6^{th} *Int. Conference on Industrial Engineering and Production Management (IEPM'03), Porto (Portugal), may 26-28 2003.*

On-line scheduling on a single batching machine to minimize the makespan

Pascal Richard, Frédéric Ridouard
Laboratoire d'Informatique Scientifique et Industrielle
ENSMA, BP 40109 Téléport 2
F-86961 Futuroscope
{*pascal.richard,frederic.ridouard*}*@ensma.fr*

Patrick Martineau
Laboratoire d'Informatique - Université de Tours
Ecole Polytechnique Universitaire
Département Informatique
64 av. Jean Portalis
F-37200 Tours
pmartineau@univ-tours.fr

Abstract. We present an on-line algorithm to minimize the makespan on a single batch processing machine. We consider a parallel batching machine that can process up to b jobs simultaneously. Jobs in the same batch complete at the same time. Such a model of a batch processing machine has been motivated by burn-in ovens in final testing stage of semiconductor manufacturing. We deal with the on-line scheduling problem when jobs arrive over time. We consider a set of independent jobs. Their number is not known in advance. Each job is available at its release date and its processing requirement is not known in advance. Deterministic algorithms that do not insert idle-times in the schedule cannot be better than 2-competitive and a simple rule based on LPT achieved this bound (Liu and Yu (2000)). If we are allowed to postpone start of jobs, then we prove that a general lower bound is equal to 1.618 for bounded and unbounded batch sizes. We then present deterministic algorithm that is a best possible for the problem with unbounded batch size and equal processing times, as well as for the problem with unbounded batch size, two distinct release dates and non equal procesing times (i.e. there cannot exist an on-line algorithm with a better performance guarantee).

Key words: On-line Scheduling, Batch Processing Machine, Competitive Analysis.

1 Problem Statement

A *batching machine* (or batch proccessing machine) is a machine that can process up to b jobs simultaneously. The jobs that are processed together form a batch. Two kinds of batching machines can be defined: in a *serial batching machine* the processing time of a batch is equal to the sum of processing times of jobs belonging to it; in a *parallel batching machine*, the processing time of a batch is the maximum of the processing times of jobs belonging to it. A recent survey on scheduling problems envolving with batching is presented by Potts and Kovalyov (2000). Next, we only focus on a parallel batching machine. Such a model of a batch processing machine is

motivated by burn-in ovens in final testing stage of semiconductor manufacturing (Uzsoy et al. (1992); Lee et al. (1992)).

A job J_j ($j \in \{1, \dots n\}$) has a processing time p_j and a release date r_j. A job cannot start before its release date and all jobs in the same batch complete simultaneaously. The completion time of J_j is noted C_j. Preemption is not allowed. A scheduling algorithm is said to be *conservative* if it is not allowed to postpone the start of an available job.

Single batching machine problems extend classical single machine problems (by setting $b = 1$). Minimizing the makespan for serial batching machines is trivial since it is sufficient to schedule jobs as early as possible (ties are broken arbitrarily). Next, we focus on minimizing the makespan for parallel batching machines. This problem is noted $1|p - batch|C_{max}$ Brucker (2001) with the classical three field notation.

In the off-line setting, if the jobs are simultaneously released, the optimal sequence is defined by taking the b longest jobs as one batch, then take the b longest remaining jobs as another batch, and so on. This algorithm runs in $\min\{O(n \log n), O(n^2/b)\}$ (Brucker et al. (1998)). When jobs are subjected to non-equal release dates, then the problem $1|p - batch, r_j|C_{max}$ is strongly $\mathcal{NP}$-hard (Brucker et al. (1998)). It remains $\mathcal{NP}$-hard, but in the ordinary sense, if there are only two different release dates $\{0, r\}$ (Liu and Yu (2000)). A pseudopolynomial time dynamic programming algorithm is known if there is a finite set of different release dates (Liu and Yu (2000)). The problem is polynomially solvable if jobs have equal processing times (Ikura and Gimple (1986)) or there is no bound on the batch size Lee and Uzsoy (1999). For the general problem, a Branch and Bound algorithm and a dynamic programming heuristic have been proposed in Sung and Choung (2000). Lee and Uzsoy (1999) proposed heuristics and analysed them through numerical experimentations.

