MAKESPAN MINIMIZATION IN THE FLOW-SHOP SCHEDULING PROBLEM WITH SEQUENCE DEPENDENT SETUP TIMES

Mário A. G. Melo, Dept. de Engenharia Mecânica/ VICAIMA SA, Instituto Superior de Engenharia do Porto, Portugal, melomario@clix.pt
Manuel J. Pereira Lopes, Dept. de Engenharia Mecânica, Instituto Superior de Engenharia do Porto, Portugal, mpl@isep.ipp.pt

Keywords: Flow shop scheduling, sequence-dependent setup times, exact algorithm, Makespan

1. INTRODUCTION

In this paper, we address the problem of finding a permutation schedule of \( n \) jobs in an \( m \) machine flow-shop environment that minimizes the maximum completion time \( C_{\text{max}} \) (Makespan) of all jobs. The jobs are available at time zero and have sequence-dependent setup times on each machine. All parameters, such as processing and setup times, are assumed to be known with certainty. This problem is classified in the scheduling literature as the sequence-dependent setup time flow-shop (SDST flow shop). It is easy to see that the SDST flow shop scheduling problem to minimize the Makespan is \( \mathcal{NP} \)-hard in the strong sense, since the problem is equivalent to the extensively studied and computationally intractable travelling salesman problem (TSP) when \( m=1 \) [1]. Further, when sequence dependent setups are present on all machines of a flow shop, Gupta and Darrow [3] showed that permutation schedules (where the sequence of job on all stages is identical) are not always optimal even for the two-machine case.

Applications of sequence-dependent setup time scheduling are commonly found in most manufacturing environments. In printing industry, for example, presses must be cleaned and settings changed when ink color, paper size or type differ from one job to the next. Setup times are strongly dependent on the job order. In the container manufacturing industry machines must be adjusted whenever the dimensions of the containers are changed, while in printed circuit board assembly, rearranging and restocking component inventories on the magazine rack is required between batches. In each of these situations, sequence-dependent setup times play a major role and must be considered explicitly when modelling the problem.

This is the case of the real problem addressed in this work – the manufacturing of natural wood doors. In this industry, the doors in different sizes, wood species and panels, create the need to adjust the machines every time one of these parameters changes, and the setup times are strongly dependent on the job order.

In this work, we develop an exact algorithm for solving a real flow-shop scheduling problem. First, we give a general integer programming formulation (IP) for the problem. Then, taking advantage of the special structure and properties of the real system, we show how this problem can be reduced it to a single machine.

The success of the method is mainly due to the excellent balance between the model simplicity and the correct level of real system representation. Although many system variables were relaxed, the results obtained show that we were able to keep the relevant system variables in the model.

This solution was validated against the real system and the results obtained show that the approach is capable of solving problems of large size to optimality within reasonable computational time.

This paper is organized as follows. In next section, we describe the problem. In section 3 we formulate our problem, describe the simplification approach and present the simplified model. Next, in section 4, we describe the computational experiments, and report the results. Finally, in section 5, we conclude the paper.

2. PROBLEM DESCRIPTION

The production process under study is a flow shop with permutation, processing large quantities of wood panels coated, machined and packaged (more than 5,000 doors per day). The production orders are obtained through the current planning system (ERP) and comprehend the following information: product type, quantity, and production date.

Product families are defined according to the characteristics of coating, size and structure. Thus, the setup time is the time needed to adjust the machine to process a sequence of two different jobs belonging to two different families. If
two consecutive jobs to be processed belong to the same family, setup time is considered to be equal to zero. These setup times are heavily dependent on the type of operation and on the family of products.

The ERP (APO) establishes the production schedule for the first work station (stage) through a heuristic process, taking in account the delivery date and machine setups. This process will not be considered in our study as it is a "blackbox" and company management don’t want to change it, for several reasons. Subsequent work stations follow a FIFO rule. The FIFO rule is justified by the wood panels lot size and weight; it takes a lot of effort to move it out of the sequential position.

The production managers claim that this approach is far from being efficient as setup times are important and are not formally considered. Our study will focus on these work stations with the objective to improve its overall efficiency.

In the flow shop environment, a set of \( n \) independent jobs \( N=\{1, 2, ..., n\} \) must be scheduled on a set of \( m \) machines \( M=\{1, 2, ..., m\} \), where each job has the same routing, that is, they have to be processed first on machine 1, then on machine 2, and so on. Therefore, without loss of generality, we assume that the machines are ordered according to how they are visited by each job. After completion on one machine, a job joins the queue at the next machine. If the queues operate under the first in first out (FIFO) discipline, the flow shop is referred as a permutation flow shop.

