Efficient Scheduling to Minimize Calibrations

Michael A. Bender[*]
Stony Brook University & Tokutek

David P. Bunde[†]
Knox College

Vitus J. Leung[‡]
Sandia National Laboratories

Samuel McCauley[*]
Stony Brook University

Cynthia A. Phillips[‡]
Sandia National Laboratories

ABSTRACT

Integrated Stockpile Evaluation (ISE) is a program to test nuclear weapons periodically. Tests are performed by machines that may require occasional calibration. These calibrations are expensive, so finding a schedule that minimizes calibrations allows more testing to be done for a given amount of money.

This paper introduces a theoretical framework for ISE. Machines run jobs with release times and deadlines. Calibrating a machine requires unit cost. The machine remains calibrated for T time steps, after which it must be recalibrated before it can resume running jobs. The objective is to complete all jobs while minimizing the number of calibrations.

The paper gives several algorithms to solve the ISE problem for the case where jobs have unit processing times. For one available machine, there is an optimal polynomial-time algorithm. For multiple machines, there is a 2-approximation algorithm, which finds an optimal solution when all jobs have distinct deadlines.

Categories and Subject Descriptors

F.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

Keywords

Integrated Stockpile Evaluation, approximation algorithms, calibration, resource allocation, scheduling.

1. INTRODUCTION

[*]Supported in part by NSF grants IIS 1247726, CCF 1217708, CCF 1114809, and CCF 0937822. This work partially done while visiting Sandia National Laboratories. `{bender,smccauley}@cs.stonybrook.edu`.

[†]Partially supported by contract 899808 from Sandia National Laboratories. `dbunde@knox.edu`.

[‡]Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. `{vjleung, caphill}@sandia.gov`.

Integrated Stockpile Evaluation (***ISE***) is a program at Sandia National Laboratories to test nuclear weapons periodically to ensure that the weapons will continue to function according to their specifications [4]. These tests require a great deal of precision and dependability, and given the nuclear context, safety mistakes can have serious ramifications. Testing machines are expensive, and some must be calibrated on a regular basis. The calibrations themselves are also expensive—in a monetary though not necessarily temporal sense. Efficient scheduling of these tests and the calibrations necessary to perform them allows more weapons to be tested within a given budget. This paper gives a theoretical framework, motivated by the ISE problem, for scheduling tasks on machines that need to be calibrated periodically.

We model calibrations in a multi-machine setting as follows. We can ***calibrate*** a machine for unit cost. The machine stays ***calibrated*** for $T \geq 2$ time steps, after which it must remain idle until it is recalibrated. We refer to these T time steps following a calibration as an ***interval***. Calibrating a machine is instantaneous, meaning that a machine can be recalibrated between two job executions running in successive time steps. Minimizing the number of calibrations helps minimize the total cost of an ISE instance.

We abstract the ISE problem by saying that we have a set of jobs $J = \{1, \ldots, n\}$, corresponding to weapons tests. Each job must be scheduled on one of P identical testing machines. A job j has a release time r_j, a due date d_j, and a processing requirement (length) p_j. If a job j is scheduled at time t, we must have $r_j \leq t$ and $t + p_j \leq d_j$. In most of this paper, we assume unit processing times, i.e., $p_j = 1$ for all j. This testing schedule is determined in advance, meaning that the jobs arrive offline.

Our objective is to find the feasible schedule that minimizes the number of calibrations. In the notation developed in [9], we denote the ISE problem as $P|r_j, d_j|\#\text{calibrations}$.

Related Work

One aspect of the ISE problem distinct from many classical scheduling problems is that it often makes sense to delay scheduling a job beyond its earliest feasible time step; see Figure 5. In contrast, for metrics such as minimizing maximum lateness or average completion time, scheduling a job later never helps; see e.g, [11]. This property that a job should be scheduled as early as possible often makes scheduling easier. For example, if the cost function has this property and jobs are restricted to unit size, the problem is polynomial even on multiple machines [5, 13].

One scheduling problem, somewhat reminiscent of the ISE problem, is that of minimizing the number of idle periods [1]. This problem can be solved in polynomial time for unit-sized jobs by dynamic programming. Minimizing idle periods captures the notion that it is expensive to turn on a machine that has been idle, whereas the ISE problem captures the notion that it is expensive to start a constant-sized period of activity. In [1] both machine activation and job processing time contribute to cost, whereas in the ISE problem only the activation intervals are important.

The restriction of unit size jobs can cause some otherwise NP-hard problems to become polynomial [3, 5, 6, 8, 12, 14, 18]. Other problems remain NP-hard; these include flow shop problems on multiple machines with a cost for switching machines [19], requiring additional resources for each job [2, 7, 8], or a partial ordering on the execution order of the jobs [16, 17]. It is not clear if the ISE problem remains NP-hard when restricted to unit-length jobs.

Results

We give the following results for the ISE problem with unit-sized jobs, and show that the problem is NP-complete with arbitrary-sized jobs:

- We give a greedy scheduler for any instance feasible on one machine.
- We show this single-machine solution remains optimal even if more machines become available. However, if an instance requires multiple machines, adding more machines may decrease the optimal cost.
- We give a polynomial-time 2-approximation algorithm for multi-machine ISE instances. This approximation algorithm gives an optimal solution for the special case where all jobs have distinct deadlines.

Overview

The rest of the paper is organized into single-machine results (Section 2) and multiple-machine results (Section 3). We conclude (Section 4) with generalizations that capture further aspects of the ISE application.

2. SINGLE-MACHINE ISE

In this section, we give an optimal algorithm for single-machine ISE. If a scheduling instance is feasible on a single machine, then the best single-machine schedule is optimal, even if more machines are available.

