Optimal Batch Schedules for Parallel Machines

Frederic Koehler[1⋆] and Samir Khuller[2]

[1] Princeton Univ., Princeton NJ 08544, USA,
f.koehler427@gmail.com
[2] Dept. of Computer Science, Univ. of Maryland, College Park, MD 20742, USA,
samir@cs.umd.edu

Abstract. We consider the problem of batch scheduling on parallel machines where jobs have release times, deadlines, and identical processing times. The goal is to schedule these jobs in batches of size at most B on m identical machines. Previous work on this problem primarily focused on finding feasible schedules. Motivated by the problem of minimizing energy, we consider problems where the number of batches is significant. Minimizing the number of batches on a single processor previously required an impractical $O(n^8)$ dynamic programming algorithm. We present a $O(n^3)$ algorithm for simultaneously minimizing the number of batches and maximum completion time, and give improved guarantees for variants with infinite size batches, agreeable release times, and batch "budgets". Finally, we give a pseudo-polynomial algorithm for general batch-count-sensitive objective functions and correct errors in previous results.

Keywords: Scheduling, Batching, Optimal Algorithms

1 Introduction

Batch Scheduling refers to the scheduling of jobs when jobs can be processed in batches of size at most B. The notion of parallel batch scheduling of jobs was initially proposed to model deliveries by trucks of bounded capacity [9]. It has, among other applications, been used to model the management of large multimedia-on-demand systems [2] and "burn-in" operations in an oven where a number of chips can be baked together at once [10]. We focus on the version where all of the jobs in a batch are processed together and start at the same time. In addition, for each job (J_α) the schedule must respect release times (r_α) and deadlines (d_α), times at which jobs become available to process and must be processed by, respectively. This can, for example, model the delivery of people flying into an airport for a conference, where each person must be transported by a given deadline using a fleet of limited capacity vehicles.

Many results in deterministic batch scheduling focus on the version where all jobs have release times, deadlines, and uniform length of p [5, 10, 9, 3, 1], where the

⋆ The first author's work was done as part of his high school research project at the Univ. of Maryland, and later an NSF REU supplement to CCF 0937865. The work of the second author is supported by NSF grants CCF 0728839 and CCF 0937865.

objective is to find a feasible schedule of batches each containing at most B jobs. The start and end time of the batch must respect the release time and deadline of each job in the batch. Using standard techniques these feasibility algorithms can be used to minimize objectives such as maximum lateness (L_{max}) and maximum completion time (C_{max}). Note that when jobs have different lengths, deciding feasibility becomes an $\mathcal{NP}$-complete problem, although approximation algorithms exist, e.g. [2].

Motivated by issues of savings energy, recently Chang et al. [4] consider the problem of minimizing the time the machine is being used, referred to as *activation time*. In this case of identical job lengths, this translates to scheduling all of the jobs using the fewest number of batches. Chang et al. [4] develop a $O(n^8)$ algorithm for this problem based on the work of Baptiste [1]; the space complexity is also very high, and it is also designed for the single processor case only. The paper also considers a variety of other cases of the activation problem — e.g. when the release times and deadlines are integral and $p = 1$ they present a linear time algorithm. Even though batch scheduling has been studied for over twenty-five years, these are the first algorithms which explicitly aim to minimize batch count. However, we expect the number of batches used almost always affects the energy cost (and thus profit) of the system.

The basic problem dealt with is that of creating a schedule of batches for m identical machines. A batch (or batch instance) B_α in a schedule is associated with three properties:

- The set of jobs contained in the batch. We let $|B_\alpha|$ denote the number of jobs in a batch. In a feasible schedule, $|B_\alpha| \leq B$, where B is the given batch size constraint.
- The start time $s(B_\alpha)$. $\forall J_j \in B_\alpha$, the completion time $C_j = s(B_\alpha) + p$. In a feasible schedule, $r_j + p \leq C_j \leq d_j$.
- The machine $m(B_\alpha)$ that the batch is scheduled upon, occupying the time interval $[s(B_\alpha), s(B_\alpha) + p)$ on that machine. In a feasible schedule, the time intervals of batches scheduled on the same machine must be disjoint.

