J Sched (2009) 12: 501–515
DOI 10.1007/s10951-009-0126-8

Algorithms for testing fault-tolerance of sequenced jobs

Marek Chrobak · Mathilde Hurand · Jiří Sgall

Published online: 25 August 2009

Abstract We study the problem of testing whether a given set of sequenced jobs can tolerate transient faults. We present efficient algorithms for this problem in several fault models. A fault model describes what types of faults are allowed and specifies assumptions on their frequency. Two types of faults are considered: *hidden faults*, that can only be detected after a job completes, and *exposed faults*, that can be detected immediately.

First, we give an $O(n)$-time fault-tolerance testing algorithm, for both exposed and hidden faults, if the number of faults does not exceed a given parameter k.

Then we consider the model in which any two faults are separated in time by a gap of length at least Δ, where Δ is at least twice the maximum job length. For exposed faults, we give an $O(n)$-time algorithm. For hidden faults, we give an algorithm with running time $O(n^2)$, and we prove that if job lengths are distributed uniformly over an interval $[0, p_{\max}]$, then this algorithm's expected running time is $O(n)$. Our experimental study shows that this linear-time performance extends to other distributions. Finally, we provide evidence that improving the worst-case performance may not be possible, by proving an $\Omega(n^2)$ lower bound, in the algebraic computation tree model, on a slight generalization of this problem.

Keywords Scheduling · Fault-tolerance · Real-time systems · Algorithms

M. Chrobak's research is supported by NSF grants CCR-9988360, CCR-0208856, OISE-0340752 and CCF-0729071.
J. Sgall is partially supported by Institutional Research Plan No. AV0Z10190503, by Inst. for Theor. Comp. Sci., Prague (project 1M0545 of MŠMT ČR), and grants IAA1019401 and IAA100190902.

M. Chrobak (✉)
Department of Computer Science, University of California, Riverside, CA, USA
e-mail: marek@cs.ucr.edu

M. Hurand
Department d'Informatique (LIX), Ecole Polytechnique, Palaiseau, France

J. Sgall
Department of Applied Mathematics, Charles University, Malostranské nám. 25, 11800 Praha 1, Czech Republic
e-mail: sgall@kam.mff.cuni.cz

1 Introduction

Ghosh et al. (1995), Mosse et al. (2003) (see also Egan et al. 1999) studied the problem of testing fault-tolerance of a collection of sequenced jobs. More specifically, we are given a sequence J of jobs, with release times, deadlines, and processing times (or lengths). The jobs in J have already been sequenced, that is, their order of execution is known. Transient faults may occur when jobs are executed. If a fault occurs, the currently executed job is re-executed. In (Ghosh et al. 1995; Mosse et al. 2003), the authors assume that a fault can be detected only after the processing of a job is complete. We refer to such faults as *hidden faults*. The question investigated in (Ghosh et al. 1995; Mosse et al. 2003) is whether all jobs in J will meet their deadlines in the presence of faults. The answer is, obviously, negative when arbitrary fault patterns are allowed. However, with reasonable assumptions on the frequency of faults, the question becomes meaningful and, in some cases, non-trivial. In (Ghosh et al. 1995; Mosse et al. 2003), the authors assume the fault frequency model in which a gap between any two faults is at least Δ, where Δ is at least twice the maximum job length. For this model, they present an $O(n^2)$-time fault-tolerance testing algorithm, under the restriction that all jobs are released at the same time. In addition, they also propose a linear-time heuristic for this problem. The authors also extend this linear-time heuristic to jobs with arbitrary release times, and discuss its applications and experimental results.

A different fault frequency model, in which the number of faults is bounded by some constant k, has been suggested by Liberato et al. (2000). For this model, the authors give an $O(n^2k)$-time dynamic programming algorithm for testing fault-tolerance, if jobs are ordered according to EDF and preemption is allowed.