We focus on unit-sized jobs in this paper, since for arbitrary job lengths on any number of machines $P \geq 1$, the ISE problem is NP-hard.

THEOREM 1. *The ISE Problem is (weakly) NP-hard for any number of machines $P \geq 1$.*

PROOF. Hardness follows from a reduction from Partition. □

EDF Scheduling with Intervals

Our ISE algorithms run jobs based on the ***Earliest Deadline First*** (***EDF***) scheduling policy. Although Section 2 discusses the single-machine ISE, we also explain here how EDF works on multiple machines.

The EDF scheduler maintains a priority queue of jobs ordered by increasing deadline. If two jobs have the same deadline, the priority queue breaks ties arbitrarily but consistently.

EDF schedules jobs by iterating over time slots from earliest to latest. When EDF reaches a time slot t during which a new job j arrives ($r_j = t$), EDF adds j to the priority queue. Then EDF selects up to P jobs to run in time step t by popping these jobs from the priority queue.

LEMMA 2 ([10, 15]). *If jobs have unit size, EDF finds a feasible schedule on single or multiple machines if one exists.*

EDF also schedules optimally in the multi-machine ISE setting where machines can run only during specific active length-T intervals and otherwise must remain idle. To see why, create "dummy" jobs for each time step of each inactive machine, where each dummy job is constrained to run in its time step. Given the EDF schedule, we can permute the jobs among machines without changing their running time to create the proper intervals.

Optimal Single-Machine Algorithm

Although EDF always finds a feasible schedule, it may not minimize calibrations. We use EDF as a subroutine in another algorithm that we call ***Lazy-Binning***. Lazy-Binning delays the start of an interval for as long as possible, until delaying it further would make it impossible to find a feasible schedule.

Before giving the algorithm, we define the notion of a ***push***.

DEFINITION 3. *A **push** is a move of an interval from time t to $t + 1$. This may mean that later intervals on the same machine are recursively pushed, so that no two intervals overlap.*

Algorithm 1 Lazy-Binning
```
 1: for Each time step $t$ until all jobs are scheduled do
 2:     if No working interval that contains time $t$ then
 3:         Run EDF starting at time step $t + 1$
 4:         if EDF cannot find a feasible schedule then
 5:             Begin a working interval at time $t$.
 6:         end if
 7:     else
 8:         Schedule jobs in the current interval using EDF
 9:     end if
10: end for
```

Lazy-Binning is the algorithm we obtain by performing all feasible pushes.

THEOREM 4. *Lazy-Binning is optimal.*

PROOF. To obtain a contradiction, assume that Lazy-Binning is not optimal. Consider the time $t \geq 0$ when Lazy-Binning first differs from every optimal schedule. In other words, from time 0 to $t - 1$, there exists at least one optimal schedule OPT that matches Lazy-Binning exactly, but no optimal schedule matches Lazy-Binning through time t. Furthermore, assume without loss of generality that OPT uses EDF to schedule jobs within an interval, as Lazy-Binning does. Thus, there are two ways for Lazy-Binning to differ from OPT:

- In Case A, Lazy-Binning starts an interval at time t while OPT remains idle.
- In Case B, Lazy-Binning does not start an interval at time t, OPT does.

We argue that neither of these cases is possible, leading to a contradiction.

Case A: Lazy-Binning starts an interval at time t, while OPT remains idle. This means that EDF failed to find a feasible schedule starting the next interval at time $t+1$ or later. But OPT is a feasible schedule that starts the next interval at time $t + 1$.

Case B: OPT starts an interval at time t, whereas Lazy-Binning remains idle. Consider a schedule OPT$'$ which has identical intervals to OPT, except that the interval that OPT starts at t is pushed.

OPT$'$ matches Lazy-Binning until time $t + 1$ so it cannot be an optimal schedule by our definition of t. Since OPT$'$ uses the same number of intervals as OPT, the cost for the schedule is the same, so it must be that OPT$'$ is infeasible; it cannot be suboptimal.

Let j be the first job that misses its deadline when EDF schedules jobs in OPT$'$. Any subsequent intervals that we add will be after j's deadline and cannot make OPT$'$ feasible. Therefore, j cannot be feasibly scheduled after the push, which means that Lazy-Binning starts an interval at t. $\square$

We thus have an optimal algorithm for the single-machine case, as well as a test to see if a feasible schedule exists.

Lazy-Binning, as stated in Algorithm 1, runs in pseudopolynomial time, but the algorithm is easily modified to run in polynomial time. The issue is that if release times and deadlines have exponential values, Lazy-Binning may iterate through an exponential number of time steps. However, an interval need never start more than n time steps before the deadline of any job, giving a polynomial solution.

Why Adding Machines Does Not Help

We show that if an instance can be feasibly scheduled on one machine, adding more machines does not decrease the cost.

THEOREM 5. *If the ISE instance is feasible for a single machine, the one-machine schedule is optimal*

The following definitions and lemma are used to prove Theorem 5. A ***configuration*** C for a one-machine scheduling instance is a bit string where the tth bit is 1 if the machine is active at time t, and 0 if the machine is idle. The string indicates when the machine is working, but not which jobs are being executed. We say that a configuration is ***feasible*** if there exists a feasible schedule corresponding to that configuration.

LEMMA 6. *Consider a feasible one-machine scheduling instance on n jobs, and a feasible configuration C on jobs* $1 \dots n-1$ *but not n, and a time t such that* $r_n \leq t < d_n$.