Our results, briefly: we can find a batch-count and C_{max} minimal schedule in $O(n^3)$ time and improve that in variants of the problem; we produce a $O(n)$ algorithm for the agreeable problem ($r_a < r_b \rightarrow d_a \leq d_b$). We give a pseudo-polynomial algorithm for a notion of batch-count-sensitive objective functions.

1.1 Related Work

Ikura and Gimple originally gave a $O(n^2)$ algorithm for scheduling agreeable jobs on a single processsor with the objective of minimizing C_{max}. Lee et al. found a $O(nB)$ algorithm for this same problem using dynamic programming [10]. Baptiste[1] finally showed that the problem with arbitrary release times was polynomial-time solvable for a broad class of sum-function objectives, such as $\sum C_j$. However his algorithms have extremely high (polynomial) complexity.

Recently, Condotta et al. [5] developed improved algorithms for the feasibility problem for general release times and deadlines: for the single machine case

they provide an $O(n^2)$ time algorithm. They also study the previously ignored multiple identical machine case and provide an $O(n^3 \log n)$ time algorithm. These algorithms are generalized forms of algorithms for the non-batching problem ($B = 1$): the $O(n^2)$ algorithm is based on the "forbidden regions" method of Garey et al. [7], and the $O(n^3 \log n)$ algorithm for the multiprocessor case is based on the "barriers" method of Simons [14].

The barriers and forbidden regions methods for a single processor are both notable for choosing schedules with the property that each job, numbered from the left (or the right), starts as soon as possible. Formally, the start time of the i^{th} job from the left in the generated schedule is a lower bound on the start time of the i^{th} job in *any* feasible schedule. These schedules are thus optimal for the objectives $\sum C_j$ and C_{max}. We shall say that these schedules have UNIT-OPTIMAL structure. Recent algorithms using graph-theoretic techniques find schedules with identical structure [6, 12].

The Condotta et al. paper claims that each batch, counting from the left, for their barriers algorithm has minimal start time (Lemma 4). This claim is incorrect: consider a problem instance with large deadlines, two machines, $B = 2$ (or any even number), B jobs released at time 0 and B jobs released at time p. The barriers algorithm will produce a schedule with two full batches, the second at time p. However a feasible schedule exists where the first two batches are started at time zero, each containing $B/2$ jobs. This disproves their claim and invalidates their proof of correctness. However by correcting this claim it is still possible to show their algorithm's correctness. (Here is a sketch: Consider only the classes of schedules where each batch, from the left, greedily takes as many jobs as possible. The schedules generated by the barriers algorithm are those which for any k, both process the *minimum* number of jobs in the first k batches numbered from the left and starts each batch B_k no earlier than any nonempty B'_k in any other schedule of this class. The processing of the minimum number of jobs is crucial to proving the batch start times are minimized.)

They also claim that their algorithms immediately minimize $\sum C_j$ and C_{max} in the batching problem. We do not believe this to be the case. Consider the single machine case: by delaying a job slightly, it may be possible to overlap it in a batch with other jobs, drastically reducing its completion time by not blocking on the processing of the first job. The barriers algorithm only creates barriers when it encounters infeasibility, so if it never encounters infeasibility, no attempt is made to delay jobs to batch them together with later released jobs. Similarly, the forbidden regions algorithm will find no forbidden regions.

The following simple example will demonstrate our claims. Run the barriers algorithm on jobs with r_j, d_j pairs $\{(1, 16), (2, 20), (6, 24))\}$, with the processing length for jobs $p = 8$, with batch size $B = 3$, and one machine: a batch will be created at time $r_1 = 1$ and at $r_1 + p = 9$. An optimal schedule for C_{max} uses only a single batch starting at $r_3 = 6$. Interestingly, an optimal schedule for $\sum C_j$ uses one batch starting at $r_2 = 2$ and another at $r_2 + p = 10$.

Theorem 1. *In the batching problem, there exist instances where minimizing* $\sum C_j$ *and* C_{max} *simultaneously is impossible.*

On a different note, when scheduling unit jobs on multiple processors, Simons [14] showed that w.l.o.g. one can only consider the *cyclic* schedules. We will make exactly this assumption in our paper. The original proof of the following claim comes from Simons for the non-batching case [14].

