A TWO-STAGE FLOW SHOP SCHEDULING WITH A CRITICAL MACHINE AND BATCH AVAILABILITY

Enrique GERSTL and Gur MOSHEIOV *

Abstract. We study a two-stage flowshop, where each job is processed on the first (critical) machine, and then continues to one of two second-stage (dedicated) machines. We assume identical (but machine-dependent) job processing times. Jobs are processed on the critical machine in batches, and a setup time is required when starting a new batch. The setting assumes batch-availability, i.e., jobs become available for the second stage only when their entire batch is completed on the critical machine. We consider three objective functions: minimum makespan, minimum total load, and minimum weighted flow-time. Polynomial time dynamic programming algorithms are introduced, which are numerically shown to be able to solve problems of medium size in reasonable time. A heuristic for makespan minimization is presented and shown numerically to be both accurate and efficient..

Keywords: Batch Scheduling, Makespan, Flowtime, Total Load, Flowshop, Critical Machine, Dynamic Programming, Heuristics

1. Introduction

We consider a two-stage flowshop, consisting of a critical (common) machine in stage 1, and two dedicated machines in stage 2. Each job is processed on the first machine, and then type 1 jobs continue to the first dedicated machine, and type 2 jobs continue to the second

* Enrique Gerstl, School of Business Administration, The Hebrew University, Jerusalem, Israel;

Gur Mosheiov, School of Business Administration, The Hebrew University, Jerusalem, Israel.

Acknowledgement: This paper was supported in part by The Recanati Fund and The Charles Rosen Chair of Management, The School of Business Administration, The Hebrew University, Jerusalem, Israel. We would like to thank Professor Michail Kovalyov for providing us a very relevant reference to this paper.

dedicated machine. This setting seems to have numerous applications. One practical application (see e.g. [9]) is of a standard production line consisting of a single production machine, followed by two painting machines. Note that the production (first) stage consists of identical processing times of all items, and the processing times of the items in the painting (second) stage may depend on the item type. Another example (see [3]) refers to furniture (e.g. chairs) production. The main bodies of the chairs are manufactured in the first stage, and different head supporters are assembled in the second stage. For another application (of a leading PC manufactures in Taiwan which assembles different types of boards) see [4].

The two-stage flowshop with a critical first machine and two machines in stage 2, is a special case of the setting containing m machines in stage 2 (as in [6]), which in itself is a special case of a hybrid flowshop (see e.g. [8], [10], [11], among others), where a number of parallel machines are available at each stage. Other relevant references that study flowshops with a critical machine are: [4] who proposed an IP model and a heuristic algorithm for the setting of two dedicated machines; [5] who introduced a heuristic method for a two-stage job shop with multiple machines; and [7] who addressed real-life two-stage hybrid flowshop, taken from a label sticker manufacturing company, and presented a heuristic method to minimize maximal tardiness.

In this paper we study the above setting (one critical machine in stage 1 followed by two dedicated machines in stage 2) with *identical processing time jobs*. Specifically, we assume that the processing times of the jobs (of both types) on the critical machine are identical. Jobs of both types may be processed on the critical machine in *batches*. When starting a new batch, a *setup time* is required. The goal is to allocate the jobs into batches and to schedule them on the critical machine (and then on the dedicated machines), such that the relevant objective function is minimized. There are two well-known settings: (i) *job-availability*: as soon as a job is completed its processing on the critical machine it is available for processing on its dedicated machine, (ii) *batch-availability*: jobs become available for the second stage only when their entire batch is completed. A number of papers studied makespan minimization on a two-stage flowshop with a critical machine, identical jobs and job availability; see e.g. [4], [1] and [2]. We are not aware of studies of this setting under batch availability.

We thus focus in this paper on the setting of batch availability. Three objective functions are considered: (i) makespan, (ii) weighted flowtime, and (iii) total load. While the first two objectives are popular and commonly studied, total load is a non-standard measure. It consists of the sum of the machine completion times. A recent paper [3] proved that for general job processing times, the total load problem is strongly NP-hard, and introduced a polynomial time solution for the special case in which the job sequence for each product type is given (and the goal is to find the optimal mixed sequence on the critical machine). We study all three objective functions for the setting of identical (machine-dependent) processing times. In all cases, we introduce dynamic programming algorithms, which are polynomial in the number of jobs. Numerical tests indicate that these algorithms are able to solve medium size instances in reasonable time. A simple heuristic algorithm based on equal allocation of jobs to batches is introduced for makespan minimization. Numerical tests confirm both its accuracy (e.g. average optimality gaps of no more than 1.7% for instances of 700 jobs or more), and its efficiency (e.g. average running time of 73 milliseconds required for solving a single 1000-job problem).

In Section 2 we provide the notation and the problem formulation. Sections 3, 4 and 5 present the dynamic programming algorithms for minimum makespan, total load and weighted flowtime, respectively. All the sections contain numerical examples and numerical tests that confirm the efficiency of the algorithms for solving medium size instances. Section 6 introduces the heuristic algorithm for makespan minimization, and its numerical results.