If t is idle let $t_R = t$. *Otherwise, let* t_R *be the earliest idle time step after t, and* t_L *be the latest idle time step before t. Then at least one of the following configurations for jobs* $1, \dots, n$ *is feasible:*

1. C_L*, which is identical to C except that* t_L *is an active time step.*
2. C_R*, which is identical to C except that* t_R *is an active time step.*

PROOF. Suppose that neither configuration is feasible. Let $X = t_R - t_L - 1$ denote the number of time slots between t_L and t_R. Note that this does not include t_L or t_R itself.

Because neither configuration is feasible, $r_n > t_L$ and $d_n \leq t_R$. If this were not the case, job n could be scheduled in t_L or t_R, making C_L or C_R feasible, respectively.

Let EDF[C] denote the schedule obtained by assigning jobs to the active times in configuration C using EDF. Consider the time slots between t_L and t_R in EDF[C]. They are all full, so there must be X jobs running in these time steps; call this set of jobs J_X. There exists some job $i \in J_X$ such that $r_i \leq t_L$ or $d_i > t_R$. Otherwise, $J_X \cup \{n\}$ is a set of $X + 1$ jobs that must be executed in X time steps, which is impossible in a feasible schedule on one machine. Let t_i be the time when i is scheduled in EDF[C]. Assume without loss of generality that i can be scheduled in t_R (if not analyze the mirror image of the schedule where time flows backwards and release times and deadlines trade roles).

There are three cases for t_i:

- Case 1: $t_L < r_n \leq t_i < d_n \leq t_R$.
- Case 2: $d_n \leq t_i < t_R$.

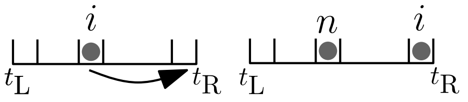

Figure 1: Case 1 from the proof of Lemma 6. Job i can be scheduled in t_R, and job n can be scheduled in t_i where i was previously.

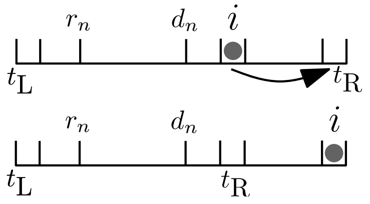

Figure 2: Case 2 from the proof of Lemma 6. Job i can be scheduled in t_R. Job n cannot be feasibly scheduled, but after a finite number of Case 2 instances we must get Case 1, when job n can be scheduled. The top of the figure depicts the schedule before the exchanges (denoted by arrows), and the bottom depicts the schedule after.

- Case 3: $t_L < t_i < r_n$.

The first two cases are shown in Figures 1 and 2, respectively. We prove that Case 1 means that C_R is feasible, Case 2 leads to a smaller instance of Lemma 6, allowing for a finite number of iterated steps until Case 1 is reached, and Case 3 can never occur. Note that we consider cases in order; Case 2 is only used if Case 1 does not apply, and Case 3 is only used if Cases 1 and 2 do not apply.

Case 1 (Figure 1): Schedule job n in time slot t_i and schedule i in t_R (feasible by definition of i). Keep all other jobs scheduled at the same time as in EDF[C]. Thus, C_R is feasible, contradicting our assumption.

Case 2 (Figure 2): Schedule i in t_R. Create a new configuration C' which is the same as C except time slot t_R is set to 1 and t_i is set to 0. Now C' is a new instance of Lemma 6 where t_i becomes the new t_R and t_L stays the same. Each time we hit Case 2, X decreases, so eventually Case 1 must occur (we will show Case 3 never occurs).

Case 3: If there are multiple jobs that meet the conditions for case 3 ($t_L < t_i < r_n$) that can be scheduled in t_R, let i be the one scheduled in the latest time slot in EDF[C].

Consider the set of jobs scheduled strictly between t_i and t_R; we must be able to schedule one in t_i. Otherwise, there are $t_R - t_i - 1$ jobs that cannot be scheduled in t_R or t_i, so each must be scheduled in $t_R - t_i - 1$ time steps. Job n must be scheduled between $r_n > t_i$ and $d_n \leq t_R$, so it must be scheduled in one of these time steps as well. Then there are $t_R - t_i$ jobs that must be scheduled in one of $t_R - t_i - 1$ time steps, which cannot be feasible on one processor.

Let i' be the job that can be feasibly scheduled in t_i. This job has deadline before t_R, since we picked the i that can be scheduled

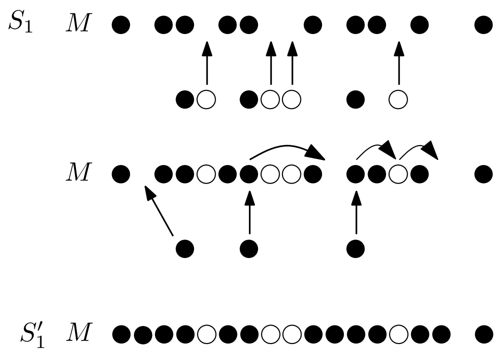

Figure 3: Moving jobs from I to M without increasing the cost. The Type 2 jobs, shown in white, can be moved over immediately, as shown on the first line of the figure. Afterwards, as shown on the second line, the Type 1 jobs shown in black must be moved over using Lemma 6, which may include shuffling some jobs scheduled on M. Finally we obtain S_1', where all jobs from I are scheduled on M.

in t_R that is scheduled latest in EDF[C]. But then i' comes before i in EDF order, so when EDF reached t_i it would have scheduled i' instead of i. Contradiction. □

PROOF OF THEOREM 5: Given a schedule on multiple machines, we move all intervals one at a time to the first machine M. Each time we move an interval I, we only calibrate M one additional time at most. We diagram our strategy in Figure 3.