Lemma 1. *For any feasible schedule, a solution identical except in machine assignment exists which is cyclic, i.e. where* $\forall x, (B_x, B_{x+m}, \ldots)$ *are scheduled on the same machine.*

1.2 Our Approach

We generalize the notion of UNIT-OPTIMALITY. We shall call our structure RIGHT-HEAVY BATCH-OPTIMALITY (RHBO). It comprises the following properties (note the *descending* batch numbering scheme):

(1) Consider any feasible schedule S' composed of batches $B'_1 \ldots B'_u$ where B'_u is the *earliest* starting batch (and B'_1 the *latest*) containing a job in schedule S'. $\forall i \leq u, \sum_{b=1}^{i} |B_b| \geq \sum_{b=1}^{i} |B'_b|$; i.e. the number of jobs in $\bigcup_{b=1}^{i} B_i$ is an upper bound for feasible schedules.
(2) For any B_i, the start time of batch B_i is a lower bound for feasible schedules; i.e. for any B'_i in any feasible schedule S', $s(B_i) \leq s(B'_i)$.

In the case that $B = 1$, the first property is trivial and the second property makes the structure identical to UNIT-OPTIMALITY. Note that unlike in the corrected version of the barriers algorithm, our bounds hold for *all* feasible schedules. Any schedule with these properties is optimal for many objectives:

1. C_{max} because the start time of B_1 is a lower bound. In fact the makespan (availability time) $m_x = B_x$ of *all* of the machines is minimized; so e.g. $\sum m_x$ (average makespan) and a variety of other norms are also minimized.
2. K, the number of batches, by the first property.
3. $\sum_{B_x \in S} s(B_x)$, the sum of batch start times, because a minimal number of batches is used and the start time of each batch is a lower bound.

The first section of our paper gives a low polynomial time complexity algorithm witnessing the existence of these structures. We also use this existence result to produce an optimal recursive algorithm for the *agreeable* batch scheduling problem [9]. The property of simultaneous makespan minimization on multiple machines is crucial to the decomposition.

In general Condotta et al.'s algorithms [5] will use batches efficiently only if that part of the schedule is highly constrained or many jobs share a release time. When batch sizes are larger than e.g. $B = 2$, this becomes evident. By using fewer batches, we also can improve our time complexity bound in the case that a feasible schedule exists with K_* batches ($n/B \leq K_* \leq n$) as excess batches increase algorithmic overhead. In the case of agreeable release times, we produce an elegant algorithm which searches for RHBO schedules. It is both more general and lower complexity than previous algorithms for this problem. This completes our study of RHBO schedules.

Finally we design a pseudo-polynomial algorithm for optimizing a broad array of batch-count-sensitive objectives, generalizing [1]. The lack of structure in this general setting leads to very high complexity. *This result, proofs of auxillary lemmas, and pseudocode versions of the algorithms are omitted for space reasons; the full version is at* http://www.cs.umd.edu/~samir/grant/BatchScheduling.pdf

2 Scheduling Jobs on Multiple Batch Machines

For this section, we study the problem of scheduling *all* of the jobs in a given instance. Thus when we refer to a feasible schedule, this schedule must successfully process all n jobs. We will work through a series of tentative (infeasible) schedules in our algorithms. Each tentative schedule will obey a RHBO structure: we refer to the two properties of a RHBO schedule as Invariant (1) and Invariant (2), matching the numbering in the definition. We say a job J_j is *deadline-available* in a batch B_b if $d_j \geq s(B_b) + p$. Using this notion we will define a third invariant which determines job selection within batches:

(3) $\forall B_x \forall J_j \in B_y$ such that $x < y$, if J_j is deadline-available in B_x then B_x is full of jobs with no less strict release times ($|B_x| = B, r_j \leq min_{J_i \in B_x} r_i$).

This invariant can be viewed directly as expressing a relationship between a single batch B_x, and a set of *preceding jobs* in higher-numbered (earlier) batches. It equivalently states that each batch must prefer to pick latest-released jobs from the set of jobs preceding the next-earliest batch. Note that increasing start times can only reduce the set of deadline-available jobs, and thus only make this invariant easier to satisfy.

We assume w.l.o.g. that $\forall J_j$, $r_j + p \leq d_j$: jobs violating this constraint are impossible to process. Initially let $s(B_b) \leftarrow -\infty$ for all B_b (including those earlier than B_n which cannot actually be used), since $-\infty$ is a trivial lower bound on the start time of any batch.

