RAIRO Operations Research
RAIRO Oper. Res. **35** (2001) 107-115

ON MINIMIZING TOTAL TARDINESS IN A SERIAL BATCHING PROBLEM

Philippe Baptiste[1] and Antoine Jouglet[1]

Communicated by Ph. Chrétienne

Abstract. We study the problem of scheduling jobs on a serial batching machine to minimize total tardiness. Jobs of the same batch start and are completed simultaneously and the length of a batch equals the sum of the processing times of its jobs. When a new batch starts, a constant setup time s occurs. This problem $1|s\text{-}batch|\sum T_i$ is known to be NP-Hard in the ordinary sense. In this paper we show that it is solvable in pseudopolynomial time by dynamic programming.

Keywords: Scheduling, batching, dynamic programming, total tardiness.

Mathematics Subject Classification. 90B35, 90C39, 90C27.

Introduction

In this paper, we study the situation where n jobs $\{J_1, \ldots, J_n\}$ have to be scheduled on a **serial batching machine**. Each job J_i is described by a processing time p_i and a due date d_i (all numerical values used in this paper being integer). All jobs of the same batch start and are completed simultaneously, *i.e.*, at the starting time (respectively at the completion time) of the batch. On a serial batching machine, the length of a batch equals the sum of the processing times of its jobs. The size of the batch, *i.e.*, the number of jobs in the batch is not known in advance. When a new batch starts, a constant setup time s occurs.

Several other type of batching machines have been studied in the literature. In particular on **parallel batching machine**, there are at most b jobs per batch and

Received May, 2000. Accepted November, 2000.

[1] CNRS, UMR 6599 HeuDiaSyC, Université de Technologie de Compiègne, Centre de Recherches de Royallieu, 60205 Compiègne Cedex, France.

TABLE 1. Overview of the complexity results.

Problem	Complexity	References
$1\|s\text{-}batch\|L_{\max}$	$O(n^2)$	[13]
$1\|s\text{-}batch\|\sum U_i$	$O(n^3)$	[6]
$1\|s\text{-}batch\|\sum w_i U_i$	binary NP-Hard, $O(n^2 \sum w_i)$	[6]
$1\|s\text{-}batch\|\sum C_i$	$O(n \log n)$	[7]
$1\|s\text{-}batch\|\sum w_i C_i$	unary NP-Hard	[1]
$1\|s\text{-}batch\|\sum T_i$	binary NP-Hard	[8]

the length of a batch is the largest processing time of its jobs (in more complex models, a job can require several units of the batching machine at the same time). We refer to [1,4,5,9,11–13] for extended reviews on pure batch scheduling problems and on extensions (*e.g.* scheduling group of jobs with group-dependent setup times, jobs requiring several machines throughout their execution, etc.). Complexity results for serial batching problems are summarized in Table 1. When processing times are equal, all problems, except the weighted total tardiness one, become polynomial even if jobs have arbitrary release dates [2]. These results leave open the exact status of the total tardiness problem $1|s\text{-}batch|\sum T_i$. Is it unary NP-Hard or can it be solved in pseudo-polynomial time? We show that it is solvable in $O(n^{11} \max(\max_i p_i, s)^7)$ by dynamic programming. This algorithm is closely related to the pseudo-polynomial algorithm of Lawler for the $1||\sum T_i$ problem [3, 10].

The remaining sections of the paper are organized as follows. Several dominance properties are stated in Section 1. The dynamic programming algorithm itself is described in Section 2. Finally, Section 3 summarizes the main results, gives a few conclusions, and describes some fruitful areas for further research.

1. DOMINANCE PROPERTY

From now on, we suppose that jobs are sorted in non-decreasing order of processing times, *i.e.*, $p_1 \leq p_2 \leq \ldots \leq p_n$. Given a solution schedule, we note $\mathcal{B}(i)$ the batch that contains the job J_i and for any batch $\mathcal{B}$, we note $S_{\mathcal{B}}$ and $C_{\mathcal{B}}$ the starting time and the completion time of the batch. To simplify the presentation, we assume that due dates are distinct.