2. Formulation

n jobs need to be processed on a two-stage flowshop. The flowshop consists of a critical machine at stage 1 (denoted by M_0), and two dedicated machines at stage 2 (denoted by M_1 and M_2 , respectively). Two types of jobs are processed: the first type (of α -jobs) are processed on the critical machine and then on machine M_1 , and the second type (of β -jobs) are processed on the critical machine and then on machine M_2 . Let n_1 and n_2 denote the numbers of α -jobs and β -jobs, respectively ($n = n_1 + n_2$). We denote the processing time of each job processed on machine M_0 by p_0 . The processing time of each one the α -jobs on machine M_1 is denoted by p_1 , and of each one the β -jobs on machine M_2 is denoted by p_2 . Jobs of the same type are processed on the critical machine in batches. As mentioned, we assume batch availability, i.e. jobs become available for the second stage only when their entire batch is completed. Let S_1 and S_2 denote the setup times performed when starting a new batch (on the critical machine) of α -jobs and of β -jobs, respectively. Finally, let w_j denote the weight of job j , $j = 1, \dots, n$.

The objective functions studied here are minimum makespan, minimum total load, and minimum weighted flowtime. Let C_{ij} denote the completion time of job j on machine i , $i = 0,1,2$, $j = 1, \dots, n$. Let, $C_{max}^1 = \max\{C_{1j}; j = 1, \dots, n\}$: The maximum completion time on machine 1; $C_{max}^2 = \max\{C_{2j}; j = 1, \dots, n\}$: The maximum completion time on machine 2.

Thus, makespan is defined as:

$$C_{max} = \max \{C_{max}^1, C_{max}^2\}. \quad (1)$$

Total load is defined as:

$$\sum_i C_{max}^i = C_{max}^1 + C_{max}^2. \quad (2)$$

Weighted flow time is defined as:

$$\sum_{i=1}^2 \sum_{j=1}^n w_j C_{ij}. \quad (3)$$

(Note that w_j refers to job j , which is processed on *one* of the two dedicated machines. For convenience, we sort all the α -jobs prior to all the β -jobs; see Section 5.)

3. Makespan

In this section we focus on makespan minimization. As mentioned, in a batch *availability* setting, the actual completion time of each job is identical to the completion time of its batch. We demonstrate this setting in the following 8-job numerical *Example 1*. There are 2 jobs processed on machine M_1 , and 6 jobs processed on machine M_2 . The job processing times are: $p_0 = 5$, $p_1 = 6$, and $p_2 = 7$. The setup times are: $S_1 = 5$, and $S_2 = 4$. The optimal schedule consists of four batches: the first two are of β -jobs, the third is of α -jobs, and the last batch is of β -jobs; see Figure 1. The completion times of the last jobs on the three machines are 57, 60 and 64, respectively. Hence, the minimum makespan is 64.

Figure 1. Optimal schedule of Example 1 (Makespan)

We present a DP algorithm for this setting. The state variables are:

j_1 – the number of α -jobs already scheduled,

j_2 – the number of β -jobs already scheduled,

C_0 – the current completion time of the last job on the critical machine;

C_1 – the current completion time of the last batch on machine M_1 ;

C_2 – the current completion time of the last batch on machine M_2 ;

l – the type of the last processed job on the critical machine (either α or β ; we use γ to denote the initial state when no previous jobs were assigned).

q – the number of jobs assigned to the critical machine since the last completed batch.

Let $f(j_1, j_2, C_0, C_1, C_2, q, l)$ denote the minimum makespan that can be obtained by assigning the remaining jobs ($n_1 - j_1$ α -jobs and $n_2 - j_2$ β -jobs), given the current numbers of jobs assigned, their maximum completion time on machine M_0 , the maximum completion time of the last batch on machines M_1 and M_2 , the number of jobs of the current batch which have already been processed on machine M_0 (but not yet on the dedicated machine), and the type of the last job on the critical machine. Also, let $f_\alpha(j_1, j_2, C_0, C_1, C_2, q, l)$ denote the minimum makespan of the remaining jobs if the next scheduled job is an α -job. Similarly, $f_\beta(j_1, j_2, C_0, C_1, C_2, q, l)$ denotes the minimum makespan if the next scheduled job is a β -job.

Note that an optimal schedule may contain two or more consecutive batches of the same type. Therefore, let $h_\alpha(j_1, j_2, C_0, C_1, C_2, q, l)$ (and $h_\beta(j_1, j_2, C_0, C_1, C_2, q, l)$) denote the minimum makespan obtained when the last scheduled job on the critical machine was an α -

job (a β -job), and the decision is to start a new α -batch (a new β -batch). Alternatively, let $g_\alpha(j_1, j_2, C_0, C_1, C_2, q, l)$ (and $g_\beta(j_1, j_2, C_0, C_1, C_2, q, l)$) denote the minimum makespan obtained when the last scheduled job on the critical machine was an α -job (a β -job), and the decision is to continue with an α -batch (a β -batch).