Suppose we are moving an interval I from machine M' to machine M. There are two types of jobs in I, those that have a job scheduled at the same time on M, which we call Type 1, and those that do not, which we call Type 2. First, move all jobs in I of Type 2 to M, which produces no infeasibility.

Let S_1 and S_1' be the schedule of jobs on M before and after the jobs from I are added, respectively. Let t_F and t_L be the first and last time where S_1 and S_1' differ.

Consider moving a type-1 job that ran at time t on machine M' to machine M. Because this is a type-1 job, machine M is busy at time t. Let t_1 be the last idle interval before t on machine M and let t_2 be the first idle time after t on Machine M. Use the algorithm from Lemma 6 to add a job with desired time t. This algorithm always places a job into either time t_1 or time t_2, extending the block of completely busy time covering time t. Thus the interval from t_F to t_L is completely busy on Machine M in configure S_1'. Therefore S_1' requires at least $q = \lceil \frac{t_L - t_F + 1}{T} \rceil$ intervals to cover the active jobs in S_1 in from t_F through t_L. Because moving one interval from machine M' moves at most T jobs into the interval $t_F \dots t_L$, configuration S_1 is inactive for at most T time steps in this interval. Therefore, configuration S_1 requires at least $q - 1$ intervals to cover jobs in $t_F \dots t_L$ before receiving jobs from interval I.

To cover configuration S_1', place one new interval at the first idle time in configuration S_1 not covered by an interval in configuration S_1 and push the following intervals as necessary, up to T time units, to eliminate interval overlap. The interval before t_F is covered by intervals as before in configuration S. The time between t_F and t_L is completely covered by q adjacent (touching) intervals.

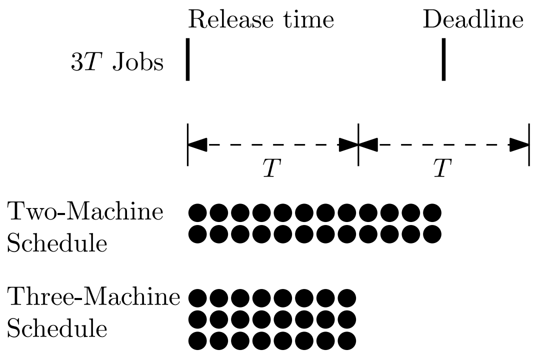

Figure 4: This schedule is feasible on two machines, but optimal on three. There are $3T$ jobs with a release time of zero and a deadline of $3T/2$. On two machines four calibrations are necessary, but with three matchines only three calibrations are necessary.

Each time after t_L that was covered in S_1 is still covered in S_1'. Either the original interval doesn't move, or it is pushed. But if an interval is pushed, then it is part of a set of adjacent intervals extending forward at least to t_F. Thus anything covered by the pushed interval is still covered by the new set of intervals. □

Theorem 5 does not extend to instances that require at least two machines for feasibility. With more machines, it is no longer true that using the minimum number of machines gives the optimal number of intervals. Consider an instance with kT jobs released at time 0 with common deadline $1.5T$, $k > 2$. These can be run on k machines with k intervals from 0 to T, but require $k + 1$ intervals on $k - 1$ machines. Figure 4 illustrates $k = 3$. Note that for $k = 2$, it is not feasible to go down to a single machine. By using this structure repeatedly, we see that removing a machine may increase the number of intervals by more than one.

3. ISE ON MULTIPLE MACHINES

We first characterize the structure of optimal solutions to the ISE problem on multiple machines. We then give a 2-approximation algorithm. We describe conditions under which our algorithms find optimal solutions.

Round Robin Machine Assignment

Let $\mathcal{I} = (I_1, I_2, \dots, I_N)$ be a sequence of intervals, a partial solution for a set of jobs, ordered by increasing start time. Ties are broken consistently. For our multiple-machine algorithm, ties can be broken by interval creation time, since the algorithm is incremental. Without loss of generality, no two intervals in $\mathcal{I}$ on the same machine have start times within T of each other.

Each interval $I_j \in \mathcal{I}$ is scheduled on some machine P_j. Assigning intervals to machines can be done using a round-robin method for any feasible schedule.

LEMMA 7. *Given a feasible schedule S on P machines, there exists a schedule having the same cost and the same assignments of jobs to intervals where $P_j \equiv j \pmod P$.*

PROOF. We use an inductive argument on $N = |\mathcal{I}|$, the number of intervals. The lemma is true for $N = 1$, where we can place the first interval on the first machine.

Suppose we have legally placed the first k intervals. Let t denote the time when I_{k+1} begins. If scheduling I_{k+1} on machine $(k+1)$

$\pmod{P}$ is not feasible, there exists another interval I_c scheduled at time t_c on P_{k+1} such that $t - T < t_c \leq t$. Since all previous intervals are scheduled using round-robin, there are $P-1$ intervals starting between times t_c and t. Each of these must also be scheduled at or before t and after $t-T$, so in total there are $P+1$ intervals that must be scheduled in that range. Since there are P processors, two of the intervals must be scheduled concurrently on the same machine, which is infeasible. This contradicts the assumption that the schedule of the first k intervals is feasible. □

Lemma 7 means that our algorithm need not explicitly determine an assignment of intervals to machines.

Our algorithm returns a sequence of interval start times. Given this sequence, we assign intervals to machines using round robin and schedule jobs within intervals using EDF.

Before proceeding, we further specify how EDF breaks ties.

INVARIANT 8. *When EDF removes job j from the priority queue to run at time step t, there may be multiple machines on which j could run. EDF breaks ties by assigning j to the first interval $I \in \mathcal{I}$ that does not yet have a job assigned at time slot t (but contains time slot t).*

Characterization of Pushes

When there are multiple machines, intervals can overlap provide they are on different machines. This leads to more subtlety in how we characterize a push.