There exist several paradigms in the on-line scheduling theory (Sgall (1998)). Next, we shall consider that jobs arrive over time. The set of jobs is not a priori known. The characteristics of a job is known when it arrives in the system. There is no constraint on release dates: they can be spread out evenly over time, bunched in several burts or come in a single burst.

We shall use competitive analysis to study on-line algorithms (Borodin and R.El-Yaniv (1998)). This approach compares on-line algorithms to an optimal *clairvoyant* algorithm. A good adversary defines instances of problems so that the on-line algorithm achieves its worst-case performance. To analyse deterministic algorithms, two equivalent adversaries can be used: the off-line or *oblivious adversary* defines a sequence of jobs in advance and serves it optimally and the *on-line adaptative adversary* that defines the next incoming jobs according to the decisions taken by the on-line algorithm. An algorithm that minimizes a measure of performance is c-competitive if the value obtained by the on-line algorithm is less than or equal to c times the optimal value obtained by a clairvoyant algorithm (up to a constant if there is an infinite set of jobs). We also say that c is the performance guarantee of the on-line algorithm. An algorithm is said *competitive* if there exists a constant c so that it is c-competitive. More formally, given an instance I, let $A(I)$ be the value obtained by the on-line algorithm A and $OPT(I)$ be the value obtained by the optimal clairvoyant algorithm, then A is c-competitive if there exists a constant c so that:

$$\frac{A(I)}{OPT(I)} \leq c \tag{1}$$

The competitive ratio c_A of the algorithm A is the worst-case ratio while considering any

instance I :

$$c_A = \sup_{anyI} \frac{A(I)}{OPT(I)} \tag{2}$$

The competitive ratio of an algorithm A is greater than or equal to 1. If $c_A = 1$, then A is an optimal algorithm.

Liu and Yu (2000) presented a greedy algorithm (i.e. an algorithm that always takes the best immediate, or local, solution while finding an answer) to solve the single batching machine scheduling problem. This rule, called H, is stated as follows: *Any time when the machine is idle and some unscheduled jobs are available, schedule the longest available unscheduled jobs as many as possible as a batch.* These authors have proved that the previous rule is 2-competitive, and this bound is tight for conservative schedule.

Next, we show that a better on-line deterministic algorithm than H can be defined by allowing inserted idle-times in the schedule. We first show that any deterministic algorithm is at least 1.618-competitive, and then we propose a deterministic algorithm with a tight performance guarantee for the problem $1|p - batch, r_i, p_i = p|C_{max}$ and also for the problem with unbounded batch size and two distinct release dates (i.e. jobs come in two bursts).

2 A general lower bound

The best deterministic algorithm that do not insert idle time in the schedule has a performance guarantee of 2 (Liu and Yu (2000)). Zhang et al. (2001) shown that inserting idle time may help to improve performance guarantees of on-line algorithms. They defined a lower bound of the competitive ratio equal to 1.618. Their proof is based on an oblivious adversary. We next propose another proof based on an on-line adaptative adversary. This result holds for bounded and unbounded batch size.

Theorem 1 *Any deterministic algorithm for scheduling a single batching machine to minimize the makespan has a competitive ratio of at least* $\frac{1+\sqrt{5}}{2} \approx 1.618$.

Proof: We use an on-line adaptative adversary that generates $b-1$ jobs of length p at time 0. The on-line algorithm, say A, can decide to schedule these jobs at time S. According to S, the on-line adversary uses two different strategies:

- no job arrives anymore. In this case, the optimal schedule has a makespan of length p, whereas the algorithm A obtains $S + p$.
- the adversary generates one more job at time $S + \epsilon$ of length p. The adversary schedules all jobs in the same batch, leading to the makespan of $S + p + \epsilon$, whereas the on-line algorithm uses two different batches, leading to a makespan of $S + 2p$.