For convenience, we define the following:

$s_{(\alpha)} = \max \{C_0, C_1\}$: the starting time of the new assigned batch on Machine 1, given that the last assigned job on the critical machine is an α -job;

$s_{(\beta)} = \max \{C_0, C_2\}$: the starting time of the new assigned batch on Machine 2, given that the last assigned job on the critical machine is a β -job;

Thus, for each machine we obtain the following recursion:

$$f_\alpha(j_1, j_2, C_0, C_1, C_2, q, l) = \begin{cases} \min \{g_\alpha(j_1, j_2, C_0, C_1, C_2, q, l), h_\alpha(j_1, j_2, C_0, C_1, C_2, q, l)\} & (\text{if } l = \alpha) \\ f(j_1 + 1, j_2, C_0 + S_1 + p_0, C_1, s_{(\beta)} + qp_2, 1, \alpha) & (\text{if } l = \beta) \end{cases} \quad (4)$$

$$f_\beta(j_1, j_2, C_0, C_1, C_2, q, l) = \begin{cases} \min \{g_\beta(j_1, j_2, C_0, C_1, C_2, q, l), h_\beta(j_1, j_2, C_0, C_1, C_2, q, l)\} & (\text{if } l = \beta) \\ f(j_1, j_2 + 1, C_0 + S_2 + p_0, s_{(\alpha)} + qp_1, C_2, 1, \beta) & (\text{if } l = \alpha) \end{cases} \quad (5)$$

where,

$$g_\alpha(j_1, j_2, C_0, C_1, C_2, q, l) = f(j_1 + 1, j_2, C_0 + p_0, C_1, C_2, q + 1, \alpha) \quad (6)$$

$$h_\alpha(j_1, j_2, C_0, C_1, C_2, q, l) = f(j_1 + 1, j_2, C_0 + S_1 + p_0, s_{(\alpha)} + qp_1, C_2, 1, \alpha) \quad (7)$$

$$g_\beta(j_1, j_2, C_0, C_1, C_2, q, l) = f(j_1, j_2 + 1, C_0 + p_0, C_1, C_2, q + 1, \beta) \quad (8)$$

$$h_\beta(j_1, j_2, C_0, C_1, C_2, q, l) = f(j_1, j_2 + 1, C_0 + S_2 + p_0, C_1, s_{(\beta)} + qp_2, 1, \beta) \quad (9)$$

The recursive equation is:

$$f(j_1, j_2, C_0, C_1, C_2, q, l) = \min \{f_\alpha(j_1, j_2, C_0, C_1, C_2, q, l), f_\beta(j_1, j_2, C_0, C_1, C_2, q, l)\} \quad (10)$$

The boundary conditions are:

$$f(n_1, n_2, C_0, C_1, C_2, q, l) = \begin{cases} \max \{s_{(\alpha)} + qp_1, C_2\} & (\text{if } l = \alpha, \text{ for every } C_0, C_1, C_2 \text{ and } q) \\ \max \{C_1, s_{(\beta)} + qp_2\} & (\text{if } l = \beta, \text{ for every } C_0, C_1, C_2 \text{ and } q) \end{cases} \quad (11)$$

$$f(j_1, j_2, C_0, C_1, C_2, q, l) = \infty \text{ if either } j_1 > n_1 \text{ or } j_2 > n_2. \quad (12)$$

The optimal solution is given by: $f(0, 0, 0, 0, 0, 0, \gamma)$.

Running Time: It is clear that an upper bound on the number of values of j_1 and j_2 are n_1 and n_2 , respectively. C_0 is determined by the number of setups of both types (α and β). Hence, for a given n_1 and n_2 , there are at most $n_1 n_2$ different values of C_0 . We note that on the standard graph representation of the makespan on a flowshop, the makespan on the first dedicated machine, C_1 , is identical to the length of its "critical path". Hence, there exists a "critical batch" (on the critical path), such that there is no idle time prior to this batch on machine M_0 , and no idle time after this batch on machine M_1 . Let t_0 , be the total processing time on machine M_0 of the batches scheduled not later than the critical batch (including

their setup times). Let t_1 be the total processing time on machine M_1 of batches scheduled not earlier than the critical batch. We observe that $C_1 = t_0 + t_1$. The number of values for t_0 is bounded by $O((n_1 + n_2)n_1n_2)$, and the number of values for t_1 is bounded by n_1 . It follows that the number of possible values of C_1 is bounded by $O((n_1 + n_2)n_1^2n_2)$. [Symmetrically, we focus on the critical path of the batches assigned to the critical machine and the second dedicated machine. Now t_0 is the total processing time on machine M_0 not later to the critical batch on the critical path of the makespan of machines M_0 and M_2 . Let t_2 be the total processing time on machine M_2 of batches scheduled not earlier than the critical batch ($C_2 = t_0 + t_2$). A similar analysis leads to $O((n_1 + n_2)n_1n_2^2)$ being an upper bound on the number of possible values of C_2 .] Note that q can be fully determined (in constant time) from C_0, C_1, C_2, p_1 and p_2 , and therefore does not add to the total complexity. Finally, the running time per one iteration of the DP is clearly a constant. We thus conclude that the total running time is $O((n_1 + n_2)^2n_1^5n_2^5)$.

Under the very reasonable assumption of a fixed upper bound on the number of batches, (see [1]), the number of possible values of C_0 is reduced to a constant, the number of possible values of C_1 is reduced to $O((n_1 + n_2)n_1)$, and the number of possible values of C_2 is reduced to $O((n_1 + n_2)n_2)$. Hence, the total running time becomes $O((n_1 + n_2)^2n_1^2n_2^2) = O(n^6)$. [Comment: As in many other batch scheduling problems with identical jobs, the running time is polynomial in the number of jobs, but not in the input size. Note that the input for the problem contains 7 parameters only: $n_1, n_2, p_0, p_1, p_2, S_1, S_2$.]

