Omega ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Omega

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

Synchronous flow shop problems: How much can we gain by leaving machines idle? ☆

Stefan Waldherr [a], Sigrid Knust [a,*], Dirk Briskorn [b]

[a] Institute of Computer Science, University of Osnabrück, 49069 Osnabrück, Germany
[b] Department of Production and Logistics, University of Wuppertal, 42119 Wuppertal, Germany

ARTICLE INFO

Article history:
Received 14 December 2015
Accepted 27 October 2016

Keywords:
Synchronous flow shop
Dummy jobs
Idle times

ABSTRACT

In synchronous production lines it may be beneficial to leave machines idle instead of processing the next job immediately. In this paper, the effects of inserting voluntary idle times are discussed in more detail for different objective functions (minimization of makespan, total completion time, maximum lateness). Besides deriving theoretical bounds on how much can be gained by inserting idle times, an extensive computational study is conducted to empirically examine the actual improvements. For this, exact algorithms and heuristics capable of incorporating voluntary idle times are proposed to find (near-) optimal schedules. It can be seen that the potential gain is very large in theory, while the empirical results indicate that in general only small improvements are achievable in practice.

1. Introduction

In this paper, we study synchronous flow shop problems and the effects of voluntarily leaving machines idle for some time periods. A "synchronous flow shop" ("SFS" for short, also called "flow shop with synchronous movement") models a synchronous unpaced assembly line (cf. [3]) and is a variant of a non-preemptive permutation flow shop where transfers of jobs from one machine to the next take place at the same time. The processing of jobs on their respective machines is therefore organized in synchronous cycles and the jobs are transferred to the next machine only after the current jobs on all machines are finished. As a consequence, the processing time of a cycle ("cycle time" for short) is determined by the maximum processing time of the operations contained in it. If the processing time of a job on a certain machine is smaller than this maximum, the corresponding machine is idle (but occupied) until the job may be transferred to the next machine. In contrast, in a classical flow shop the transfer of jobs is asynchronous: Jobs may be transferred to the next machine as soon as their processing on the current machine is completed and processing on the next machine immediately starts as soon as this machine is available.

Synchronous flow shops have been studied in the broader field of assembly line balancing. Most research in this area is performed on the problem of designing and configuring the production system (e.g., assigning workers and tasks to workstations, determining a line balancing or allocating buffer storage), see e.g., Doerr et al. [5], Urban and Chiang [16], Vairaktarakis et al. [17]. In comparison to the asynchronous unpaced case, which relates to the intensively studied classical flow shop model (see e.g., Emmons and Vairaktarakis [6], Gupta and Stafford [7], Yenisey and Yagmahan [22]), there is only few literature concerning solution procedures for SFS models as pointed out by Boysen et al. [3]. Kouvelis and Karabati [12] considered a production line with synchronous movement, where a set of jobs is produced periodically and the throughput is to be maximized (which is equivalent to minimizing the total cycle time of the production line). In [12] it is proven that the problem is $\mathcal{NP}$-hard for an arbitrary number of machines, furthermore, a mixed integer programming formulation and a heuristic solution approach are presented. For the non-periodic version of this problem, the $\mathcal{NP}$-hardness result was strengthened by Waldherr and Knust [20] who proved that the SFS problem with the makespan objective is already strongly $\mathcal{NP}$-hard for three machines. Moreover, they showed that minimizing the maximum lateness as well as minimizing the total completion time is strongly $\mathcal{NP}$-hard even for two machines. Soylu et al. [14] present a branch-and-bound approach and several heuristics to minimize the makespan in SFSs. Huang [8] as well as Huang and Ventura [10] consider rotating production units with synchronous movement and a loading/unloading (L/U) station. In this

☆This work was supported by the Deutsche Forschungsgemeinschaft, KN 512/7-1
* Corresponding author.
E-mail addresses: stefan.waldherr@uni-osnabrueck.de (S. Waldherr), sigrid.knust@uni-osnabrueck.de (S. Knust), briskorn@uni-wuppertal.de (D. Briskorn).

http://dx.doi.org/10.1016/j.omega.2016.10.006
0305-0483/

framework, a job enters the production unit at the L/U station and is then processed on all machines before returning to the L/U station where it is unloaded. A polynomial time algorithm to minimize the makespan for a production unit with two machines and constant product-independent removal times is presented. Additionally, dynamic programming approaches for the case with non-constant removal times and two or three machines are proposed. A genetic algorithm is presented by Huang and Hung [9]. Baker [2] evaluates a spreadsheet-based approach for synchronous flow shops. A practical application was studied in Waldherr and Knust [19], where a production line with synchronous movement resembling a synchronous flow shop was used in the production process of shelf-boards at a kitchen manufacturer.