Theorem 1.1. *There exists an optimal schedule such that, for any jobs J_i, J_j with $d_i < d_j$, if J_j is processed strictly before $\mathcal{B}(n)$ then J_i is processed before or in $\mathcal{B}(n)$.*

Proof. Let $\mathcal{S}$ be an optimal schedule on which the completion time of $\mathcal{B}(n)$ is maximal and on which the number of jobs scheduled in batches before $\mathcal{B}(n)$

is minimal. Assume that there exist two jobs J_i, J_j with $d_i < d_j$, such that J_j is processed strictly before $\mathcal{B}(n)$ and J_i is processed strictly after $\mathcal{B}(n)$.

We first show that $d_i > d_n$. If this is not the case, so if $d_i \leq d_n$, let us exchange J_i and J_n, *i.e.*, remove J_n from $\mathcal{B}(n)$ and add J_i (the batch is completed at $C_{\mathcal{B}(n)} - p_n + p_i$) and symmetrically, remove J_i from $\mathcal{B}(i)$ and add J_n instead (the batch starts at $S_{\mathcal{B}(i)} - p_n + p_i$. It is easy to see that because $p_i \leq p_n$, the batches of all jobs except J_n are not completed later than before. Moreover, the cost Δ of the exchange for J_i and J_n is:

$$\begin{aligned}\Delta &\leq \max(0, C_{\mathcal{B}(i)} - d_n) - \max(0, C_{\mathcal{B}(n)} - d_n) \\ &\quad + \max(0, C_{\mathcal{B}(n)} - p_n + p_i - d_i) - \max(0, C_{\mathcal{B}(i)} - d_i).\end{aligned}$$

If $C_{\mathcal{B}(i)} \geq d_n$ then we have:

$$\Delta \leq d_i - d_n + \max(0, C_{\mathcal{B}(n)} - p_n + p_i - d_i) - \max(0, C_{\mathcal{B}(n)} - d_n).$$

Either $C_{\mathcal{B}(n)} - p_n + p_i - d_i \leq 0$, which leads to $\Delta \leq 0$ or $C_{\mathcal{B}(n)} - p_n + p_i - d_i \geq 0$ and therefore, $\Delta \leq C_{\mathcal{B}(n)} - d_n - \max(0, C_{\mathcal{B}(n)} - d_n) \leq 0$. Now consider the case where $C_{\mathcal{B}(i)} < d_n$ then

$$\Delta \leq \max(0, C_{\mathcal{B}(n)} - p_n + p_i - d_i) - \max(0, C_{\mathcal{B}(i)} - d_i) \leq 0.$$

We have shown that the resulting schedule is still optimal and moreover, the batch containing J_n is completed later than before; which contradicts our hypothesis.

We can now assume that $d_n < d_i < d_j$. First notice that if $d_j \geq C_{\mathcal{B}(n)}$ then J_j is on-time and it is still on time if it is removed from $\mathcal{B}(j)$ and put into $\mathcal{B}(n)$ (batches between $\mathcal{B}(j)$ and $\mathcal{B}(n)$ are scheduled p_j time units earlier). (1) The total tardiness is not increased, (2) the batch containing J_n is completed at the same time than before and (3) one less job is scheduled before this batch. This contradict our initial hypothesis hence, $d_j < C_{\mathcal{B}(n)}$. Second, we propose to exchange J_n and J_i. Since $p_i \leq p_n$, batches between $\mathcal{B}(n)$ and $\mathcal{B}(i)$ are not scheduled later than before and then, the cost of the exchange can be upper bounded by Δ'.

$$\begin{aligned}\Delta' &= \max(0, C_{\mathcal{B}(i)} - d_n) - \max(0, C_{\mathcal{B}(n)} - d_n) \\ &\quad + \max(0, C_{\mathcal{B}(n)} - p_n + p_i - d_i) - \max(0, C_{\mathcal{B}(i)} - d_i) \\ &= C_{\mathcal{B}(i)} - d_n - (C_{\mathcal{B}(n)} - d_n) + \max(0, C_{\mathcal{B}(n)} - p_n + p_i - d_i) - (C_{\mathcal{B}(i)} - d_i) \\ &= \max(d_i - C_{\mathcal{B}(n)}, p_i - p_n) \\ &\leq 0.\end{aligned}$$