Table 1. Makespan minimization: average and worst case running times (milliseconds) as a function of n

n	Average (ms)	Worst case (ms)
16	0.1	1
17	138.92	406
18	542.24	1299
19	2056.24	3757
20	5333.36	9975
21	17588	43436
22	25974.72	86746
23	136696.08	260767
24	284775	794353

We performed numerical tests in order to evaluate the efficiency of the DP algorithm. We solved instances of the following sizes: $n = 16, 17, \dots, 24$. For each instance, the number of jobs of type 1 (n_1) was generated uniformly in the interval $[1, n]$, and the number of jobs of type 2 (n_2) was defined as $n - n_1$. The setup times S_1 and S_2 , and the processing times p_0, p_1 and p_2 were generated uniformly in the interval $[1, 10]$. For each problem size, 25 instances were generated and solved. The running times were measured, and the average and worst case times are reported in Table 1. [The C program was executed on Macintosh 2.8 GHz, Intel Core 2 Duo.] The DP appears to perform well on these

instances. The average running time required for solving a 20-job problem did not exceed 5.5 seconds. For a 23-job problem, the average running time increased to 136.7 seconds, and for a 24-job problem it became 284.8 seconds. The polynomial regression demonstrated in Figure 2 indicates that, indeed, the worst case running time is proportional to $O(n^6)$.

Figure 2. Makespan minimization: Worst case running times (in seconds) as a function of the number of jobs (n)

4. Total Load

In the case of total load minimization, we minimize the sum of the maximal completion times on machine M_1 and on machine M_2 . As mentioned above, the general version of the problem solved here (i.e. assuming *job-dependent* processing times), was shown by [3] to be strongly NP-hard. We focus on the special case of (machine-dependent) identical job processing times. The following 8-job numerical *Example 2* demonstrates this setting. There are 4 jobs processed on machine M_1 , and 4 jobs processed on machine M_2 . The job processing times are: $p_0 = 5$, $p_1 = 6$, and $p_2 = 7$. The setup times are: $S_1 = 5$, and $S_2 = 4$. The optimal schedule consists of five batches: the first two are of β -jobs, and the three following are of α -jobs; see Figure 3. The completion times of the last jobs on the three machines are 63, 69 and 42, respectively. The optimal total load is $\sum_i C_{max}^i = 111$.

Figure 3. Optimal schedule of Example 2 (Total Load)

We present a DP algorithm for this setting. The state variables defined in Section 3 ($j_1, j_2, C_0, C_1, C_2, q, l$) are still valid. Let $f(j_A, j_B, C_0, C_A, C_B, q, l)$ denote the minimal total load that can be obtained by assigning the remaining jobs ($n_1 - j_1$ α -jobs and $n_2 - j_2$ β -jobs), given the current numbers of jobs assigned, their maximum completion time on machine M_0 , the maximum completion time of the last batch on machines M_1 and M_2 , the number of jobs that will be processed in the coming batch (already scheduled on machine M_0), and the type of the last job on the critical machine.

As in Section 3, the starting times of the new assigned batch on machines M_1 and M_2 are $s_{(\alpha)} = \max \{C_0, C_1\}$ and $s_{(\beta)} = \max \{C_0, C_2\}$, respectively.

Thus, for each machine we obtain the following recursion:

$$f_\alpha(j_1, j_2, C_0, C_1, C_2, q, l) = \begin{cases} \min \{g_\alpha(j_1, j_2, C_0, C_1, C_2, q, l), h_\alpha(j_1, j_2, C_0, C_1, C_2, q, l)\} & (\text{if } l = \alpha) \\ f(j_1 + 1, j_2, C_0 + S_1 + p_0, C_1, s_{(\beta)} + qp_2, 1, \alpha) & (\text{if } l = \beta) \end{cases} \quad (13)$$

$$f_\beta(j_1, j_2, C_0, C_1, C_2, q, l) = \begin{cases} \min \{g_\beta(j_1, j_2, C_0, C_1, C_2, q, l), h_\beta(j_1, j_2, C_0, C_1, C_2, q, l)\} & (\text{if } l = \beta) \\ f(j_1, j_2 + 1, C_0 + S_2 + p_0, s_{(\alpha)} + qp_1, C_2, 1, \beta) & (\text{if } l = \alpha) \end{cases} \quad (14)$$

where,

$$g_\alpha(j_1, j_2, C_0, C_1, C_2, q, l) = f(j_1 + 1, j_2, C_0 + p_0, C_1, C_2, q + 1, \alpha), \quad (15)$$

$$h_\alpha(j_1, j_2, C_0, C_1, C_2, q, l) = f(j_1 + 1, j_2, C_0 + S_1 + p_0, s_{(\alpha)} + qp_1, C_2, 1, \alpha), \quad (16)$$

$$g_\beta(j_1, j_2, C_0, C_1, C_2, q, l) = f(j_1, j_2 + 1, C_0 + p_0, C_1, C_2, q + 1, \beta), \quad (17)$$

$$h_\beta(j_1, j_2, C_0, C_1, C_2, q, l) = f(j_1, j_2 + 1, C_0 + S_2 + p_0, C_1, s_{(\beta)} + qp_2, 1, \beta), \quad (18)$$

and the recursive equation is:

$$f(j_1, j_2, C_0, C_1, C_2, q, l) = \min \{f_\alpha(j_1, j_2, C_0, C_1, C_2, q, l), f_\beta(j_1, j_2, C_0, C_1, C_2, q, l)\}. \quad (19)$$

The boundary conditions are:

$$f(n_1, n_2, C_0, C_1, C_2, q, l) = \begin{cases} s_{(\alpha)} + qp_1 + C_2 & (\text{if } l = \alpha \text{ and for every } C_0, C_1, C_2 \text{ and } q) \\ C_1 + s_{(\beta)} + qp_2 & (\text{if } l = \beta \text{ and for every } C_0, C_1, C_2 \text{ and } q) \end{cases} \quad (19)$$

$$f(j_1, j_2, C_0, C_1, C_2, q, l) = \infty \text{ if either } j_1 > n_1 \text{ or } j_2 > n_2. \quad (20)$$

The optimal solution is given by: $f(0, 0, 0, 0, 0, 0, \gamma)$.