Testing for fault-tolerance and enhancing schedules to improve their fault-tolerance are significant issues in real-time systems with hard deadlines, where missing a deadline by a job may result in a malfunction of the whole system. Although scheduling approaches as those discussed above can only guarantee limited fault-tolerance, they still provide useful tools. A system designer can choose a fault-tolerance model most appropriate for its application, and determine the expected level of its fault tolerance (say, the value of Δ or k, in the models discussed above). Alternatively, one can use the desired value of Δ or k to determine either the maximum load conditions or the hardware requirements needed to meet these fault-tolerance goals. The experiments reported in (Ghosh et al. 1995) show that, in fact, even if the faults do not strictly match the model, the task loss is minimal. We refer the reader to (Ghosh et al. 1995; Mosse et al. 2003) for more background on this problem and discussion of its practical aspects. See also Sect. 7 for a brief discussion of more general models.

Our results In this paper, the jobs are already ordered (as in the previous work), and preemption is not allowed. We consider both fault frequency models from (Ghosh et al. 1995; Mosse et al. 2003; Liberato et al. 2000) in this paper. In addition to the hidden faults, we also consider another type of faults that we call *exposed*. Unlike hidden faults, exposed faults can be detected immediately, and the running job can be restarted from scratch at the time when a fault occurs.

First, we discuss the issue of scheduling. A *schedule* assigns to each job its planned start time, that is the time when the job would be started if the previous jobs are not delayed due to faults. The *greedy schedule* starts each job either at its release time or at the completion time of the previous job, whichever comes later. In the exposed-fault model, it is not difficult to see that it is sufficient to consider only greedy schedules. For hidden faults, however, there are fault patterns for which it is beneficial, in some situations, to delay execution of a job and stay idle for some time. In Sect. 3, we prove that this cannot happen if the set of all possible fault sequences satisfies the following *sparsifiability* property: if a certain fault sequence can occur, then any sparser sequence can occur as well (see Sect. 2 for precise definitions). This is a very natural restriction on potential fault sequences and all the fault frequency models we consider are sparsifiable. Thus throughout the paper we can restrict our attention to greedy schedules only.

We then propose a number of fault-tolerance testing algorithms. Our first algorithm is for the fault frequency model NUM_k, where the number of faults is bounded by k. This algorithm runs in time $O(n)$ (for both fault types), independently of k. In (Aydin 2004), the fault model from (Liberato et al. 2000) is extended so that a re-execution of a job could take time different from its processing time. Our algorithm can be adapted to handle a similar case as well.

Then we consider the fault frequency model GAP_Δ, introduced in (Ghosh et al. 1995; Mosse et al. 2003), in which any two consecutive faults are separated by a gap at least Δ. As in (Ghosh et al. 1995; Mosse et al. 2003), we assume that Δ is at least twice the maximum processing time. (Thus each job can fail at most once.) For exposed faults, we give an algorithm that runs in time $O(n)$.

The case of hidden faults in the GAP_Δ model is more difficult. Here, we present an algorithm with running time $O(n^2)$. Our algorithm applies to jobs with arbitrary release times, generalizing the work from (Ghosh et al. 1995; Mosse et al. 2003). We further show that if job lengths are distributed uniformly then the running time of this algorithm is $O(n)$ with high probability. (And thus its expected running time is $O(n)$ as well.) This result is, in fact, much more general, as it holds for all probability distributions in which certain sub-intervals of $[0, \Delta/2]$ have non-zero probability. (See Sect. 6.2.) We also include the results of an experimental study that confirms our analysis.

Whether the worst-case running time can be improved remains an open question. However, we provide evidence that such an improvement is unlikely, by showing that a slight generalization of this problem cannot be solved faster than in time $\Omega(n^2)$ in the algebraic computation tree model.

All our algorithms are very simple, efficient, and easy to implement. The basic idea behind all these algorithms is similar: For each fault model, we first show that one needs to consider only some specific "cruel" fault patterns. With this restriction, using dynamic programming, we design an algorithm that for each job computes its latest completion time on faults that are "cruel" for this model. Comparing these completion times with the deadlines, we determine whether the given set of jobs is fault-tolerant.

Other related work Substantial work has been done on fault tolerant scheduling in multiprocessor systems. For example, Liberato et al. (1999) studied scheduling of periodic preemptive real-time jobs in the presence of transient faults. A different model, with processor faults and non-periodic and non-preemptive tasks was investigated by (Manimaran and Siva Ram Murthy 1998). Kalyanasundaram and Pruhs (1997) studied fault-tolerant scheduling from the perspective of competitive analysis. (See Liberato et al. 1999; Manimaran and Siva Ram Murthy 1998; Qin et al., 2000, 2002; Girault et al. 2004; Kalyanasundaram and Pruhs 1997 and references therein for other work on this and related topics.)