Lemma 2. *Invariants (2) and (3) imply Invariant (1).*

Proof. Assume Invariant (1) is violated while the other two invariants hold. Let B'_x be the latest batch in a feasible schedule S' such that $\sum_{b=1}^{x} |B'_b| > \sum_{b=1}^{x} |B_b|$. Because we chose the latest batch where the invariant is violated, the invariant holds for $B_{x-1} \dots B_1$, and so B'_x must contain at least one additional job J_j which is not in B_x. As B_x cannot be full, Invariant (3) implies that $d_j < s(B_x) + p$. By Invariant (2), $s(B_x) \leq s(B'_x)$ and so $d_j < s(B'_x) + p$. The deadline for job J_j is violated, so schedule S' cannot be feasible.

Lemma 3. *If the* OPTIMALITY *invariants holds for a partial schedule $B_{x-1} \dots B_1$ then $\forall J_l \notin \bigcup_{b=1}^{x-1} B_b$, for any feasible schedule S' composed of $B'_1, B'_2, \dots$ it must be true that $r_l \leq s(B'_x)$.*

Proof. Let $J_l \in B'_y$. If $y \geq x$, then we have that $r_l \leq s(B'_y) \leq s(B'_x)$. Otherwise ($y < x$): because $J_l \notin \bigcup_{b=1}^{x-1} B_b$ and $J_l \in \bigcup_{b=1}^{x-1} B'_b$, by Invariant (1) there exists

some job $J_k \in \bigcup_{b=1}^{x-1} B_b$ such that $J_k \notin \bigcup_{b=1}^{x-1} B'_b$. Also, by Invariant (2), $d_l \geq s(B'_y) + p \geq s(B_y) + p$, so by Invariant (3), $r_l \leq r_k$. Because $J_k \in B'_z$ with $z \geq x$, as above $s(B'_x) \geq r_k \geq r_l$.

2.1 Scheduling with an Unbounded Number of Machines

Theorem 2. *A feasible schedule obeying the* OPTIMALITY *invariants can be computed in $O(n^2)$ time if $m = \infty$.*

Proof. For the first (latest) batch, r_{max} is a lower bound on the start time — thus setting $s(B_1) = r_{max}$ obeys Invariant (2). Invariant (3) determines that this batch should be filled with the maximal number (up to B) of the latest-released deadline-available jobs. For any other batch B_i, we can inductively assume that the partial schedule of $B_{i-1}, \ldots, B_1$ obeys the invariants. Let U be the set of unscheduled jobs. All jobs in U can only be scheduled in B_i and earlier batches. Set $s(B_i)$ to be the latest release time in U; Lemma 3 guarantees that this satisfies Invariant (2). Once again, Invariant (3) dictates that the maximal number of the latest-released deadline-available jobs are chosen to fill the batch.

Every batch created contains at least one job. Thus there are at most n batches and this construction takes $O(n^2)$ time.[3]

An example tentative schedule is shown in Figure 1, based on the input from Table 1 with batches taking three units of time ($p = 3$) to process up to two ($B = 2$) jobs at a time.

Table 1. Jobs for Example 2

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
r_j	0	2	1	2	0	4	3	4	1	5	4	3	8	9	10	7
d_j	5	11	6	12	8	13	8	10	7	8	9	9	16	13	14	12

2.2 Scheduling with a Bounded Number of Machines

Theorem 3. *Given a tentative schedule containing all jobs with no more than B jobs in any batch, and obeying the* OPTIMALITY *invariants, in $O(n^3)$ time it is possible to either show no feasible schedule exists or to find a feasible schedule obeying the* OPTIMALITY *invariants.*

Proof. We show how to use invariant-preserving transformations to make this schedule into a feasible one. We use two cooperative alternating passes: PushForward, which increases start times, and MoveBack, which moves jobs which are provably in the wrong batch backward.[4]

[3] Though $O(n \log n)$ is possible

[4] MoveBack tightens the bounds of Invariant (1) while PushForward tightens Invariant (2).