For classical flow shops, positive effects of voluntarily leaving machines idle (also referred to as idle time insertion) have been known for a long time, e.g., Baker [1], Kanet and Sridharan [11]. In these models, the scheduler is allowed to delay the start of an operation despite it being ready for processing in order to wait for the availability of an operation of another job. This strategy can be beneficial, e.g., in models with release times for the jobs or problems in which jobs finishing too early are penalized. Since in the classical flow shop the transportation is asynchronous, delaying the start of an operation on one machine does not affect the operations currently processed on the other machines. In the SFS model, this is only true if the delay of an operation is shorter than the idle time enforced by the maximum processing time of an operation processed in the same cycle. Otherwise, delaying the operation also results in additional idle times on all other machines. Kouvelis and Karabati [12] pointed out that the performance of synchronous production lines may be improved by leaving machines completely idle via not starting a job on the first machine in some cycles and instead only moving all other jobs to their respective next machine. Due to the synchronous movement of all jobs, an idle machine in the current cycle leads to its succeeding machine being idle in the next cycle. To model this, in [12] it was suggested to introduce dummy jobs with zero processing times on all machines and to insert them into the schedule accordingly. Using a small example with three machines and three jobs, it was shown that introducing such dummy jobs may decrease the total cycle time. The authors stated that the total cycle time is not a monotone function in the number of dummy jobs (i.e., an optimal solution for $k+1$ dummy jobs may be worse than an optimal solution for k dummy jobs). Thus, one has to try different numbers of dummy jobs in order to find an optimal solution in the class of schedules allowing an arbitrary number of dummy jobs.

While in [12] it has been shown that voluntary idle times can improve the optimal objective value in a SFS, the gains have not been quantified. Motivated by the practical application studied in [19] we became interested in the question whether the example of [12] is artificially constructed or the use of dummy jobs may really be beneficial in practice. The aim of this paper is to study more deeply (analytically and empirically) the effects of introducing dummy jobs.

From a theoretical view, it is interesting to study the limits of how much can be gained by inserting dummy jobs. In this paper we do so, first, by a formal analysis bounding the absolute and relative improvement of the objective value when dummy jobs are allowed. For both, theoretical and practical purposes, we derive upper bounds on the maximum number of dummy jobs whose insertion may still improve the objective value. This information can be very useful in algorithms where the runtime depends on the number of dummy jobs to be considered or in situations where we want to evaluate whether adding another dummy job can lead to a further improvement of the objective value at all.

To quantify the practical use of inserting dummy jobs, we conducted a computational study empirically examining the actual improvements that can be achieved. Since our goal is to evaluate the effects of dummy jobs and the improvement achievable by including them, we concentrate on exact solution methods and established heuristics for synchronous flow shop problems which obtain good results for the case without dummy jobs. Further, we focus on methods that are easy to implement. Note that highly sophisticated methods are only rarely used in practice. Since we aim at conclusions to be drawn for real-world settings, we employ methods that are likely to be used in practice.

The remainder of this paper is organized as follows. In Section 2 we give a more formal description of SFS problems and the insertion of dummy jobs. Afterwards, in Section 3 we determine upper bounds on the number of dummy jobs such that adding even more of them cannot be beneficial. In Section 4 we provide bounds on the improvement that can be realized by introducing a certain number of dummy jobs. Algorithms to solve SFS problems with dummy jobs are presented in Section 5. These algorithms are evaluated conducting a computational study in Section 6. Finally, conclusions are presented in Section 7.

2. Problem description

In this section, we describe the problems under consideration more formally and introduce the notation used throughout the remainder of the paper. We consider a permutation flow shop with m machines $M_1,\dots,M_m$ and n jobs $N=\{1,\dots,n\}$ where job $j\in N$ consists of m operations $O_{1j}\to O_{2j}\to\cdots\to O_{mj}$ which have to be processed in this order. Operation O_{ij} has to be processed without preemption on machine M_i for p_{ij} time units. Only permutation schedules are feasible, i.e., the jobs have to be processed in the same order on all machines.

The processing is organized in synchronized cycles which is motivated by production lines where jobs have to be moved from one machine to the next by an unpaced synchronous transportation system. This means that in a cycle all current jobs start at the same time on the corresponding machines. Then all jobs are processed and have to wait until the last one is finished. Afterwards, all jobs are moved to the next machine simultaneously. The job processed on the last machine M_m leaves the system, a new job (if available) is put on the first machine M_1.

Each feasible schedule may be represented by a sequence σ of jobs reflecting the order in which each machine processes the jobs. With each sequence a corresponding (left-shifted) schedule is associated in which each operation starts as early as possible. A schedule consists of $n+m-1$ cycles, which are divided into a starting phase ($m-1$ cycles, until jobs are present on each machine), a standard phase ($n-m+1$ cycles, as described above), and a final phase ($m-1$ cycles, no more jobs are available for M_1). For any sequence σ, let $\sigma(\lambda)$ be the λ-th entry in σ. In cycle t, $1\le t\le n+m-1$, operations $O_{1\sigma(t)},\dots,O_{m\sigma(t-m+1)}$ are processed (with obvious adaptations for the first $m-1$ and the last $m-1$ cycles). The cycle (processing) time P_t of cycle t can then be calculated by

$$P_t=\max\{p_{i\sigma(t+1-i)}|\max\{1,t-n+1\}\le i\le\min\{m,t\}\}$$

and its completion time is given by $\sum_{\tau=1}^{t}P_\tau$.

Let C_j be the completion time of job j, i.e., the time when j has been processed on all machines and leaves the system. We assume that a job can only be accessed after the whole cycle has been completed, i.e., the job has to wait until all jobs on the other machines in the corresponding cycle are finished. Thus, the completion time C_j of a job j is defined as the time when the

Please cite this article as: Waldherr S, et al. Synchronous flow shop problems: How much can we gain by leaving machines idle? Omega (2016), http://dx.doi.org/10.1016/j.omega.2016.10.006