The total tardiness of the resulting schedule is not worse than before and the batch containing J_n is completed later than before. Again, this contradict our initial hypothesis. □

2. A dynamic programming algorithm

Consider the schedule of Theorem 1.1. We claim that there is an index k such that (1) jobs J_i executed in batches processed before $\mathcal{B}(n)$ are such that $d_i \leq d_k$ and (2) jobs executed in batches processed after $\mathcal{B}(n)$ are such that $d_i > d_k$ (*cf.*, Fig. 1). Notice that in the batch $\mathcal{B}(n)$, there are jobs with smaller and larger due dates than k. Job J_k can be chosen as the job that has the largest due date among jobs in batches processed before $\mathcal{B}(n)$ on the schedule of Theorem 1.1 (if there are no such jobs, choose J_k as the job with smallest due date).

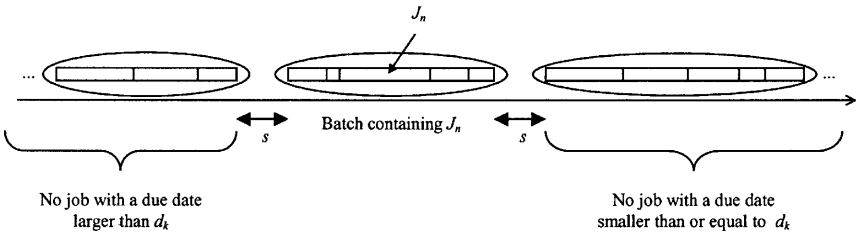

FIGURE 1. Decomposition scheme.

The basic idea of our algorithm is to decompose the scheduling problem into a left sub-problem with $\{J_i : (i \leq n-1) \wedge (d_i \leq d_k)\}$ and a right sub-problem with $\{J_i : (i \leq n-1) \wedge (d_i > d_k)\}$. There are of course some side constraints:

- the completion time of the last batch of the left sub-problem equals the starting time of the first batch of the right sub-problem;
- the total processing time of the jobs of the last batch of the left sub-problem plus the total processing time of the jobs of the first batch of the right sub-problem plus p_n equals the length of the batch $\mathcal{B}(n)$.

This decomposition is applied in turn to both sub-problems. To formalize the dynamic search, we introduce the variables $T(m, d^-, d^+, C^l, P^l, C^r, P^r)$ that equals the minimum tardiness of a schedule of $\{J_i : (i \leq m) \wedge (d^- \leq d_i \leq d^+)\}$ such that

- the completion time of the the first (*i.e.*, the **l**eft) batch is C^l and at most P^l time units are available in this batch to schedule the jobs;
- the completion time of the the last (*i.e.*, the **r**ight) batch is C^r and P^r time units of this batch cannot be used (they are booked for some other jobs).

If no such schedule exists, $T(m, d^-, d^+, C^l, P^l, C^r, P^r)$ is set to ∞. Constraints induced by C^l, P^l, C^r, P^r are recalled in Figure 2.

It comes directly from the definition of $T(m, d^-, d^+, C^l, P^l, C^r, P^r)$ that the optimum is met for

$$\begin{cases} m & \leftarrow & n \\ d^- & \leftarrow & \min_i d_i \\ d^+ & \leftarrow & \max_i d_i \\ C^l & \leftarrow & -s \\ P^l & \leftarrow & 0 \\ C^r & \leftarrow & makespan \\ P^r & \leftarrow & 0 \end{cases} \quad (1)$$

where *makespan* is an integer value that is lower than or equal to the sum of the processing times plus n times the setup time s (in the the worst case, batches are made of a single job and thus n setups occur). C^l is set to $-s$ to let the first non-empty batch of the schedule start at time 0: the left batch starts at time $-s$ and is empty but due to our definition, at least s time units have to ellapse before starting another batch.

