the initial schedule. For the future development of heuristics, Framinan et al. [19] clearly stated the importance of heuristic development in phase I, index development, should not be underestimated, as it is required for the other two phases.

In a case study including 19 constructive heuristics for $Fm/prmu/C_{max}$ problems, Ruiz and Maroto [10] concluded that the NEH heuristic is best, the CDS heuristic is 8th, and two PDRs (LPT and SPT rules) are the worst. However, the CDS heuristic has the simplest computational complexity among the first 8 heuristics, $O(m^2n+mn\log n)$. Moreover, King and Spachis [11] did case studies of 5 PDRs and the CDS heuristic for two different TFS problems, $Fm/prmu/C_{max}$ and $Fm/nwt/C_{max}$. They concluded that the CDS heuristic and LWBJD (least weighted between jobs delay) rules work best for $Fm/prmu/C_{max}$ problems and MLSS (maximum left shift savings) rule works best for $Fm/nwt/C_{max}$