DEFINITION 9. *A **push** is a move of an interval $I \in \mathcal{I}$ from time t to $t+1$ that may recursively push later intervals in order to maintain the following:*

1. *The ordering of intervals in $\mathcal{I}$ does not change. Thus, any other interval that starts at t and is ordered after I in $\mathcal{I}$ is also pushed.*
2. *The intervals on each processor are nonoverlapping. Thus, any interval that starts at $t+T$ on the same processor as I is also pushed.*

Suppose the push of an interval results in a new set of intervals $\mathcal{I}'$ and let t be the latest start time of any interval in $\mathcal{I}'$. This push is *feasible* for an instance of jobs if by augmenting $\mathcal{I}'$ by zero or more intervals, each starting at time t or later, all jobs can be feasibly scheduled.

We categorize feasible pushes of the last currently-scheduled interval into four cases. Because we only consider pushing the last interval, we do not need to worry about cascades of pushes. There may be jobs in J that cannot be scheduled in any of the intervals.

DEFINITION 10. *Each feasible push of the last interval I fits one of four cases. When I is pushed, there may exist some job in I before the push that EDF no longer schedules after the push; denote this job j_{out}. Similarly, there may exist some job not scheduled before the push that EDF schedules after the push; denote this job j_{in}.*

Case 1: *There exists no job j_{out}.*
Case 2: *There exists a job j_{out} but no job j_{in}.*
Case 3: *Job j_{out} is after j_{in} in EDF order.*
Case 4: *Job j_{out} is before j_{in} in EDF order.*

Framework for Optimal Solutions

We extend Lazy-Binning to multiple machines.

DEFINITION 11. *A **push algorithm** repeatedly creates a new interval at the end of $\mathcal{I}$ and then pushes that interval. The algorithm*

- *performs all Case-1 and Case-3 pushes,*
- *never performs an infeasible push, and*
- *has the flexibility to choose which Case-2 and Case-4 pushes to perform.*

Once the algorithm stops pushing the interval, if there are remaining unscheduled jobs, the algorithm creates a new interval and begins pushing again.

At intermediate steps of a push algorithm, EDF assigns jobs to intervals only temporarily. EDF may assign some job j to some interval, but when an interval is pushed or a new interval is added to $\mathcal{I}$, re-running EDF may assign j to a different interval.

When $P = 1$, all feasible pushes are Case 1 and Case 3, so Lazy-Binning is a push algorithm. Theorem 4 proves a push algorithm is optimal for $P = 1$. Theorem 18 shows that an optimal push algorithm always exists for any number of processors.

Algorithm 2 Push Algorithm

```
 1: $t \leftarrow 0$
 2: **while** Not all jobs are scheduled **do**
 3:     Remove all assignments of jobs to intervals
 4:     **if** $t$ is contained in $P$ or more intervals **then**
 5:         $t \leftarrow$ next time with $\leq P-1$ intervals
 6:     **end if**
 7:     Create a new interval $I$ at $t$
 8:     Schedule jobs in all intervals with EDF
 9:     **while** Pushing $I$ is Case 1 or Case 3 **do**
10:         Push $I$
11:         $t \leftarrow t+1$
12:     **end while**
13:     Perform 0 to $T$ Case-2 or Case-4 pushes on $I$
14: **end while**
```

LEMMA 12. *A push of an interval I starting at t is of type Case 2 or Case 4 if and only if*

1. *before this push there is a $t_f \leq t+T$ such that jobs are scheduled in all calibrated time slots between t and t_f*
2. *each of these jobs is due no later than t_f, and*
3. *the push is feasible.*

The smallest such t_f has the property that $t_f - t \geq 2$ and at least two jobs are due exactly at t_f.

PROOF. We first explain why $t_f - t$ must be greater than 1. If $t_f - t = 1$ then there exists a job j that must be scheduled at time t, the first time step of I before the push. Scheduling j later would miss its deadline. Thus after the push, when I starts at $t+1$, EDF does not schedule j in I. By Invariant 8, EDF schedules in the earliest possible interval in $\mathcal{I}$, so if j could be feasibly scheduled in an earlier interval it would be scheduled there before the push. Thus, j cannot be scheduled in an interval before I in $\mathcal{I}$. All later intervals must also start after the deadline of j by Definition 9, so j cannot be scheduled there either. Thus, if $f = 1$ the push is infeasible.

(⇐) If these jobs do exist, there must be a j_{out}, because after we push I forward there are more jobs that must run between t and t_f than there are calibrated time steps between times t and t_f. The push cannot be Case 3 because before the push, any jobs released by time $t+T$ that precede j_{out} in EDF order would have been scheduled before j_{out} and they still will be after the push. If j_{in} is released at $t+T$, its deadline is at least $t+T+1$. This is after j_{out} because j_{out} has deadline no later than $t_f \leq t+T$.

(⇒) We now show that if there is a Case-2 or Case-4 push, such a t_f must exist. Let t_f be the deadline of j_{out}. We must have $t_f \leq t+$

T because if j_{out} could be scheduled at $t+T$, it could be scheduled in I after the push, ruling out Case 2. In Case 4, j_{out} is replaced in the set of scheduled jobs by a job later in EDF order. That would not happen if j_{out} could be feasibly scheduled in the new set of intervals.