Running Time: As in the previous DP, the total running time is $O((n_1 + n_2)^2 n_1^5 n_2^5)$. Again, for a given upper bound on the number of batches, the total running time is reduced to $O((n_1 + n_2)^2 n_1^2 n_2^2) = O(n^6)$.

As in the previous section, we performed numerical tests in order to evaluate the efficiency of the DP algorithm. We considered the following numbers of jobs: $n = 15, 16, \dots, 24$. For each instance, $n_1, n_2, S_1, S_2, p_0, p_1$ and p_2 were generated as in Section 3. Again, for each problem size, 25 instances were generated and solved. The running times were measured, and the average and worst case times are reported in Table 2. It appears that our proposed DP is applicable and efficient for medium size instances. The average running time required for solving a 20-job problem did not exceed 9 seconds. The solution of larger problems required much larger effort: The average running time for a 23-job problem was 197 seconds, and for a 24-job problem it became 462 seconds. As expected, the increase of the running times is proportional to $O(n^6)$; see Figure 4 for a polynomial regression reflecting the worst case running times.

Table 2. Total load minimization: average and worst case running times (milliseconds) as a function of n

n	Average (ms)	Worst case (ms)
15	0	0
16	31.4	198
17	323	686
18	936.5	2000
19	3603.4	5752
20	8807.1	17091
21	38753.4	45577
22	59224.1	140574
23	197313.2	431090
24	462186.7	1094305

Figure 4. Total load minimization: Worst case running times (in seconds) as a function of the number of jobs (n)

5. Weighted Flowtime

In this section we introduce an exact solution algorithm (DP) for the objective of minimum weighted flowtime, i.e. we minimize $\sum_{i=1}^2 \sum_{j=1}^n w_j C_{ij}$. The assumption of identical (machine-dependent) job processing times still holds. The following 8-job numerical *Example 3* demonstrates this setting. There are 3 α -jobs and 5 β -jobs. The job processing times are: $p_0 = 3$, $p_1 = 6$, and $p_2 = 4$. The setup times are: $S_1 = 7$, and $S_2 = 4$. The weights (w_j) for the three α -jobs are: 1, 2 and 3; and the weights for the five β -jobs are: 1, 2, 3, 4 and 5. Jobs are assumed to be sorted in a non-decreasing order of their weights. The optimal schedule consists of 6 batches as depicted in Figure 5: The first β -batch contains jobs 5 and 4, the second β -batch contains jobs 3 and 2, the first α -batch contains job 3, the second α -batch contains job 2, the last β -batch contains job 1, and the last α -batch contains job 1. The optimal weighted flowtime is $\sum_j w_j C_j = 616$.

Figure 5. Optimal schedule of Example 3 (Weighted Flowtime)

We introduce a DP algorithm for this setting. As for the previous settings, the algorithm requires the following state variables: j_1 , j_2 , C_0 , C_1 , C_2 , q and l . For given j_1 , j_2 , C_0 , C_1 , C_2 , q and l , let $f(j_A, j_B, C_0, C_A, C_B, q, l)$ denote the minimal weighted flowtime obtained by assigning the remaining $n_1 - j_1$ α -jobs and $n_2 - j_2$ β -jobs. As in Section 3, the starting times of the new assigned batch on Machine M_1 and on Machine M_2 are $s_{(\alpha)} = \max\{C_0, C_1\}$ and $s_{(\beta)} = \max\{C_0, C_2\}$, respectively.

Note that unlike the previous settings, in the case of weighted flowtime, the order of jobs of a given type is crucial. In particular, the following property is easily proved by a standard pair-wise interchange argument:

Property 1: An optimal schedule exists such that the α -jobs and the β -jobs are scheduled in a non-increasing order of w_j .

Following Property 1 (and for convenience) we set the job index (j) as follows: the first n_1 jobs are the α -jobs, sorted in a non-increasing order of w_j ; the following n_2 jobs are the β -jobs, also sorted in a non-increasing order of w_j . We introduce the following recursion:

$$f_\alpha(j_1, j_2, C_0, C_1, C_2, q, l) = \begin{cases} \min\{g_\alpha(j_1, j_2, C_0, C_1, C_2, q, l), h_\alpha(j_1, j_2, C_0, C_1, C_2, q, l)\} & (\text{if } l = \alpha) \\ (s_{(\beta)} + qp_2) \sum_{i=0}^{q-1} w_{n_1+j_2-i} + f(j_1 + 1, j_2, C_0 + S_1 + p_0, C_1, s_{(\beta)} + qp_2, 1, \alpha) & (\text{if } l = \beta) \end{cases} \quad (21)$$