By regrouping these two cases, we obtain a competitive ratio c_A that verifies:

$$c_A \geq \max\left(\frac{S+p}{p}; \frac{S+2p}{S+p+\epsilon}\right) \tag{3}$$

The right part of the previous inequality is minimized by setting $\epsilon = 0$ and $\frac{S+p}{p} = \frac{S+2p}{S+p}$. This leads to:

$$S(S+p) = p^2 \tag{4}$$

As a consequence, $S = p\frac{-1+\sqrt{5}}{2}$. If we report this value in equation (3), then we obtain:

$$\frac{S+p}{p} = \frac{p\frac{-1+\sqrt{5}}{2}+p}{p} = \frac{1+\sqrt{5}}{2} \tag{5}$$

So, a lower bound of the competitive ratio is $\frac{1+\sqrt{5}}{2} \approx 1.618$. □

3 Upper Bound for the problem $1|p-batch, r_i, p_i = p, b = \infty|C_{max}$

As presented in the introduction, the problem $1|p-batch, r_i, b = \infty|C_{max}$ can be solved in polynomial ($O(n^2)$) time by using dynamic programming (Lee and Uzsoy (1999)). A more efficient implementation of the previously mentioned algorithm has been proposed by Poon and Zhang (2000) leading to a $O(n \log n)$ algorithm. Thus, when processing times are equal, minimizing makespan can be solved in polynomial time in the off-line setting (Ikura and Gimple (1986)).

When the machine is idle and a new job arrives, it should be benefit to wait in order to see if new jobs should arrive in the near future. If a job of length p arrives in the system, we cannot wait for extra information more that p units of time because otherwise if no new job arrives in the system, then the on-line algorithm has a competitive ratio greater than 2. It is also easy to prove that waiting exactly p units of time when a job of length p arrives at an idle-time leads to a competitive ratio of 2.

We now investigate a generalization of these approaches by leaving the machine idle for an interval of time that is proportional to the processing time of the incoming job. When a new job of length p arrives and the machine is idle, then the machine is left idle αp units of time, $0 \leq \alpha \leq 1$. We call this new rule αH. This rule extends the rule H by setting $\alpha = 0$.

Definition 1 *αH: Any time when the machine is idle and some unscheduled jobs are available, let J_j be an available job, then the next batch is not scheduled before $r_j + \alpha p_j$ units of time and then schedule available unscheduled jobs as many as possible as a batch.*

We now prove that αH is a best possible deterministic algorithm when α is set to $(-1+\sqrt{5})/2$ for the problem $1|p-batch, r_i, p_i = p, b = \infty|C_{max}$. Thus, we assume that jobs have equal processing times. Such an assumptation usually simplifies the problem in the off-line setting from the computational complexity point of view (see for instance Baptiste (2000)). Hereafter, we assume that jobs are indexed in non-decreasing order of their release dates:

$$i < j \Rightarrow r_i \leq r_j. \tag{6}$$

We present three simple results in order to define the worst-case competitive ratio of αH for the problem $1|p-batch, r_i, p_i = p, b = \infty|C_{max}$.

Lemma 1 *For any instance I with n jobs of the problem $1|p-batch, r_i, p_i = p, b = \infty|C_{max}$, the optimal makespan is $OPT(I) = r_n + p$.*

Proof: Since $p_i = p, 1 \leq i \leq n$, every job J_i so that $r_i < r_j$ can be schedule in the same batch as J_j without increasing the makespan. Thus, the optimal makespan is achieved by batching all jobs as a single batch that is scheduled at time r_n and completes at time $r_n + p$. □

Lemma 2 *Let I be an instance with n jobs of the problem $1|p-batch, r_i, p_i = p, b = \infty|C_{max}$, then J_n ends any schedule built-up by αH.*

Proof: In a schedule defined by αH, jobs are batched in non-decreasing order of their release dates since, when a batch is begun, then all already released jobs are scheduled. Thus, J_n is scheduled in the last batch of the schedule. □

Lemma 3 *For any instance I of the problem $1|r_i, p_i = p, b = \infty|C_{max}$, αH completes no job after $2p$ units of time after its release: $F_i = C_i - r_i \leq 2p$.*

Proof: Let us consider an arbitrary job J_k and let σ be the schedule built-up by αH. A block is defined as a partial schedule without any idle-time. If J_k starts a block without idle-time in σ, then it cannot start after $r_k + \alpha p$ and thus completes by time $r_k + (1 + \alpha)p$. As a consequence, $C_k - r_k \leq 2p$, since $0 \leq \alpha \leq 1$. If J_k is scheduled in a block but not in the first position, then the worst-case flow time F_k is achieved when J_k is released just after the beginning of a batch. This batch completes at time $r_k + p - \epsilon$, then J_k is scheduled in the next batch and thus completes at time $r_k + 2p - \epsilon$, where ϵ is an arbitrary small number. As a consequence, in both cases: $F_k = C_k - r_k \leq 2p$. □