After the push, all Y calibrated time slots from t to t_f must contain jobs since j_{out} cannot be scheduled. The configuration after the push is the same as before except that the slot in I at time t is replaced by a slot at time $t + T$. The EDF schedule is the same up to the slot at time t that is removed. This is the last slot in time t since I is the last interval. If there is no job in that slot, then the rest of the schedule can run as before and there is no j_{out}, which contradicts the type of push. The job j_b that ran at time t in I must compete with newly-released jobs at time $t + 1$. Some may have earlier deadlines than j_b so j_b may not recieve the very next slot. If j_b is never scheduled, it is j_{out} and all slots are full through t_f. Otherwise j_b is scheduled at some time $t < t_b \le t_f$. By the same arguments as above, that slot must have held a job before the push. This job is either j_{out}, delayed by other jobs till past its deadline, or it displaces another till the cascade finally ends with the real j_{out}. All slots are then full until j_{out} fails to meet its deadline at time t_f.

All Y jobs the run between t and t_f in the pushed schedule also ran between t and t_f before the push. All jobs affected by the push (including j_{out}) ran no earlier than time t before the push. Every job in this set runs no earlier after the push (most at the same time except those involved in the cascade). All finish by t_f since only j_{out} was pushed later (to failure). Job j_{out} also ran before t_f before the push, since t_f is job j_{out}'s deadline. Thus, there were $Y + 1$ jobs running in the $Y + 1$ slots between time t and t_f before the push.

None of the $Y + 1$ jobs just described has a deadline later than t_f. Consider the discussion above about how the push affects the schedule. All jobs scheduled between times $t + 1$ and t_b, when the displaced job j_b is finally scheduled, had deadlines no larger than job j_b's deadline. This is because j_b was scheduled later by EDF. Job j_c, displaced by j_b, has a deadline no earlier than j_b's deadline and no earlier than the deadline for any job scheduled between t_b and t_c. Thus the displaced jobs have monotonically increasing deadlines and the current displaced job has a deadline no smaller than that of any job schedule since time t. Thus, job j_{out} has deadline no smaller than that of any job that ran between t and t_f.

We now show that at least two jobs are due exactly at the smallest t_f. Consider the smallest t_f. Because $t_f - t > 1$, there is at least one job scheduled at time $t_f - 1$ among the Y jobs described above (in interval I, for example). There is at least one job scheduled before $t_f - 1$ due after $t_f - 1$; otherwise $t_f - 1$ would satisfy the definition for t_f, violating the assumption that this is the smallest t_f. All jobs scheduled at $t_f - 1$ are due at t_f. There must be at least one such job in interval I. Thus, there are at least two jobs due at t_f. □

COROLLARY 13. *Every push algorithm does no more than T Case-2 or Case-4 pushes on any interval.*

PROOF. Since there exists a job in I with deadline no later than t_f, by definition of the j_{out} for the push, I cannot be pushed past t_f. Otherwise that job could not be feasibly scheduled. Since $t_f \le T$ the lemma follows. □

LEMMA 14. *If intervals are added to a schedule, EDF schedules all jobs no later than without the added intervals.*

PROOF. Let S and S' be the schedule before and after the intervals are added. Assume the contrary: there exists a job j that is scheduled later in S' than in S. Let j be the first such job in EDF order and let t be the time when j is scheduled in S. When EDF reaches t in S', there must be some job j' before j in EDF order that has not been scheduled (otherwise EDF would schedule j). But j' must have already been scheduled when EDF reached t in S (otherwise EDF would schedule j' at t in S). Then j' is scheduled later in S' than in S, which contradicts our definition of j. □

LEMMA 15. *Consider a Case-1 or Case-3 push of an interval I. If we remove any set of jobs J' from J, this will still be a Case-1 or Case-3 push.*

PROOF. It is immediate that removing jobs can never result in infeasibility.

We now show that removing jobs cannot cause a push to become Case 2 or Case 4. Assume the contrary, that the push of interval I at time t becomes Case 2 or Case 4 after removing J'. By Lemma 12, there must be some time step t_f where each calibrated time slot between t and t_f is running a job that must be scheduled before t_f. In each time slot, removing J' can only cause a job with a later deadline to be scheduled by EDF. Then before J' is removed, all calibrated time slots between t and t_f were already active with jobs with potentially earlier deadlines. Then this was a Case-2 or Case-4 push initially, leading to a contradiction. □

LEMMA 16. *Performing Case-1 and Case-3 pushes can never increase the number of calibrations. That is, there is always an optimal schedule that performs a Case-1 or Case-3 push whenever one is available during the incremental schedule construction.*

PROOF. To obtain a contradiction, assume the contrary: no optimal schedule performs all available Case-1 and Case-3 pushes. Let OPT be the schedule that performs the most Case-1 and Case-3 pushes. We show that we can obtain a schedule that performs one more Case-1 or Case-3 push without increasing the cost, reaching a contradiction.

A Case-1 or Case-3 push is only defined on the last interval placed so far while we are making the schedule, so we refer to the push as Case 1 or Case 3 if the push is Case 1 or Case 3 with subsequent intervals removed. That is, EDF schedules jobs from the beginning after this removal to determine the case of the push.

We show that we can feasibly schedule all jobs in OPT after another Case-1 or Case-3 push on some interval I without increasing the number of intervals. Let t be the start time of the first interval I_1 in OPT that did not have all Case-1 or Case-3 pushes applied. So a push on I_1 is Case 1 or 3. Let I be the last interval started at time t. A push on I must also be Case 1 or 3. If it were Case 2 or 4, then by Lemma 12 there must be a time t_f before which all calibrated time slots must contain jobs with deadlines before t_f. But since EDF schedules in time slots in order by interval, all jobs in time slots before t_f in I_1 must also have deadlines before t_f. This means I_1 is Case 2 or 4, which contradicts our definition. Thus pushing I is Case 1 or 3. Furthermore, all subsequent intervals in OPT start at $t + 1$ or later, so pushing interval I preserves the ordering of the starting times of the intervals.