2 Terminology and notation

Jobs and schedules By J we denote the sequence of n jobs on input. Jobs are identified by integers $1, 2, \ldots, n$. Each job j is specified by a triple (r_j, d_j, p_j), where r_j is its *release time*, d_j is its *deadline*, and p_j is its *processing time*. Without loss of generality, we assume that $0 \le r_j < r_j + p_j \le d_j$ for all j. By $p_{\max} = \max_j p_j$ we denote the maximum processing time.

A *schedule* of J is any sequence $s = (s_1, \ldots, s_n)$ such that $s_j \ge r_j$ for all j, and $s_{j+1} \ge s_j + p_j$ for $j < n$. We refer to s_j as the *scheduled start time* of job j.

Without loss of generality, throughout the paper we assume that $r_{j+1} \ge r_j + p_j$ for all $j < n$. For any set of jobs J, we can easily modify, in linear time, the release times in J to satisfy this property, without affecting job completion times for any schedule of this sequence of jobs. (Recall that the order of jobs is already fixed in advance.) The *greedy schedule* for J is then defined simply by $s_j = r_j$ for all j.

Faults Each fault is specified by a real number, namely the time of the fault. *Fault sequences* (or *patterns*) are denoted by letters f, g, h. We assume that the faults in these sequences are listed in increasing order, that is, if $f = (f_1, f_2, \ldots, f_m)$ then $f_1 < f_2 < \cdots < f_m$. By $|f|$ we denote the length of sequence f. (We allow infinite sequences as well, in which case $|f| = \infty$.)

A *fault frequency model* is a set F of potential fault sequences. F is called *sparsifiable* if for all $f \in F$, any $1 \le a \le b \le |f|$, and any fault sequence g with $|g| = b - a + 1$, if $f_{a+i-1} - f_{a+i-2} \le g_i - g_{i-1}$ for $i = 2, \ldots, b - a + 1$, then $g \in F$ as well. Intuitively, this means that any sequence "sparser" than a segment of a sequence in F is also in F.

The two particular fault frequency models we consider are:

GAP_Δ: The set of all sequences f in which $f_i - f_{i-1} \ge \Delta$ for each i, and

NUM_k: The set of all sequences f with at most k faults.

Both models are easily seen to be sparsifiable. As we show later in the paper, for sparsifiable models, we can restrict ourselves to studying only greedy schedules.

Completion times Next, we explain how execution of a job is affected when a fault occurs. This depends on the type of faults under consideration.

Fix a sequence of n jobs J and a fault model F. By $S_j(s, f)$ and $C_j(s, f)$ we denote the *start time* and *completion time* of job j, if we execute the jobs according to schedule s and the fault sequence is f. Informally, $S_j(s, f)$ is either s_j or the completion time of job $j - 1$, whichever is greater. If no fault occurs between $S_j(s, f)$ and $S_j(s, f) + p_j$, then $C_j(s, f)$ equals $S_j(s, f) + p_j$. If a fault occurs in this interval, j will need to be re-executed, starting either at the fault time or at $S_j(s, f) + p_j$, depending on whether we consider exposed or hidden faults. The completion time is the time when j has been fully processed without faults.

We now give a rigorous definition. Initially, set $S_1(s, f) = s_1$. Then, for $j = 1, \ldots, n$, assume that $S_j(s, f)$ has been defined, and proceed as follows:

(C) The completion time $C_j(s, f)$ depends on the fault type:
  - (CE) For exposed-faults, $C_j(s, f)$ is the smallest $\tau \ge S_j(s, f) + p_j$ such that $f \cap (\tau - p_j, \tau] = \emptyset$, that is, the interval $(\tau - p_j, \tau]$ contains no faults.
  - (CH) For hidden-faults, $C_j(s, f)$ is the smallest $\tau \ge S_j(s, f) + p_j$ such that $f \cap (\tau - p_j, \tau] = \emptyset$ and $\tau - S_j(s, f)$ is an integer multiple of p_j.