We can assume that $C^l + s \leq C^r - P^r$ and that $P^l \geq 0$ otherwise we have $T(m, d^-, d^+, C^l, P^l, C^r, P^r) = \infty$. Moreover, if $\{J_i : (i \leq m) \wedge (d^- \leq d_i \leq d^+)\}$ is empty, then $T(m, d^-, d^+, C^l, P^l, C^r, P^r) = 0$. This condition will be used to stop the recursion. In the following, we assume it does not hold.

We claim that $T(m, d^-, d^+, C^l, P^l, C^r, P^r)$ equals the minimum of L, R, I where

$$\begin{aligned} L &= \max(0, C^l - d_m) + T(m-1, d^-, d^+, C^l, P^l - p_m, C^r, P^r) \\ R &= \max(0, C^r - d_m) + T(m-1, d^-, d^+, C^l, P^l, C^r, P^r + p_m) \end{aligned}$$

and where I is the minimum of

$$\begin{gathered} T(m-1, d^-, d_k, C^l, P^l, \kappa, \kappa - \sigma - \pi) \\ +T(m-1, d_k, d^+, \kappa, \kappa - \sigma - \pi - p_m, C^r, P^r) + \max(0, \kappa - d_m) \end{gathered}$$

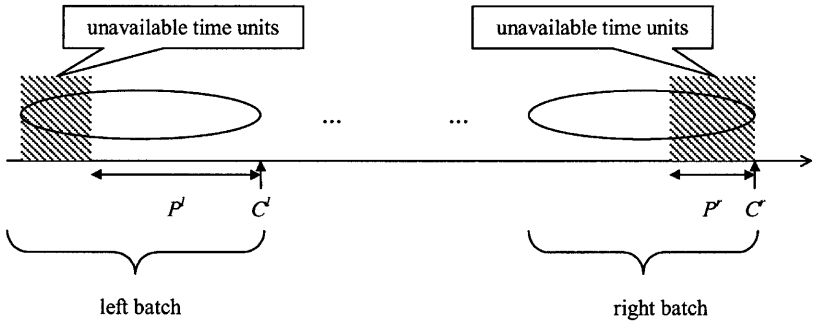

FIGURE 2. Variables definition.

under the constraints

$$\begin{cases} C^l + s \leq \sigma \\ \sigma < \kappa \\ \kappa \leq C^r - P^r - s \\ 0 \leq \pi \leq \kappa - \sigma - p_m \\ d_k \in \{d_i : (i \leq m) \wedge (d^- \leq d_i \leq d^+)\}. \end{cases}$$

Let us give a rationale for our claim.

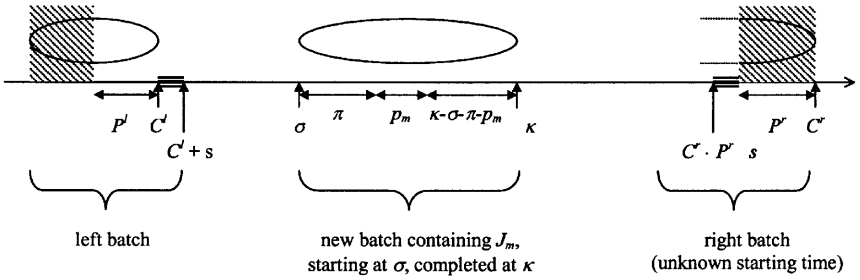

FIGURE 3. An in-between batch.