Let t be the time when I starts in OPT. For now, assume that all later intervals in OPT start strictly later than t. We will consider the case where one starts at t later in this proof.

We set the stage for an exchange argument. Schedule all jobs in OPT using EDF. There is a (possibly empty) set of jobs J' that are (1) scheduled between times t and $t + T - 1$ inclusive, and (2) scheduled in an interval after I in OPT. Figuratively, we remove these jobs and set their time slots as inactive. What we really do is peg these jobs to these time slots for our exchange, so that their positions in the schedule are not determined by rerunning EDF.

Perform the push on I in OPT with J' removed; now we have a sequence of intervals OPT$'$. When I was the last interval placed so far, the push was remains a Case-1 or Case-3 push. Now define j_{out} and j_{in} as in Definition 10. By Lemma 15, job j_{out}, if it exists, is released before j_{in} and has a deadline after j_{in} in EDF order. To summarize, j_{out} and j_{in} are defined, assuming that we run the schedule only allocating intervals up to I.

When we transform OPT into OPT$'$, we push I, and run EDF on OPT$'$ ignoring the jobs in J' and their time slots. Interval I ends at $t+T$. Let j'_{out} be a job scheduled before $t+T$ (in any interval) in OPT but after $t+T$ and not in I in OPT$'$; let j'_{in} be any job scheduled in an interval after $t+T$ in OPT but before $t+T$ or in I in OPT$'$.

Then j'_{in} is j_{in} and j'_{out} is j_{out} if it exists. This is because when I was the last interval, the push was Case 1 or Case 3. After adding subsequent intervals, none of the jobs scheduled in I are scheduled later than $t+T$, nor will a later job be scheduled in I or earlier than $t+T$.

Now we exchange jobs to show that OPT$'$ is feasible. We schedule the intervals up to the pushed I according to (the new) EDF. We schedule the jobs strictly after the interval I ends according the old EDF, except that in the slot where j'_{in} used to be, we replace that with j'_{out}. We keep the jobs J' exactly where they were before.

Then we have performed an extra Case-1 or Case-3 push of I without increasing the cost of the schedule, contradicting our assumption that we have the OPT with the most Case-3 pushes.

As described, this exchange argument leaves out an important detail. This argument applies when interval I has a gap before the next interval on the same processor. If there is no gap, that is, the two intervals are adjacent, then we need to adjust our argument slightly. A simple adjustment is just to view the adjacent intervals as one bigger interval and then to define the j'_{out} where j'_{in} according to this superinterval. □

LEMMA 17. *A set of k Case-2 pushes on an interval I can only be a part of an optimal push algorithm if, after the Case-2 pushes are performed, the next push would be a Case-4 push.*

PROOF. We will show that no job in J can be scheduled in the newly-calibrated time slots after the Case-2 pushes. Let J_I be the set of jobs feasibly scheduled by EDF in the first I intervals, before I is pushed. Let J'_I be the jobs that cannot be feasibly scheduled, so $J'_I = J \setminus J_I$. Since the pushes are Case 2, none of the jobs in J'_I can be scheduled in I after the Case-2 pushes on I. Adding subsequent intervals can only cause jobs to be scheduled earlier by Lemma 14, so no job in J_I is scheduled at a later point after we add intervals later than I in $\mathcal{I}$. Thus, no job in either J'_I or J_I can be scheduled in the newly-calibrated time slots.

By Lemma 12, after a Case-2 push, any further feasible pushes on the interval are Case 2 or Case 4.

Thus, any newly calibrated time slots due to the push remain inactive, and the pushes cannot decrease the cost of the remaining schedule unless there is a later Case-4 push. □

THEOREM 18. *There exists a sequence of Case-2 and Case-4 pushes such that the corresponding push algorithm is optimal.*

PROOF. This proof generalizes Theorem 4. All Case-1 pushes are identical to the single-machine case, and can still always occur without added cost. We show that Case-3 pushes never add to cost in Lemma 16. The remainder of the argument proceeds as in Theorem 4. □

2-Approximation Algorithm

We now give a 2-approximation algorithm (Algorithm 3). Algorithm 3 is not a push algorithm, though it has structural similarities. Observe that the algorithm also becomes Lazy-Binning if $P = 1$.

Algorithm 3 pushes intervals similarly to a push-algorithm. It never performs an infeasible push and performs all Case-1 and Case-3 pushes of each interval I. Whenever it sees an opportunity for an interval I to have a Case-4 push or a series of Case-2 pushes followed by a Case-4 push, Algorithm 3, instead, creates another interval I' right after I. These extra intervals are ignored for the purpose of determining the case of a push; otherwise the pushed interval may not be the last one scheduled. Like Lazy-Binning, this algorithm is pseudopolynomial as written. If release times and deadlines have exponential values, Algorithm 3 may iterate through an exponential number of time steps. However, an interval need never start more than n time steps before the deadline of any job. Incorporating this restriction makes Algorithm 3 run in polynomial time.