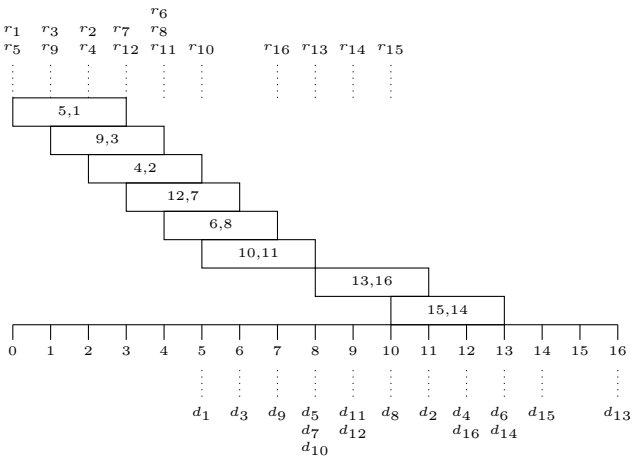

Fig. 1. Schedule constructed by Theorem 2 on Example 2

PushForward is the pass that starts first. It processes batches left-to-right (earliest-to-latest) consecutively; initially, it starts from B_i, the earliest non-empty batch in the schedule. We describe its action. Let B_c be the current batch. Let P be the set of batches which are earlier than B_c (higher indexed): inductively assume that the batches in P (1) are in non-decreasing order of start time (counting higher indexed batches first), (2) can be scheduled without overlap on m machines, and (3) contain jobs whose release times and (4) deadlines are satisfied. To process B_c, let

$$s(B_c) \leftarrow \max\left[\{s(B_{c+m}) + p, s(B_c), s(B_{c+1})\} \cup \{r_l | J_l \in B_c\}\right].$$

As noted before, increasing start times always preserves Invariant (3); we now show Invariant (2) is maintained as well. For any of the first three terms, it is possible that they may reference an empty batch past the end of the schedule: for all such batches their start time has been set to $-\infty$, so the term reduces to $-\infty$ which is a correct lower bound. Otherwise (the normal case), the first term is a valid lower bound because we have restricted ourselves to the class of cyclic schedules, and there can be no overlap between batches run on the same machine. This also satisfies inductive hypothesis (2). For the third term, by definition a lower bound for the start time of B_{c+1} extends to B_c. This satisfies inductive hypothesis (1). The final set of release times are valid lower bounds by Lemma 3, satisfying the inductive hypothesis (3).

After updating the start time, if there are any jobs in B_c which are no longer deadline-available, pick one such J_i arbitrarily and move on to the next phase MoveBack. If there are no such jobs, then the final hypothesis (4) is satisfied. If $B_c = B_1$ then terminate: supposing that all batches in our schedule obey the batch size constraint (which we have not shown yet), then using our inductive

hypotheses the requirements for a feasible schedule are satisfied. Otherwise ($B_c \neq B_1$), continue on to the next batch (B_{c-1}).

We now describe MoveBack. This phase will not adjust start times so Invariant (2) is preserved. We will study Invariant (3) separately for each batch and its set of preceding jobs to show that it holds for all batches (when obvious, we will leave implicit which batch the invariant is preserved with respect to). We now describe the action of this phase. The first action this phase takes is to remove J_i from B_c. If there do not exist preceding deadline-available jobs to B_c, this does not affect Invariant (3) with respect to B_c. If there does exist at least one such job, pick the one with latest release time and move it from its current batch B_a into B_c. We say in this case that job $J_{a'}$ was *brought forward* from B_a. This may violate Invariant (3) with respect to B_a; if so, we will show that invariant is restored before the end of this phase. The removal of a job guarantees that $|B_a| < B$. The rest of this phase moves right-to-left over consecutive batches, starting with B_{c+1}. Call the current batch being processed B_z; also let the current job, initially J_i, be called J_j. We now describe the action performed for B_z; remember that when we say this phase *continues*, that means the next batch examined is the preceding batch B_{z+1}.

Case 1 ($|B_z| = B$)**.** Let $J_{j'} = \text{argmin}_{J_y \in B_z} r_y$.

Case 1.a ($r_{j'} \leq r_j$)**.** Swap J_j into B_z, removing $J_{j'}$. Continue MoveBack with $J_{j'}$.

Case 1.b ($r_{j'} > r_j$)**.** Continue MoveBack with J_j.

In either case a new, possibly deadline-available, job will now precede B_z (either J_j or $J_{j'}$). Even if the job is deadline-available, its release time is no bigger than the smallest in B_z so Invariant (3) is preserved.

Case 2 ($|B_z| < B$)**.** Place J_j into B_z.