Theorem 2 *αH is 1.618-competitive for the problem $1|p-batch, r_i, p_i = p, b = \infty|C_{max}$ when $\alpha = (-1 + \sqrt{5})/2$.*

Proof: Let us consider an arbitrary instance I of the problem. According to lemma 1, $OPT(I) = r_n + p$. And according to lemma 2, J_n is always scheduled in the last batch. Let $A(I)$ be the makespan achieved by αH. We consider two cases:

- If the batch B in which J_n is scheduled starts after an idle-time, then the latest completion time of B is:

  $$max_{j \in B}(r_j + (\alpha + 1)p) = r_n + (\alpha + 1)p$$

  Thus, $A(I) \leq r_n + (\alpha + 1)p$, for any instance I. Thus, the worst-case competitive ratio verifies:

  $$\frac{A(I)}{OPT(I)} \leq \frac{r_n + (\alpha + 1)p}{r_n + p} \leq (\alpha + 1) \tag{7}$$

- If B is not the only batch in the last block without idle-time, then according to lemma 3, for every job J_k is verified: $C_k - r_k \leq 2p$. Due to lemma 2, J_n ends the schedule. Thus $A(I) \leq r_n + 2p$. Furthermore, since B is not the first batch in the block that ends the schedule, then we verifies that $r_n \geq \alpha p$. Otherwise all jobs are scheduled in a single batch according to the αH rule. So, the worst-case competitive ratio in this case is:

$$\frac{A(I)}{OPT(I)} \leq \frac{r_n + 2p}{r_n + p} \leq \frac{\alpha p + 2p}{\alpha p + p} = \frac{\alpha + 2}{\alpha + 1} \tag{8}$$

If we regroup both cases, then we have:

$$\frac{A(I)}{OPT(I)} \leq \max\left(\alpha + 1; \frac{\alpha + 2}{\alpha + 1}\right) \tag{9}$$

Simple algebra analysis shows that the worst-case value of the above expression is achieved when:

$$\alpha + 1 = \frac{\alpha + 2}{\alpha + 1} \tag{10}$$

$$\alpha^2 + \alpha - 1 = 0 \tag{11}$$

So, the positive solution of this equation is $\alpha = (-1 + \sqrt{5})/2$, leading to the competitive ratio:

$$\frac{A(I)}{OPT(I)} \leq \alpha + 1 = \frac{1 + \sqrt{5}}{2} \approx 1.618 \tag{12}$$

□

Zhang et al. (2001) studied the problem with unbounded batch size (even for problems having non equal processing times). But, they proposed an on-line algorithm that inserts huge interval of idle times in the schedule. Let U_i be the set of available jobs at some time t. Let p_k be the longest job in U_i. Then the next batch is not started before time: $(1 + \alpha)r_k + \alpha p_k$. As a consequence, even for instances of problems having a huge number of jobs, then no more than two batches can be scheduled within a continuous block of batches. More precisely, a batch can be delayed by at most one batch in every schedule. From the practical point of view, inserted idle-times depend simultaneously on release dates and processing times. In our algorithm, inserted idle-times only depend on processing times of available jobs. Such a property is quite useful if the on-line algorithm must deal with large scheduling horizons. Furthermore, our algorithm always introduce less idle times in the schedule than the algorithm presented by Zhang et al. (2001). But, we do not know if our algorithm still $(1 + \alpha)$-competitive for the general problem with non-equal processing times. In the next section, we prove that our rule is still a best possible algorithm for scheduling problems having two distinct release dates.

4 Problems with two release dates and unbounded batch size

We now consider a special case where every instance has two distinct release dates but we relax the assumption of equal processing times studied in the previous section. Considering scheduling

problems with two distinct release dates allow to model jobs arriving in two bursts. In this specific case, we show that αH is still a best possible deterministic algorithm. The offline version of this problem is obviously solved in polynomial time when batch sizes are not bounded (but if such a bound exists, then the offline problem is solved in pseudo-polynomial time (Ikura and Gimple (1986))). To study this problem, we only have to consider two jobs: the longest of the first burst and the longest of the second burst.