Each job \( j \) consists of \( m \) operations \( O_{ij} (i=1,\ldots,m) \) which have to be processed in the order \( O_{ij} \rightarrow O_{i(j+1)} \rightarrow \ldots \rightarrow O_{im} \). Operation \( O_{ij} \) has to be processed on machine \( M_i \) without preemption for \( p_{ij} \geq 0 \) time units. Each machine can only process one job at a time.

We suppose that each job is available at time zero. We also assume that there is a setup time which is sequence dependent so that, for every machine \( i \), a setup time must precede the start of a given operation that depends on both the job to be processed \((k)\) and the job that immediately precedes it \((j)\) - a sequence dependent setup time is incurred whenever jobs are from different families of products. The setup time on machine \( i \) is denoted by \( s_{ijk} \) and is assumed to be symmetric, i.e., \( s_{ijk} = s_{kij} \). After the last job has been processed on a given machine, the machine is brought back to an acceptable "ending" state. We assume that this last operation can be done instantaneously because we are interested in job completion time rather than machine completion time.

As the buffers have high capacity (more than 4 hours) and work in-process is, typically, less than that, it is reasonable to assume that there’s no machine idle time and buffers have unlimited capacity.

Our objective is to minimize the time at which the last job in the sequence finishes processing on the last machine, that is, completion time \( C_{\text{max}} \) (Makespan) of all jobs.

All the parameters are deterministic nonnegative integers.

In the literature [2] this problem is denoted by \( Fm|s_{ijk}, \text{prmu}|C_{\text{max}} \) or SDST flow shop.

3. PROBLEM FORMULATION

In this chapter, first we will present a general formulation for the flow shop SDST problem. Then, we will describe the process followed to simplify it and present the final mathematical model.

3.1 Flow Shop Integer Programming Formulation

A standard flow shop scheduling problem can be formally stated as follows: A set \( N=\{1,2,\ldots,n\} \) of \( n \) jobs is to be processed on \( m \) stages sequentially. There is one machine at each stage. All machines are continuously available. A job is to be processed on one machine at a time without preemption and a machine processes no more than one job at a time. The objective is to schedule the jobs so as to minimize some performance measure such as the Makespan, total completion time, maximum tardiness, total tardiness, weighted tardiness, and weighted sum of earliness and tardiness, among others.

The formulation presented here is an adaptation from [2] and is based on the following assumptions:

- All \( n \) jobs are independent and available for processing at the initial time.
- The production work station has sufficient capacity to store and manage the work-in-process (WIP) inventory generated during the execution of the complete set of jobs. That is, we suppose infinite storage capacity at each stage.
- One machine can process only one job at a time and one job can be processed by only one machine at any time.
- The ready times for all machines (times when the machines become available to process the set of jobs to be scheduled) are known.
- For all the jobs, the processing times at each stage are known and deterministic.
• Job set-up times are sequence-dependent.
• Travel time between consecutive stages is negligible. In instances where this assumption does not hold true, inter-stage travel can be treated as a processing step with the process time equal to the travel time.
• Pre-emption is not allowed, that is no interruption of a job processing is allowed.

To present the mathematical, the following notation is needed:

- $x_{jk}$: a binary variable that is equal to 1 if job $j$ is the $k$th job in the sequence, 0 otherwise;
- $l_k$: an auxiliary variable which denotes the idle time on machine $i$ between the processing of the jobs in the $k$th position and $(k+1)$ position;
- $W_k$: an auxiliary variable which denotes the waiting time of the job in the $k$th position in between machines $i$ and $i+1$:
  \[ W_k = p_{ij} : \text{the processing time on machine } i \text{ of the job } j. \]
- $s_{ij}$: setup time on machine $i$ if job $i$ is processed immediately after job $j$.

The problem can now be formulated as:

1. \[ \min \ C_{\text{max}} \]
2. \[ \sum_{j=1}^{n} x_{jk} = 1, k = 1, \ldots, n \]
3. \[ \sum_{k=1}^{n} x_{jk} = 1, j = 1, \ldots, n \]
4. \[ \begin{align*} I_k + \sum_{j=1}^{n} x_{j,k+1} P_{ij} + \sum_{j=1}^{n} (x_{jk} - x_{j,k+1}) s_{ij} + W_{k} - I_{k+1} &- \sum_{j=1}^{n} x_{jk} P_{i+1,j} - \sum_{j=1}^{n} (x_{jk} - x_{j,k+1}) s_{ij} - W_{k} = 0, \quad k = 1, \ldots, n-1; \\
\end{align*} \]
5. \[ C_{\text{max}} = \left( \sum_{j=1}^{n} \sum_{k=1}^{n-1} x_{jk} P_{ij} + \sum_{j=1}^{n-1} I_{mj} \right) \]
6. \[ x_{jk} \in [0,1], I_k \geq 0, W_k \geq 0, j = 1, \ldots, n \]

The cost function (1) minimizes the Makespan which is equivalent to minimize the total idle time on last machine $m$ – equation (5). The first set of constraints (2) specifies that exactly one job has to be assigned to position $k$, for any $k$. The second set of constraints (3) specifies that job $j$ has to be assigned to exactly one position. The third set of constraints (4) relate the decision variables with the physical constraints which enforce the necessary relationships between the idle time variables and the waiting time variables. Equation (5) defines the Makespan. Constraints (6) impose binary integrality for decision variables and non-negative values for auxiliary variables.