Case 2.a (Job $J_{a'}$ was brought forward from B_a) Suppose $B_z \neq B_a$. Since $J_{a'}$ preceded B_z before its move (after the execution of the previous phase) and $|B_z| < B$, by Invariant (3) $J_{a'}$ cannot have been deadline-available in B_z. However, it is deadline-available in B_c, and by the action of PushForward we know that this implies $J_{a'}$ is deadline-available in all earlier (higher-numbered) batches. By contradiction $B_z = B_a$.

Since J_j came from some batch later than B_z but not later than B_c, and $J_{a'}$ was deadline-available in this origin-batch, $r_j \geq r_{a'}$ by Invariant (3). Therefore the replacement of J_a by J_j cannot violate Invariant (3) with respect to B_a.

Case 2.b (No job was brought forward) Adding an additional job to a nonfull batch cannot violate Invariant (3), so it is preserved.

In either case, the transformations of this phase are complete. Only batches between B_a and B_c inclusive have been modified. With respect to batches B_f with $f > a$ of $f < c$, this implies that Invariant (3) has been maintained. Thus we have shown that for every batch, Invariant (3) holds with respect to it at the end of this phase. Recall that Invariant (1) holds now by Lemma 2. If $z > n$, declare the scheduling instance infeasible: by Invariant (1), only at most $n - 1$ jobs can be scheduled in $B_1, \ldots, B_n$. Otherwise, continue on to

PushForward at Bz: because we have not modified any batches earlier than B_z, the required inductive hypotheses hold for them.

This completes the description of the algorithm itself. As noted before, we still must show that the batch size restriction is obeyed to show that the algorithm is *partially correct*: if it terminates, it gives a correct answer. Recall we required our initial schedule to obey the restriction. Only the MoveBack pass modifies the assignment of jobs to batches, but it only adds a net job to a batch which has at most $B - 1$ jobs. Therefore the batch size restriction is always obeyed.

We must now show that our algorithm terminates. We claim that there can be at most $O(n^2)$ passes: for every MoveBack pass, J_j can never be placed in B_c again, because start times only increase and jobs are brought forward only if they are deadline-available; there are n jobs and at most n batches, so this makes $O(n^2)$ possible passes. Both passes run in $O(n)$ time, so a $O(n^3)$ time bound follows.

A complete algorithm is formed by composing the previous two theorems: feasible schedules for finite m are a subclass of those for unbounded m so the precondition for Theorem 3 holds. See Figure 2 and Figure 3 where $m = 2$. However, as Theorem 3 requires little from its initial schedule, far less intelligent schemes would give the same time bounds.

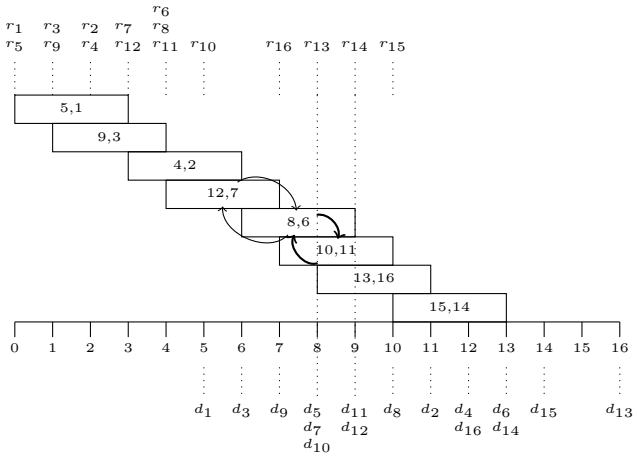

Fig. 2. Operations performed by first (thick arrows) and third (thin) pass of MoveBack.