Theorem 3 *αH is at best* 1.618*-competive for minimizing the makespan on a single batching machine with two release dates. This is achieved for* $\alpha = \frac{-1+\sqrt{5}}{2}$.

Proof: Let p_1 be the longest job belonging to the first burst and p_2 this one related to the second burst. The worst-case scenario that must deal with αH is defined by the following strategy: generate a job of length p_1 at time 0 and another job of length p_2 at time $\alpha p_1 + \epsilon$, where ϵ is an arbitrary small number. The on-line algorithm αH uses 2 batches: the first one for the job of length p_1 and the other one for the jobs of length p_2. The makespan of the schedule obtained by αH, say $A(I)$, is:

$$A(I) = \alpha p_1 + p_1 + p_2 \tag{13}$$

The optimal off-line algorithm has two possibilities according to the values α, p_1, p_2:

- it schedules jobs in a single batch starting at time $\alpha p_1 + \epsilon$. The makespan is $\alpha p_1 + \max(p_1, p_2)$.
- it schedules jobs in two batches. The first one is used to schedule the job of length p_1, and the second one for the remaining jobs of length p_2. In that case, the makespan is $p_1 + p_2$.

The former case is useful if $p_1 \ll p_2$, whereas the latter is useful if $p_1 \gg p_2$. If we regroup these two cases, then the optimal makespan is defined by:

$$OPT(I) = \min\left(p_1 + p_2; \alpha p_1 + \max(p_1; p_2)\right) \tag{14}$$

The competitive ratio with parameters α, p_1, p_2 is:

$$c_A = \frac{A(I)}{OPT(I)} = \frac{p_1(\alpha + 1) + p_2}{\min\left(p_1 + p_2; \alpha p_1 + \max(p_1; p_2)\right)} \tag{15}$$

Simple arithmetic calculations show that the previous expression is minimized while considering any instance I for $\alpha = \frac{-1+\sqrt{5}}{2}$. According to this value of α, the competitive ratio is maximized for $p = p_1 = p_2$. So, we obtain:

$$c_A \leq \frac{p(\frac{-1+\sqrt{5}}{2} + 2)}{p(\frac{-1+\sqrt{5}}{2} + 1)} = \frac{3 + \sqrt{5}}{1 + \sqrt{5}} = \frac{1 + \sqrt{5}}{2} \approx 1.618 \tag{16}$$

□

Figure 1 represents the evolution of the competitive ratio that is achieved by αH in function of $\alpha \in [0, 1]$.

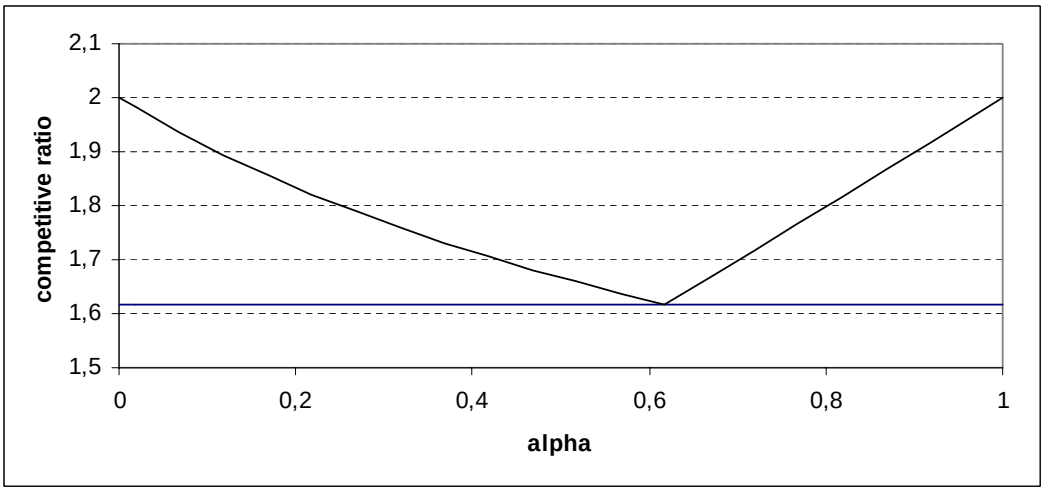

Figure 1: Competitive ratio of the αH rule for the problem with two release dates.