(S) If $j < n$, then the start time of job $j + 1$ is $S_{j+1}(s, f) = \max\{s_{j+1}, C_j(s, f)\}$.

For the sake of brevity, as in the definition above, we sometimes treat f as a *set* of real numbers (fault times), so that we can apply to it set-theoretic operations.

In (CE) and (CH), if such a τ does not exist, job j (and all subsequent jobs) never completes. Note that in the case of a "tie", when a fault occurs exactly at a time when some job j completes its execution and job $j + 1$ is about to start, we assume that the fault affects job j but not $j + 1$. (All the results remain valid if we assumed that $j + 1$ is affected instead of j, or even if the choice of the affected job was arbitrary.)

By $C_j(s, F)$ we denote the maximum completion time of a job j if the faults are from F, that is, $C_j(s, F) = \max_{f \in F} C_j(s, f)$. Throughout the paper, we will simplify notation by omitting the arguments that are understood from context, for example, $S_j(s)$, $C_j(F)$, C_j, etc.

For either fault type, exposed or hidden, a schedule s of J is called *F-tolerant*, if each job completes by its deadline, that is, $C_j(s, F) \le d_j$ for all j. All algorithms we present will actually compute, for all j, the maximum completion times $C_j(s, F)$. Testing fault-tolerance, that is, whether $C_j(s, F) \le d_j$ for all j, can then be done trivially in linear time.

3 Fault tolerance and greedy schedules

In this section, we show that we can restrict ourselves to greedy schedules only. For two schedules s, t, we write $s \prec t$ if $s_i \le t_i$ for all i.

Recall that the job sequence is given, and we cannot reorder the jobs. It is quite intuitive that attempting to execute each job as soon as possible in the given order should yield the most fault-tolerant schedule. We prove this in the two lemmas below.

Lemma 1 *For exposed faults, for any fault frequency model F, if J has any F-tolerant schedule then the greedy schedule for J is F-tolerant.*

Proof Fix any fault sequence f and schedules s, t such that $s \prec t$. It is enough to show the following claim:

$(*)$ $C_j(s, f) \le C_j(t, f)$ for all j.

Indeed, if $(*)$ holds, to get the lemma we take s to be the greedy schedule. If J has any F-tolerant schedule t, then $s \prec t$, and thus $(*)$ implies that the greedy schedule is F-tolerant as well.

We show inequality $(*)$ by induction on j. For $j = 0$, define artificially $C_0(s, f) = C_0(t, f) = 0$, and the claim holds trivially.

Suppose that $j \ge 1$ and $C_{j-1}(s, f) \le C_{j-1}(t, f)$. Then $\tau = C_j(t, f)$ satisfies $f \cap (\tau - p_j, \tau] = \emptyset$ and $\tau \ge S_j(t, f) + p_j = \max\{t_j, C_{j-1}(t, f)\} + p_j \ge \max\{s_j, C_{j-1}(s, f)\} + p_j = S_j(s, f) + p_j$. Thus $C_j(s, f) \le C_j(t, f)$ as well. □

Lemma 1 does not hold for hidden faults. For hidden faults, it is possible that J has an F-tolerant schedule even though the greedy schedule is not F-tolerant. For example, take $J = \{(r_1, d_1, p_1) = (0, 5, 3)\}$ (just one job) and $F = \{(1)\}$, one fault sequence with a single fault at time 1. Schedule $s = (2)$ is F-tolerant, but the greedy schedule $s = (0)$ is not. However, we show that a lemma analogous to Lemma 1 holds for hidden faults if we assume that F is sparsifiable.

Lemma 2 *For hidden faults, for any sparsifiable fault frequency model F, if J has any F-tolerant schedule then the greedy schedule for J is F-tolerant.*

Proof The proof is a little harder than that of Lemma 1, although the general idea is similar. Fix any fault sequence f and schedules s, t such that $s \prec t$. It is enough to show the following claim:

$(*)$ For any $b \in J$ and $f \in F$ there is $g \in F$ such that $C_b(s, f) \le C_b(t, g)$.

That $(*)$ is sufficient to prove the lemma should be quite obvious: Take s to be the greedy schedule. If t is any F-tolerant schedule, then $s \prec t$, and thus $(*)$ implies that the greedy schedule is F-tolerant as well.