$$f_\beta(j_1, j_2, C_0, C_1, C_2, q, l) = \begin{cases} \min\{g_\beta(j_1, j_2, C_0, C_1, C_2, q, l), h_\beta(j_1, j_2, C_0, C_1, C_2, q, l)\} & (\text{if } l = \beta) \\ (s_{(\alpha)} + qp_1) \sum_{i=0}^{q-1} w_{j_1-i} + f(j_1, j_2 + 1, C_0 + S_2 + p_0, s_{(\alpha)} + qp_1, C_2, 1, \beta) & (\text{if } l = \alpha) \end{cases} \quad (22)$$

where,

$$g_\alpha(j_1, j_2, C_0, C_1, C_2, q, l) = f(j_1 + 1, j_2, C_0 + p_0, C_1, C_2, q + 1, \alpha) \quad (23)$$

$$h_\alpha(j_1, j_2, C_0, C_1, C_2, q, l) = (s_{(\alpha)} + qp_1) \sum_{i=0}^{q-1} w_{j_1-i} + f(j_1 + 1, j_2, C_0 + S_1 + p_0, s_{(\alpha)} + qp_1, C_2, 1, \alpha) \quad (24)$$

$$g_\beta(j_1, j_2, C_0, C_1, C_2, q, l) = f(j_1, j_2 + 1, C_0 + p_0, C_1, C_2, q + 1, \beta) \quad (25)$$

$$h_{\beta}(j_1, j_2, C_0, C_1, C_2, q, l) = (s_{(\beta)} + qp_2) \sum_{i=0}^{q-1} w_{n_1+j_2-i} + f(j_1, j_2 + 1, C_0 + S_2 + p_0, C_1, s_{(\beta)} + qp_2, 1, \beta) \quad (26)$$

The recursive equation is:

$$f(j_1, j_2, C_0, C_1, C_2, q, l) = \min \{f_{\alpha}(j_1, j_2, C_0, C_1, C_2, q, l), f_{\beta}(j_1, j_2, C_0, C_1, C_2, q, l)\} \quad (27)$$

The boundary conditions are:

$$\begin{cases} f(n_1, n_2, C_0, C_1, C_2, q, l) = \\ (s_{(\alpha)} + qp_1) \sum_{i=0}^{q-1} w_{j_1-i} & (\text{if } l = \alpha \text{ and for every } C_0, C_1, C_2 \text{ and } q) \\ (s_{(\beta)} + qp_2) \sum_{i=0}^{q-1} w_{n_1+j_2-i} & (\text{if } l = \beta \text{ and for every } C_0, C_1, C_2 \text{ and } q) \end{cases} \quad (28)$$

$$f(j_1, j_2, C_0, C_1, C_2, q, l) = \infty \text{ if either } j_1 > n_1 \text{ or } j_2 > n_2. \quad (29)$$

The optimal solution is given by: $f(0, 0, 0, 0, 0, 0, \gamma)$.

Running Time: The running time remains (as in the previous sections) $O((n_1 + n_2)^2 n_1^5 n_2^5)$, and assuming an upper bound on the number of batches it is $O((n_1 + n_2)^2 n_1^2 n_2^2) = O(n^6)$.

Again, we tested the DP numerically, for $n = 16, 17, \dots, 23$. $n_1, n_2, S_1, S_2, p_0, p_1$ and p_2 were generated as in Section 3. Again, for each problem size, 25 instances were generated and solved. The average and the worst case running times are reported in Table 3. Figure 6 reflects the worst case running times as a function of n (which, as expected, is very close to $O(n^6)$).

Table 3. Weighted flowtime minimization: average and worst case running times (milliseconds) as a function of n

n	Average (ms)	Worst case (ms)
16	57.24	216
17	269.72	589
18	579.72	1715
19	2819.08	5200
20	4785.68	14355
21	15050.12	40606
22	76962.04	129183
23	167618.4	397709

Figure 6. Weighted flowtime minimization: Worst case running times (in seconds) as a function of the number of jobs (n)

6. A heuristic procedure

Although the DP algorithms introduced above were shown numerically to be able to obtain the optimal solutions for medium size problems in reasonable times, it is clear that for larger instances, faster solution procedures are required. In this section we introduce a simple and efficient heuristic algorithm for makespan minimization based on *equal allocation* of jobs to batches. We also propose a tight lower bound on the optimal makespan, and a report of numerical tests is provided.

In the initial step of the heuristic (see below), we evaluate the cost of splitting an α -batch into two α -batches. If splitting does not lead to a lower cost, we increase the original batch by two jobs, and evaluate again the cost of splitting. This procedure is repeated until splitting becomes worthwhile. The procedure returns the maximal (single) α -batch size, (i.e. larger batches should be split). Let $C_{max}^{\alpha}(bs)$ denote the cost of a single α -batch *before* splitting. Similarly, let $C_{max}^{\alpha}(as)$ denote the cost of two (half-size) α -batches *after* splitting.

Procedure *Max_Size* (α -batch):

Input: n_1, p_0, p_1, S_1

size = 0

do

size = size + 2

$C_{max}^{\alpha}(bs) = S_1 + size * (p_0 + p_1)$ /* The cost of a single batch */

$C_{max}^{\alpha}(as) = \max \left\{ 2S_1 + size * p_0, S_1 + \frac{size}{2}(p_0 + p_1) \right\} + \frac{size}{2} p_1$
/* The cost of two batches */

```

while ( $C_{max}^\alpha(bs) \leq C_{max}^\alpha(as)$  and ( $size \leq n_1$ ))
 $size_\alpha = size - 2$ 
return ( $size_\alpha$ )

```

A similar procedure (Procedure *Max_Size* (β -batch)) returns the maximal β -batch size (which is not worthwhile to split). We denote this value by $size_\beta$.