5 Conclusion

We have studied the on-line scheduling problem of a single batching machine to minimize the makespan. If jobs arrive over time and inserted idle-times are not allowed in the schedule then no deterministic algorithm can be better than 2-competitive. If inserted idle-times are allowed, then it might be beneficial to leave the machine idle since we have presented a general lower bound of $(1+\sqrt{5})/2$ both for bounded and unbounded batch sizes. We have proposed a simple deterministic rule that yields a tight performance guarantee for the problem $1|p-batch, r_i, p_i = p, b = \infty|C_{max}$ and for the problem with two distinct release dates (i.e. jobs arrive in two bursts). We think that the worst-case behaviour of on-line algorithms are not related to the batch size, but is only due to the interaction between batch processing times and job arrival times.

Perspectives of this work are to study the on-line single batching machine scheduling problem:

- when unbounded batch sizes and arbitrary processing times are considered. We have only obtain a best possible deterministic algorithm when jobs have equal processing times or if there exist only two distinct release dates. We do not know if the algorithm αH, presented in this paper, leads to a best possible deterministic algorithm in the general problem with unbounded batch size.

- when bounded batch sizes and arbitrary processing times are considered. The best known deterministic on-line algorithm is 2-competitive. We do not know if a better algorithm can be designed. This problem remains an interesting open issue.

- to minimize other performance measures such as maximum lateness, mean completion time and number of tardy jobs.

To the best of our knowledge, no randomized on-line algorithm (i.e. an algorithm that performs a random choice whenever a decision has to be taken) has been proposed for scheduling a single

batching machine. For numerous problems, randomized algorithms allow to improve performance guarantees in the on-line setting. It might be the case for the scheduling problem considered in this paper.

References

Baptiste, P., 2000. Batching identical jobs. Mathematics methods of Operations Research 53, 355–367.

Borodin, A., R.El-Yaniv, 1998. Online Computation and Competitive analysis. Cambridge University Press.

Brucker, P., 2001. Scheduling Algorithms. Springer Verlag.

Brucker, P., Gladky, A., Hoogoveen, H., Kovalyov, M., Potts, C., Tautenhahn, T., VanDeVelde, S., 1998. Scheduling a batching machine. Journal of Scheduling 1 (1), 31–58.

Ikura, Y., Gimple, M., 1986. Efficient scheduling algorithms for a single batch processing machine. Operations Research Letters 5, 61–65.

Lee, C., Uzsoy, R., 1999. Minimizing makespan on a single batch processing machine with dynamic job arrivals. International Journal of Production Research 37 (1), 219–236.

Lee, C., Uzsoy, R., Martin-Vega, L., 1992. Efficient algorithms for scheduling semiconductor burn-in operations. Operations Research 40, 219–236.

Liu, Z., Yu, W., 2000. Scheduling one batch processor subject to job release dates. Discrete Applied Mathematics 105, 129–136.

Poon, C., Zhang, P., 2000. Minimizing makespan in batch machine scheduling. in: proc Int. Symposium on Algorithms and Computations (ISAAC 2000), Lecture Notes in Computer Science, Springer Verlag 1969, 386–397.

Potts, C., Kovalyov, M., 2000. Scheduling with batching: a review. European Journal of Operational Research 120 (2), 228–249.

Sgall, J., 1998. On-line scheduling - a survey. in: On-line algorithms. The state of the art, Lecture Notes in Computer Science, Springer Verlag 1442, 196–231.

Sung, C., Choung, Y., 2000. Minimizing makespan on a single burn-in oven in semiconductor manufacturing. European Journal of Oerational Research 120, 559–574.

Uzsoy, R., Lee, C., Martin-Vega, L., 1992. A review of production planning and scheduling models in the semiconductor industry, part i:system characteristics, performance evaluation and production planning. IEE Transactions on Scheduling and Logistics 26, 44–55.

Zhang, G., Cai, X., Wong, C., 2001. On-line algorithms for minimizing makespan on batch processing machines. Naval Research Logistics 48, 241–258.