- The first case ($\min(L, R, I) = L$) corresponds to situation where J_m is scheduled in the left batch. The cost of scheduling J_m is exactly $L = \max(0, C^l - d_m)$. The remaining jobs can be scheduled everywhere between the left and the right batch but the number of available time units in the left batch has to be decreased of p_m. Hence we can follow the schedule that realizes $T(m-1, d^-, d^+, C^l, P^l - p_m, C^r, P^r)$.
- The second case ($\min(L, R, I) = R$) corresponds to situation where J_m is scheduled in the right batch. The cost of scheduling J_m there is exactly $R = \max(0, C^r - d_m)$. The remaining jobs can be scheduled everywhere between the left and the right batch but the number of unavailable time units in the right batch has to be increased of p_m. Hence we can follow the schedule that realizes $T(m-1, d^-, d^+, C^l, P^l, C^r, P^r + p_m)$.
- The third case ($\min(L, R, I) = I$) is more complex. It corresponds to the situtation where J_m is scheduled in an in-between batch (*cf.*, Fig. 3). This batch can start at any time σ after the completion of the left batch plus the set-up time, hence, $C^l + s \leq \sigma$. It is completed at time $\kappa > \sigma$ and at least s time units must ellapse between κ and $C^r - P^r$. We can apply the dominance rule of Theorem 1.1: There is an index k such that (1) jobs J_i executed in batches processed before the in-between batch are such that $d_i \leq d_k$ and (2) jobs executed in batches processed after the in-between batch are such that $d_i > d_k$. Let π denote the total processing time of the jobs, except J_m, in the in-between batch that have a due-date lower than or equal to d_k ($0 \leq \pi \leq \kappa - \sigma - p_m$).

First, the cost of scheduling J_m there is exactly $\max(0, \kappa - d_m)$. Second, jobs with a due-date smaller than or equal to d_k and distinct from m, *i.e.*$\{J_i : (i \leq m-1) \wedge (d^- \leq d_i \leq d_k)\}$, can be scheduled between left batch and the in-between batch. In this batch, no more than π units are available for the above jobs. Hence we can follow the schedule that realizes $T(m-1, d^-, d_k, C^l, P^l, \kappa, \kappa - \sigma - \pi)$. Third, jobs with a due-date larger than d_k and distinct from m, *i.e.* $\{J_i : (i \leq m-1) \wedge (d_k \leq d_i \leq d^+)\}$, can be scheduled between the in-between batch and the right batch. In this batch, no more than $\kappa - \sigma - \pi - p_m$ units are available for the above jobs. Hence we can follow the schedule that realizes $T(m-1, d_k, d^+, \kappa, \kappa - \sigma - \pi - p_m, C^r, P^r)$.

Algorithm 1 Computation of the values $T(m, d^-, d^+, C^l, P^l, C^r, P^r)$

$\mathcal{H} \leftarrow \sum_i p_i + ns$
for $m \leftarrow 1$ to n **do**
  for $d^+ \in \{d_1, \cdots, d_m\}$ taken in increasing order **do**
    for $d^- \in \{d_1, \cdots, d_m\}$ and $d^- \leq d^+$ taken in decreasing order **do**
      for $C^l \leftarrow \mathcal{H}$ down to $C^l \leftarrow 0$ **do**
        for $P^l \leftarrow 0$ to $\mathcal{H}$ **do**
          for $C^r \leftarrow 0$ to $\mathcal{H}$ **do**
            for $P^r \leftarrow \mathcal{H}$ down to 0 **do**
              if $C^l + s > C^r - P^r$ or $P^l < 0$ **then**
                $T(m, d^-, d^+, C^l, P^l, C^r, P^r) \leftarrow \infty$
              else
                $L \leftarrow \max(0, C^l - d_m) + T(m-1, d^-, d^+, C^l, P^l - p_m, C^r, P^r)$
                $R \leftarrow \max(0, C^r - d_m) + T(m-1, d^-, d^+, C^l, P^l, C^r, P^r + p_m)$
                $I \leftarrow \infty$
                for σ, κ such that $C^l + s \leq \sigma < \kappa \leq C^r - P^r - s$ **do**
                  for $\pi \leftarrow 0$ to $\pi \leftarrow \kappa - \sigma - p_m$ **do**
                    for $d_k \in \{d_i : (i \leq m) \wedge (d^- \leq d_i \leq d^+)\}$ **do**
                      $I \leftarrow \min(I, \quad \max(0, \kappa - d_m) +$
                      $T(m-1, d^-, d_k, C^l, P^l, \kappa, \kappa - \sigma - \pi) +$
                      $T(m-1, d_k, d^+, \kappa, \kappa - \sigma - \pi - p_m, C^r, P^r))$
                    end for
                  end for
                end for
                $T(m, d^-, d^+, C^l, P^l, C^r, P^r) \leftarrow \min(L, R, I)$
              end if
            end for
          end for
        end for
      end for
    end for
  end for
end for