It is sufficient to prove $(*)$ for the special case where s and t differ in just one start time, say $t_m = s_m + \epsilon$, for some $\epsilon > 0$, and $t_j = s_j$ for $j \ne m$. For otherwise, if $s \prec t$ are arbitrary, we can define schedules $s = s^0 \prec s^1 \prec \cdots \prec s^l = t$, where each two consecutive schedules s^q, s^{q+1} differ in only one start time. If $(*)$ holds for any pair of consecutive schedules $s' = s^q$, $t' = s^{q+1}$, then it holds for s, t as well.

Without loss of generality, $C_b(s, f) < \infty$. Fix b, and pick the largest $a \le b$ such that $S_a(s, f) = s_a$. Thus jobs $a, a+1, \ldots, b$ execute back-to-back, some possibly several times. Without loss of generality, we can assume that all faults in f occur in $(s_a, C_b(s, f)]$, since we can remove other faults without changing the value of $C_b(s, f)$. (Note that removing faults at the beginning or end of f creates a fault sequence that is still in F.) We choose g depending on the value of m. There are three cases to consider.

Case 1: $m > b$. This is the easiest case since here we can simply take $g = f$. The execution of jobs $1, 2, \ldots, b$ is the same in f and g, so $C_b(s, f) = C_b(t, g)$.

Case 2: $a \le m \le b$. Let $\epsilon' = \max\{t_m - C_{m-1}(s, f), 0\}$ be the amount of time by which m will be delayed if we change its start time from s_m to t_m. Define $g_i = f_i$ for all $f_i \le S_m(s, f)$ and $g_i = f_i + \epsilon'$ for all $f_i > S_m(s, f)$. Since F is sparsifiable, $g \in F$, and in g each fault will hit the same job as in f. Therefore, $C_b(s, f) \le C_b(t, g)$, and we are done.

Case 3: $m < a$. Recall that, according to our assumption, no fault occurs before $S_a(s, f)$. Denote by $\theta = \sum_{j=m+1}^{a} \max\{s_j - C_{j-1}(s, f), 0\}$ the total idle time between $C_m(s, f)$ and s_a. We have two sub-cases.

If $\epsilon \le \theta$, then increasing s_m to t_m will not change the start time of job a, and therefore we can simply take $g = f$.

In the other case, when $\epsilon > \theta$, the jobs $a, a+1, \ldots, b$ will be started earlier by $\epsilon' = \epsilon - \theta$. We take $g_i = f_i + \epsilon'$ for all i. Then $g \in F$, and (similarly to Case 2) each fault will hit the same job as in f. Therefore, $(*)$ holds in this case as well. □

The following lemma follows almost directly from the definitions:

Lemma 3 *Both fault frequency models NUM_k and GAP_Δ are sparsifiable.*

Proof Consider first NUM_k. The definition of sparsifiability requires that for any $f \in \mathsf{NUM}_k$, any sequence g not longer than f and sparser than a segment of f of length $|g|$ is also in NUM_k. Since NUM_k contains *all* sequences with at most k faults, this condition is satisfied vacuously.

For GAP_Δ, the proof is equally simple: Suppose that $f \in \mathsf{GAP}_\Delta$ and let g be any sequence with $|g| = b - a + 1$, where $1 \le a \le b \le |f|$, that satisfies $f_{a+i-1} - f_{a+i-2} \le g_i - g_{i-1}$ for $i = 2, \ldots, b - a + 1$. Since $f_{a+i-1} - f_{a+i-2} \ge \Delta$ for all $i = 2, \ldots, b - a + 1$, we get $g_i - g_{i-1} \ge \Delta$ as well, implying that $g \in \mathsf{GAP}_\Delta$. □

From the lemmas above, throughout the rest of the paper we can assume that the jobs are scheduled greedily, and we will use notation $S_j(f)$, $C_j(f)$, etc., for the start time and completion time in the greedy schedule. Also, we will say that a job sequence J is *F-tolerant* if the greedy schedule for J is F-tolerant.

4 Sequences with at most *k* faults

In this section, we give a linear-time algorithm for testing fault tolerance when $F = \mathsf{NUM}_k$, that is, F consists of all sequences with at most k faults, where k is a given parameter. By the results from the previous section, we can assume that the jobs are scheduled according to the greedy schedule. The general idea of the algorithm is that in the worst case all faults will affect just one "critical" job.