Algorithm 3 Lazy-Binning on Multiple Machines

```
 1: $t \leftarrow 0$; $I \leftarrow \emptyset$
 2: while Not all jobs are scheduled do
 3:     Remove all assignments of jobs to intervals
 4:     if $P$ or more intervals overlap with $I$ then
 5:         $t \leftarrow$ next time with $\leq P-1$ intervals
 6:     end if
 7:     Create a new interval $I$ at $t$
 8:     while Pushing $I$ is Case 1 or Case 3 do
 9:         Push $I$
10:         $t \leftarrow t+1$
11:     end while
12:     $t' \leftarrow t$
13:     while Pushing $I$ from $t'$ is Case 2 do
14:         $t' \leftarrow t'+1$
15:     end while
16:     if Pushing $I$ is Case 4 then
17:         Create another interval at $t+T$
18:     end if
19: end while
```

LEMMA 19. *The set of jobs scheduled in the first i intervals of any push algorithm A can be feasibly scheduled after i rounds of Algorithm 3 (a round refers to an iteration of the while loop in Step 2).*

PROOF. By Corollary 13, any time when a time slot is calibrated in A's schedule, the corresponding time slot is also calibrated in Algorithm 3's schedule. Thus, for the first i intervals, we can copy the assignments of jobs to machines and times from A directly to time slots calibrated by Algorithm 3, giving a feasible schedule. □

COROLLARY 20. *Algorithm 3 gives a solution with cost no more than twice optimal, i.e. it is a 2-approximation.*

COROLLARY 21. *Algorithm 3 is optimal if all deadlines are distict.*

PROOF. From Lemma 12, a Case-4 push occurs only when multiple jobs have the same deadline. Thus, if no two jobs have the same deadline, Algorithm 3 adds no extra intervals and gives an optimal solution. □

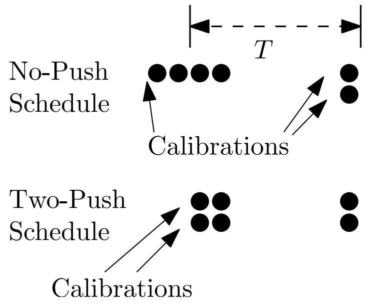

Figure 5: An example showing that both Case-2 and Case-4 pushes are sometimes necessary to obtain an optimal solution. This is also an example where Algorithm 3 gives a solution that is $3/2$ times optimal. Four jobs have release time 0 and deadline 4. Two jobs have release time $2+T$ and deadline $3+T$. Algorithm 3 gives the solution shown in the top of the figure with a cost of three. Pushing the first interval twice (first a Case-2, then a Case-4 push) gives the solution shown in the bottom of the figure, with a cost of 2.

LEMMA 22. *Algorithm 3 is no better than a 3/2-approximation.*

PROOF. Consider an ISE instance with four jobs that have release time 0 and deadline 4, and two jobs that have release time $T + 2$ and deadline $T + 3$; see Figure 5. Algorithm 3 gives a solution with three intervals. However, the optimal solution pushes the first interval twice (a Case-2 and a Case-4 push) and gives a solution with two intervals. □

4. CONCLUSION

The complexity of the ISE problem with unit-sized jobs on multiple machines remains open. We suspect that the analysis of Algorithm 3 can be tightened, given that it performs better than twice optimal on all instances we have constructed.

As a next step we hope to generalize our model to capture more aspects of the actual ISE problem. For example, machines may not be identical, and calibrations may require machine time. Moreover, some jobs may not have unit size. We hope that efficient constant-factor approximations are still possible on more general instances of the ISE problem.

5. REFERENCES

[1] P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 364–367, 2006.

[2] J. Blazewicz, J.K. Lenstra, and A.H.G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[3] P. Brucker. *Scheduling Algorithms*. Springer, New York, 2007.

[4] Chris Burroughs. New integrated stockpile evaluation program to better ensure weapons stockpile safety, security, reliability, March 2006. `http://www.sandia.gov/LabNews/060331.html`.

[5] M.I. Dessouky, B.J. Lageweg, J.K. Lenstra, and S.L. van de Velde. Scheduling identical jobs on uniform parallel machines. *Statistica Neerlandica*, 44(3):115–123, 1990.

[6] M.M. Dessouky. Scheduling identical jobs with unequal ready times on uniform parallel machines to minimize the maximum lateness. *Computers and Industrial Engineering*, 34(4):793–806, 1998.

[7] M.M. Dessouky, M.I. Dessouky, and S.K. Verma. Flowshop scheduling with identical jobs and uniform parallel machines. *European Journal of Operational Research*, 109(3):620–631, 1998.

[8] M.R. Garey and D.S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4(4):397–411, 1975.

[9] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5(2):287–326, 1979.

[10] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Technical report, Management Science Research Project Research Report 43, University of California, Los Angeles, 1955.

[11] D. Karger, C. Stein, and J. Wein. Scheduling algorithms. *CRC Handbook of Computer Science*, 1997.

[12] W. Kern and W.N. Nawijn. Scheduling multi-operation jobs with time lags on a single machine. In *Twente Workshop on Graphs and Combinatorial Optimization*, pages 81–86, 1991.

[13] B.J. Lageweg, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Computer aided complexity classification of deterministic scheduling problems. Technical Report BW 138/81, Stichting Mathematisch Centrum, Amsterdam, 1981.

[14] Y. Lin and W. Li. Parallel machine scheduling of machine-dependent jobs with unit-length. *European Journal of Operational Research*, 156(1):261–266, 2004.

[15] B. Simons and M. Sipser. On scheduling unit-length jobs with multiple release time/deadline intervals. *Operations Research*, 32(1):80–88, 1984.

[16] J.A. Stankovic, M. Spuri, M. Di Natale, and G.C. Buttazzo. Implications of classical scheduling results for real-time systems. *Computer*, 28(6):16–25, 1995.

[17] J. D. Ullman. Polynomial complete scheduling problems. *ACM SIGOPS Operating Systems Review*, 7(4):96–101, January 1973.

[18] W. Yu. *The two-machine flow shop problem with delays and the one-machine total tardiness problem*. PhD thesis, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, 1996.

[19] W. Yu, H. Hoogeveen, and J.K. Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5):333–348, 2004.