Next we describe the procedure *Calculate_Cost*, which calculates the makespan by equal allocation of jobs to batches, given a number of batches (r). The input for this procedure contains $r, n_1, n_2, p_0, p_1, p_2, S_1, S_2, first_batch_type, size_\alpha$ and $size_\beta$. The procedure consists of two steps. In the first step, we consider a sequence of r batches of alternating types, starting with *first_batch_type* (either α or β). For example, for $r = 5$ with *first_batch_type* = α , the sequence of batches is: $(\alpha, \beta, \alpha, \beta, \alpha)$. The batch sizes are obtained by equal allocation. In the above example, the size of each of the α -batches is equal to $n_1/3$, and the size of each of the β -batches is equal to $n_2/2$. Clearly, the batch sizes are rounded to the closest integer while allocating all jobs. In the second step, the size of the α -batches is compared to $size_\alpha$. If the size of the α -batches is larger than $size_\alpha$, these batches are split into smaller α -batches of size $size_\alpha$ (with a possible smaller last batch). This procedure is repeated for the β -batches. The output of *Calculate_Cost* is the makespan of batch sequence obtained after the second step. The formal algorithm (*Min_Makespan*) for the heuristics is provided below:

Algorithm *Min_Makespan*:

```

Input:  $n_1, n_2, p_0, p_1, p_2, S_1, S_2$ 
 $min\_makespan = \infty$ 
 $size_\alpha = Max\_Size(\alpha\text{-batch})$ 
 $size_\beta = Max\_Size(\beta\text{-batch})$ 
 $max\_iteration = (\min(n_1, n_2) \cdot 2 + 1)$  /* maximum number of alternating batches
                                         assuming equal allocation prior to splitting */
for  $r = 2$  to  $max\_iteration$ 
   $makespan = Calculate\_Cost(r, n_1, n_2, p_0, p_1, p_2, S_1, S_2, \alpha, size_\alpha, size_\beta)$ 
  if  $makespan \leq min\_makespan$ 
     $min\_makespan = makespan$ 
   $makespan = Calculate\_Cost(r, n_1, n_2, p_0, p_1, p_2, S_1, S_2, \beta, size_\alpha, size_\beta)$ 
  if  $makespan \leq min\_makespan$ 
     $min\_makespan = makespan$ 
return ( $min\_makespan$ ).

```

Running Time: An upper bound on the running time of procedure *Max_Size* is $O(n)$. Procedure *Calculate_Cost* also requires in the worst case $O(n)$ time. In algorithm *Min_Makespan*, procedure *Calculate_Cost* is performed no more than $O(n)$ times. Thus, the total running time of the algorithm does not exceed $O(n^2)$.

The following numerical example demonstrates the general steps of the above algorithm:

Example 4: Assume $n_1 = 50, n_2 = 150$. We first call *Max_Size* (α -batch) and *Max_Size* (β -batch), and obtain $size_\alpha = 25$ and $size_\beta = 10$, respectively. Next we enter

the main loop of *Min_Makespan*, where we call procedure *Calculate_Cost*. For $r = 5$, for example, the resulting schedule based on equal allocation consists of the following five batches: the first (β -type) contains 50 jobs, the second is an α -type batch of size 25, then another β -type batch of 50 jobs, another α -type batch of 25 jobs, and then the last β -type batch containing 50 jobs. Note that the first batch is of β -type since $type = \beta$. Since the size of the β -type batches exceed *Max_Size* (which is 10), the β batches are further split, and the final schedule consists of 15 β -type batches (of size 10) and 2 α -type batches (of size 25); see Figures 7 and 8. [We note that if equal allocation does not lead to integer batch sizes, the algorithm uses "as close as possible" to equal allocation in a non-increasing order of batch sizes. For example, in the previous example, if the α -jobs ($n_1 = 50$) need to be allocated to 3 batches, the resulting batch sizes and their order are: 17, 17 and 16.]

Figure 7. A schedule based on equal allocation prior to batch splitting (Example 4)

Figure 8. A schedule based on equal allocation after batch splitting (Example 4)

Lower Bound: We introduce three lower bounds on the optimal makespan. The first considers the total load on the critical machine (of both α and β jobs and *assuming* at least one batch of each type), and at least one additional job (beyond the maximal completion time on the critical machine) on one of the dedicated machines:

$$LB_1 = S_1 + S_2 + np_0 + \min\{p_1, p_2\} \quad (30)$$

A second lower bound consists of the total load on the first dedicated machine assuming (i) idle time prior to processing due to one (necessary) α -batch of a single job on the critical machine, and (ii) no idle time between consecutive jobs on the dedicated machine:

$$LB_2 = S_1 + p_0 + n_1 p_1 \quad (31)$$

The third lower bound is obtained similarly by calculating the total load on the second dedicated machine:

$$LB_3 = S_2 + p_0 + n_2 p_2 \quad (32)$$

The lower bound used for the evaluation of the heuristic results is the maximum of the above:

$$LB = \max \{LB_1, LB_2, LB_3\} \quad (33)$$

We tested the performance of the heuristic on large size instances. The following numbers of jobs were considered: $n = 100, 200, \dots, 1000$. The parameters $n_1, n_2, S_1, S_2, p_0, p_1$ and p_2 were generated as in Section 3 (n_1 was generated uniformly in the interval $[1, n]$, $n_2 = n - n_1$; and S_1, S_2, p_0, p_1 and p_2 were generated uniformly in the interval $[1, 10]$). For each problem size, 50 instances were generated, solved by *Min_Makespan* (to obtain $C_{max}^{heuristic}$), and the lower bound (LB) was computed. The average and worst case optimality gaps ($\frac{(C_{max}^{heuristic} - LB)}{LB} \cdot 100\%$) are reported in Table 4.