We have not so far discussed how to efficiently represent the contents of a batch. Let each batch's contents be represented by two data structures: a binary min-heap of the jobs ordered by deadlines, and an AVL tree of the jobs' release times maintaining counts in each node for duplicate release times. Our efficiency proofs are omitted for space reasons; they modify the algorithm's internals very

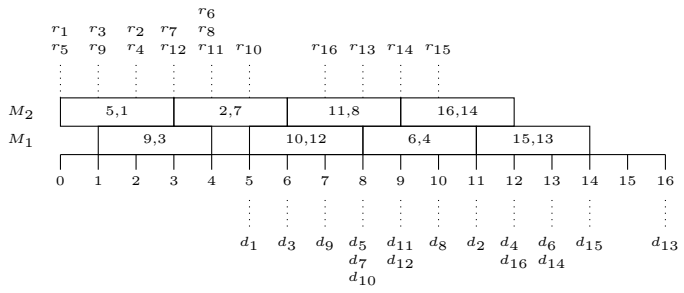

Fig. 3. Final schedule produced for Example 2

slightly to improve its performance. If we are given a fixed batch budget K_* (modifying the algorithm to exit after exceeding its budget of batches rather than n batches), we can call this budget K_* and the improved bounds will hold; alternatively, if a feasible schedule exists with K_* batches this bound also holds.

Corollary 1. *The batch-budgeted algorithm is* $O(\min(n^2 K_*, nK_*^2 \log B))$.

Corollary 2. *The algorithm terminates in* $O(n^2)$ *time for agreeable release times. The batch-budgeted algorithm is* $O(nK_*)$.

Corollary 3. *The algorithm terminates in* $O(n^2 \log n)$ *time for the unbounded case* ($B = \infty$). *The batch-budgeted algorithm is* $O(nK_* \log n)$.

Using the enhanced binary search approach outlined in Condotta et al. [5], $L_{max} = \max C_j - d_j$ minimization can be performed with a $O(n^2 \log n)$ preprocessing phase and by calling a feasibility algorithm $O(\log n)$ times. In addition, by following the approach outlined in Condotta et al. [5], we can easily respect start-start precedence constraints. First, the schedule can be passed through their $O(n^2)$ preprocessing phase, which guarantees that if job a precedes job b then $r_a \leq r_b$ and $d_a \leq d_b$. After generating a schedule, swapping will produce a schedule which obeys the precedence constraints.

3 Scheduling Agreeable Jobs

We now design a faster algorithm to solve the problem with agreeable jobs, where $r_i < r_j \Rightarrow d_i \leq d_j$. We assume the jobs are sorted by increasing deadline (giving non-decreasing release time). Let us describe the structure of the solution we search for. By our previous result, if there exists a feasible partial schedule, there exists a RHBO partial schedule. By a simple swapping argument [10], which does not violate our invariants, we also assume w.l.o.g. that each batch consists of consecutively numbered jobs. Finally, we note that these schedules are "left-shifted" (see e.g. [1]). This implies that given an assignment of jobs to batches, the start time of a batch B_x is fully determined: it must be the maximum of r_j

for all $J_j \in B_x$ and of the time the previous batch on the machine completes (machine assignments remain determined by cyclic scheduling).

We will need to maintain lists of machine availability times: to do this we use purely functional queues [8, 13]. We are given three functions: head(Q) returns the front of the queue Q, tail(Q) returns Q with its front removed, and snoc(X, Q) produces a new queue with X inserted into the back of Q. All of these operations are $O(1)$ and non-destructive. Availability time lists will be maintained sorted ascending order, such that head(A) is the earliest availability time in A. We define a new operation, $U(q, t) = \text{snoc}(\text{tail}(q), t)$. This will be used to *update* availability times: when a new batch is scheduled ending at time t, by cyclic scheduling it runs on the same machine as B_{m+1} in the resultant schedule, formerly (in the previous partial schedule) B_m.

Now we can easily describe the actual algorithm. Let L_i be defined (see below) such that J_{L_i+1} is the earliest job which can be batched together with J_i in a feasible schedule. Consider the RHBO feasible schedule for i jobs: by Invariant (1), the last batch must consist of jobs $J_{L_i+1}, \ldots, J_i$. Upon removing this final batch, observe that a RHBO feasible schedule is left for the first L_i jobs. Thus we inductively assume we have the RHBO schedules for each of the first $j < i$ ($i \leq n$) jobs (from which we can compute L), and then find the only possible RHBO schedule for i jobs (or fail if none exists). Note that L is a non-decreasing function ($L_{i-1} \leq L_i$): this observation makes the tabulation more efficient. F_i is the availability time list for a RHBO schedule of the first i jobs. $E(j, i) = \max\{r_i, \text{head}(F_j)\} + p$ is the left-shifted end time of the last batch in the schedule for i jobs, where the schedule is composed of a batch of jobs $J_{j+1}, \ldots, J_i$ appended to a RHBO schedule for the first j jobs. Formally:

$$L_0 = 0, \quad L_i = \min\left\{j \mid \max\{L_{i-1}, i - B\} \leq j < i, E(j, i) \leq d_{j+1}\right\},$$
$$F_0 = \text{a persistent queue with } m \text{ copies of } 0, \quad F_i = U(F_{L_i}, E(L_i, i)).$$