We have a straight dynamic programming algorithm to reach the optimum. $T(m, d^-, d^+, C^l, P^l, C^r, P^r)$ is computed and stored in a multi-dimensional array for all relevant values of the parameters:

- m belongs to $\{1, \cdots, n\}$;
- d^-, d^+ are due dates and then belong to $\{d_1, \cdots, d_n\}$;
- C^l, C^r, P^l, P^r belong to $[0, \sum_i p_i + ns]$.

Hence there are $O(n * n^2 * (\sum_i p_i + ns)^4) = O(n^7 \max(\max_i p_i, s)^4)$ relevant combination of the parameters. Actually, one additional value "$-s$" should be taken into account for C^l. It is used for the computation of the the optimal solution, *cf.*, equation (1). To simplify the pseudo-code, this special value has been omitted in Algorithm 1). The algorithm performs several loops on the parameters and for each relevant combination the values L, R, I are computed. This can be done in constant time for L and R. To compute I a minimum over $O((\sum_i p_i + ns)^3 * n)$ terms ($O(\sum_i p_i + ns)$ for σ, κ, π and $O(n)$ for d_k) has to be computed. Each time, this can be done in $O(1)$. This leads to an overall time complexity of $O(n^{11} \max(\max_i p_i, s)^7)$.

3. Conclusion

In this paper, we have shown that the problem of minimizing the total Tardiness on a serial batching machine can be solved in pseudo-polynomial time. Therefore, we have solved one of the last open batching problem. Nevertheless, this algorithm has a prohibitive complexity in $O(n^{11} \max(\max_i p_i, s)^7)$, which surely might be improved.

The authors would like to thank Jacques Carlier for his very intuitive comments. Special thanks to Sigrid Knust for many discussions on batching and scheduling.

References

[1] S. Albers and P. Brucker, The Complexity of One-Machine Batching Problems. *Discrete Appl. Math.* **47** (1993) 87-107.
[2] Ph. Baptiste, *Batching Identical Jobs*, Technical Report, University of Technology of Compiègne (1999).
[3] P. Brucker, *Scheduling Algorithms*. Springer Lehrbuch (1995).
[4] P. Brucker, A. Gladky, H. Hoogeveen, M. Kovalyov, C. Potts, T. Tautenhahn and S. van de Velde, Scheduling a Batching Machine. *J. Sched.* **1** (1998) 31-54.
[5] P. Brucker and S. Knust, *Complexity Results of Scheduling Problems.* `URL: www//mathematik.uni-osnabrueck.de/research/OR/class`
[6] P. Brucker and M.Y. Kovalyov, Single machine batch scheduling to minimize the weighted number of late jobs. *Math. Methods Oper. Res.* **43** (1996) 1-8.
[7] E.G. Coffman, M. Yannakakis, M.J. Magazine and C. Santos, Batch sizing and sequencing on a single machine. *Ann. Oper. Res.* **26** (1990) 135-147.
[8] J. Du and J.Y.-T. Leung, Minimizing Total Tardiness on One Machine is NP-Hard. *Math. Oper. Res.* **15** (1990) 483-495.
[9] L. Dupont, Ordonnancements sur machines à traitement par batch (fournée). *TSI* **10** (to appear).

[10] E.L. Lawler, A "Pseudopolynomial" Algorithm for Sequencing Jobs to Minimize Total Tardiness. *Ann. Discrete Math.* **1** (1977) 331-342.
[11] C.L. Monma and C.N. Potts, On the Complexity of Scheduling with Batch Setup Times. *Oper. Res.* **37** (1989) 798-804.
[12] C.N. Potts and M.Y. Kovalyov. Scheduling with batching: A review. *European J. Oper. Res.* **120** (2000) 228-249.
[13] S. Webster and K.R. Baker, Scheduling Groups of Jobs on a Single Machine. *Oper. Res.* **43** (1995) 692-703.