This is a large-scale integer programming problem which is not easy to solve because it involves a nonlinear expression (4).

### 3.2 Simplified Flow Shop Integer Programming Formulation

The detailed study of the production system revealed some opportunities to simplify the formulation presented in (3.1):

- As the buffers have high capacity (more than 4 hours) and the production system initial condition is always with WIP (not empty), we can assume that machine idle time is equal to zero.
- Once a processing sequence is defined for a production batch, the FIFO rule is to be followed for all the stages. This is equivalent to say that that initial machine setup configuration is equal for all the stages.
- As there is no machine idle time, machine setup time during idle time is not applied.
- Processing times are constant for all the stages, that is, do not depend on the job sequence. Therefore, minimizing the total production batch setup times is equivalent to minimize the Makespan.

Based on these assumptions, the problem can be reduced to a single stage (machine) formulation with the objective to minimize the total setup times. This formulation is equivalent to the traveling salesman problem (TSP).

To formulate the single machine (stage) SDST model, the following notation is needed:

- $x_{jk}$: a binary variable that is equal to 1 if job $k$ is processed immediately after job, 0 otherwise;
- $s_{0k}$: a binary variable that is equal to 1 if job $k$ is the first job to be processed in the production sequence, 0 otherwise;
- $s_{0j}$: a binary variable that is equal to 1 if job $j$ is the last job to be processed in the production sequence, 0 otherwise;
- $s_{0k}$: total setup time for all $m$ stages if job $k$ is processed immediately after job, 0 otherwise;
- $s_{0k}$: total setup time (for all $m$ stages) if job $k$ is the first job to be processed in the production sequence;
- $s_{0j}$: total setup time (for all $m$ stages) if job $k$ is the first job to be processed in the production sequence;
The simplified formulation is:

(7) \[ \text{min} \sum_{j=0}^{n} \sum_{k=1}^{n} s_{jk} x_{jk} \]

(8) \[ \sum_{k=0}^{n} x_{jk} = 1, \quad j = 0, \ldots, n \]

(9) \[ \sum_{k=0}^{n} x_{jk} = \sum_{k=0}^{n} x_{kj}, \quad j = 1, \ldots, n \]

(10) \[ x_{jk} \in \{0,1\}, \quad j = 0, \ldots, n, \quad k = 0, \ldots, n \]

The cost function (7) minimizes the total sum of setup times. The first set of constraints (8) specifies that each job is processed exactly once. The set of equations (9) are flow conservation constraints. Constraints (10) impose binary integrality for decision variables.

4. COMPUTATIONAL IMPLEMENTATION AND RESULTS

In this section, we describe the method to obtain the test instances and we report the computational results.

All the algorithms involved were coded in Microsoft Excel and the Lingo package was used to solve the linear programming problems. The computational tests were performed on a desktop computer with a 2.6 GHz Pentium IV processor and 512 Mb of RAM.

4.1 Test Problems

The test problems were generated as follows:

- Selection of product families with significant setup times between them (there are about 37 product families);
- Planning Horizon: one work day which is equivalent to approximately 12 families (jobs) of products \((n = 12)\);
- For these 12 families, it was determined the total setup time (for all stages) matrix.
- Number of test instances: 10 (randomly selected)

4.2. Test Results

From the 10 instances tested we can conclude the following:

- This algorithm is able to solve one day size problems in less than 40 seconds in average;
- The schedules generated were analyzed by production management and considered correct according to their knowledge and experience;
- Production management considered this approach is valid and shows a clear improvement from the actual situation.

5. CONCLUSIONS

In this work, we develop an exact algorithm for solving a real flow-shop scheduling problem. First, we presented a general integer programming formulation (IP) for the problem. Then, taking advantage of the special structure and properties of the real system, we reduced it to a single machine problem.

The success of the method is mainly due to the excellent balance between the model simplicity and the correct level of real system representation.

All the algorithms involved were coded in Microsoft Excel and the Lingo package was used to solve the linear programming problems.

Ten test instances were generated based on the real system. The test results show this algorithm is able to solve one day size problems in less than 40 seconds in average.

The schedules generated by the algorithm were validated by production management against the real system and. They also considered that this approach represents a clear improvement from the actual situation.

References