If at any point L_i is undefined because it minimizes over an empty set, there can exist no RHBO schedule and thus no feasible schedule at all. E and U are not tabulated in the dynamic program. F_n and L_n can be computed in $O(n)$ time. In the case of integer release times and deadlines, the binary search algorithm for L_{max} created by Lee et al. [10] can be combined with our algorithm to solve the multiprocessor problem in $O(n \log(np))$ time.

3.1 Agreeable Processing Times

The relaxation to agreeable processing times was first studied by Li and Lee [11]. Multiprocessor scheduling with no release times and a single deadline ($d_j = d$), which necessarily agrees with the processing times, is unary $\mathcal{NP}$-Hard. However, our algorithm adapts easily to the single processor case.

$$E(j, i) = \max\{r_i, F_j\} + p_i, \quad L_0 = F_0 = 0, \quad F_i = E(L_i, i),$$
$$L_i = \min\left\{j \mid \max(L_{i-1}, i - B) \leq j < i, E(j, i) \leq d_{j+1}\right\}.$$

4 Conclusions

The hardness of multi-processor batch scheduling for the objectives not satisfied by RHBO structure remains an open problem: is a pseudo-polynomial algorithm best possible? If so, what are the best approximation algorithms? Most of these problems are open even when $B = 1$; $\sum C_j$ is a notable exception. Because of Theorem 1, it may be difficult to efficiently minimize $\sum C_j$.

References

1. Philippe Baptiste. Batching identical jobs. *Math. Meth. of O.R.*, 53:355–367, 2000.
2. Amotz Bar-Noy, Sudipto Guha, Yoav Katz, Joseph (Seffi) Naor, Baruch Schieber, and Hadas Shachnai. Throughput maximization of real-time scheduling with batching. In *Proc. of SODA*, pages 742–751, 2002.
3. Peter Brucker. *Scheduling Algorithms*. Springer, 2007.
4. Jessica Chang, HaroldN. Gabow, and Samir Khuller. A model for minimizing active processor time. In Leah Epstein and Paolo Ferragina, editors, *Algorithms ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 289–300. Springer Berlin Heidelberg, 2012. Full version at http://www.cs.umd.edu/ samir/grant/active.pdf.
5. Alessandro Condotta, Sigrid Knust, and Natalia V. Shakhlevich. Parallel batch scheduling of equal-length jobs with release and due dates. *J. of Scheduling*, 13:463–477, October 2010.
6. Christoph Dürr and Mathilde Hurand. Finding total unimodularity in optimization problems solved by linear programs. *Algorithmica*, 59:256–268, 2011.
7. M. R. Garey, D. S. Johnson, B. Simons, and R. E. Tarjan. Scheduling Unit-Time Tasks with Arbitrary Release Times and Deadlines. *SIAM J. on Computing*, 10(2):256–269, 1981.
8. Robert Hood and Robert Melville. Real-time queue operation in pure LISP. *Information Processing Letters*, 13(2):50–54, 1981.
9. Yoshiro Ikura and Mark Gimple. Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 5:61–65, 1986.
10. Chung-Yee Lee, Reha Uzsoy, and Louis A. Martin-Vega. Efficient algorithms for scheduling semiconductor burn-in operations. *Op. Research*, 40(4):764–775, 1992.
11. Chung-Lun Li and Chung-Yee Lee. Scheduling with agreeable release times and due dates on a batch processing machine. *European J. of Operational Research*, 96(3):564 – 569, 1997.
12. Alejandro López-Ortiz and Claude-Guy Quimper. A fast algorithm for multi-machine scheduling problems with jobs of equal processing times. In *STACS*, pages 380–391, 2011.
13. Chris Okasaki. Simple and efficient purely functional queues and deques. *Journal of Functional Programming*, 5(04):583–592, 1995.
14. Barbara Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM J. Comput.*, 12(2):294–299, 1983.