The results indicate that both the heuristic and the lower bound are very accurate. The average optimality gap for 100 job problems was about 4%. The average optimality gap for problems of 200 jobs or more did not exceed 3%. For problems of 700 jobs or more, the average optimality gap did not exceed 1.7%. The average running time per one 1000-job problem was 73 milliseconds. Based on both its accuracy and efficiency, it appears that the heuristic is applicable for large-scale real-life settings.

Table 4. *Min_Makespan* Algorithm: average and worst case optimality gaps
 $\left(\frac{(C_{max}^{heuristic} - LB)}{LB} \cdot 100\% \right)$

n	Average optimality gap (%)	Worst case optimality gap (%)
100	4.03	36.70
200	2.73	20.71
300	2.47	32.49
400	2.28	12.9
500	1.87	14.04
600	2.31	16.63
700	1.68	13.2
800	1.52	10.13
900	1.56	9.57
1000	1.57	13.57

7. Conclusion and future research

We developed dynamic programming algorithms for solving a two-stage flowshop, with one critical machine at stage 1 and two dedicated machines at stage 2. Jobs are assumed to have identical (but machine dependent) processing times. Jobs may be processed in batches, where a setup time is required when starting a new batch on the critical machine.

We assume batch availability; jobs cannot be processed in the dedicated machine until the processing of their entire batch is completed on the critical machine. The objective functions considered were: minimum makespan, minimum total load and minimum weighted flowtime. The DP algorithms perform well on problems of medium size, and their complexity becomes $O(n^6)$ under the reasonable assumption of a fixed upper bound on the number of batches.

As mentioned, the DP algorithms produce optimal solutions for medium-size problems in reasonable time, but it is clear that for large instances, heuristic algorithms are needed. We developed and tested numerically a simple heuristic for makespan minimization based on *equal allocation* of jobs to batches. Limited numerical tests indicated that similar heuristics are less appropriate for total load and weighted flowtime minimization. [For example a heuristic based on equal allocation for 1000- job total load minimization problems, produced an average optimality gap of 13.27% and worse case optimality gap of 64.84%.]

The main question for future research remains whether the computational effort required for solving the above problems can be reduced. In addition, it seems challenging to develop both more accurate heuristics and tight lower bounds for total load and weighted flowtime minimization. Finally, future research may also focus on solving problems of the same setting (a two-stage flowshop, identical jobs and batching), with different objectives, such as minimizing maximum lateness and minimizing the number of tardy jobs.

References

- [1] Cheng T.C.E. and Kovalyov M. Y., An exact algorithm for batching and scheduling two part types in a mixed shop: A technical note, *International Journal of Production Economics*, **55**, 1998, 53-56.
- [2] Cheng T.C.E., Kovalyov M.Y. and Chakhlevich K. N., Batching in a two-stage flowshop with dedicated machines in second stage. *IIE Transactions*, **36**, 2004, 87-93.
- [3] Cheng T.C.E., Lin B.M.T. and Tian Y., Scheduling of a two-stage differentiation flowshop to minimize weighted sum of machine completion times, *Computers & Operations Research*, **36**, 2009, 3031-3040.
- [4] Ching C. Y., Liao, Wu C. J., Batching to minimize total production time for two part types, *International Journal of Production Economics*, **48**, 1997, 63-72.
- [5] Drobouchevitch I. G., Strusevich V. A., Heuristics for the two-stage job shop scheduling problem with a bottleneck machine, *European Journal of Operational Research*, **123**, 2000, 229-240.
- [6] Kyparisis G. J., Koulamas, C., Flow shop and open shop scheduling with a critical machine and two operations per job, *European Journal of Operational Research*, **127**, 2000, 120-125.
- [7] Lin H.-T., Liao, C. -J., A case study in a two-stage hybrid flow shop with setup time and dedicated machines, *International Journal of Production Economics*, **86**, 2003, 133-143.
- [8] Linn R., Zhang W., Hybrid flow shop scheduling: A survey, *Computers & Industrial Engineering*, **37**, 1999, 57-61.

- [9] Mosheiov G., Yovel U., Comments on “Flow shop and open shop scheduling with critical machine and two operations per job”, *European Journal of Operational Research*, **157**, 2004, 257-261.
- [10] Oguz C., Ercan, M. F., A Genetic Algorithm for Hybrid Flow-Shop Scheduling with Multiprocessor Tasks, *Journal of Scheduling*, **8**, 2005, 323-351.
- [11] Ruiz, R., Vazquez-Rodriguez J.A., The hybrid flow shop scheduling problem, *European Journal of Operational Research*, **205**, 2010, 1-18.

Received August, 2011