Lemma 4 *For both exposed and hidden faults, for each $b \in J$ and $f \in \mathsf{NUM}_k$, there is a $g \in \mathsf{NUM}_k$ that causes one job in J to execute $k+1$ times, and for which $C_b(g) \ge C_b(f)$. In addition, all the faults in g appear at the end of execution of that job.*

The fault sequences g from the above lemma constitute the *cruel sequences* for NUM_k.

Proof Pick the smallest a such that the jobs $a, \dots, b$ are executed back-to-back, that is, $S_a(f) = r_a$ and $S_j(f) = C_{j-1}(f)$ for $j = a+1, \dots, b$. We can assume, without loss of generality, that all faults in f occur in the interval $(r_a, C_b(f)]$. Let e, $a \le e \le b$, be the job in this block that has the largest processing time p_e.

We first give the argument for hidden faults. If there is any fault in the interval $(C_e(f), C_b(f)]$, we can do this: (i) remove the last fault from f, (ii) increase the time of all faults in f after $C_e(f)$ by p_e, and (iii) add one fault during the last execution of e, that is, in the interval $(C_e(f) - p_e, C_e(f)]$. Let f' be the new fault sequence. There are at most k faults in f', and, by the choice of e, this change can only increase the completion time of b, that is, $C_b(f') \ge C_b(f)$. By repeating this process we eliminate all faults after the completion of e.

So suppose now that e is executed $l+1$ times, due to l faults, and that there are no faults after $C_e(f)$. If there are any faults in the interval $(S_a(f), S_e(f)]$, remove the last such fault. This will decrease the start time of e by some amount $\delta \le p_e$. Modify f by decreasing times of all the faults on e by δ, and then add one more fault on the last execution on e. Let the resulting sequence be f'. There are at most k faults in f' and, by the choice of e, the above modification can only increase the completion time of b, so $C_b(f') \ge C_b(f)$.

Overall, the process above transforms f into another sequence g in which all faults (at most k) occur during the executions of e, and which satisfies $C_b(g) \ge C_b(f)$. If the number of faults is smaller than k, we can add another fault on the last execution of e. By repeating this, we obtain a sequence with k faults on e.

Finally, to satisfy the second requirement in the lemma, each fault can be shifted to the end of the corresponding execution of job e, without any change in the resulting completion times.

The proof for exposed faults is similar. The main observation is that for exposed faults, using a similar shifting argument, we can assume that all faults occur at completions of executions of jobs, that is at times of the form $S_j(f) + ip_j$, for $i = 1, 2, \dots, \ell - 1$, for some j, where $C_j(f) = S_j(f) + \ell p_j$. The rest of the argument is the same as for hidden faults. □

Algorithm 1 given below will compute the latest completion time $C_j^* = C_j(\mathsf{NUM}_k)$ for each job j. To test fault-tolerance, one then only needs to check if $C_j^* \le d_j$ for all j. Note that Lemma 4 implies that the cruel sequences and the completion times for them are the same for hidden and exposed faults, so we can handle both cases at once.

Algorithm 1 — Computing the $C_j^* = C_j(\mathsf{NUM}_k)$ for both fault types

```
$C_0^* \leftarrow 0$
for $j = 1, \dots, n$ do
    $C_j^* \leftarrow \max\{C_{j-1}^* + p_j, r_j + (k+1)p_j\}$.
```

Analysis Clearly, Algorithm 1 works in linear time. We need to show that the completion times are computed correctly, that is, $C_j^* = C_j(\mathsf{NUM}_k)$ for all j.

The proof is by induction. To make the base case easy to handle, we artificially set $C_0(\mathsf{NUM}_k) = 0$.

The ($\le$) inequality is quite easy: Suppose it holds for indices $0, 1, \dots, j-1$, and consider job j. Using a fault sequence $f \in \mathsf{NUM}_k$ that forces j to execute $k+1$ times, we get $C_j(f) \ge r_j + (k+1)p_j$. If $g \in \mathsf{NUM}_k$ is a fault sequence that realizes $C_{j-1}(\mathsf{NUM}_k)$, then $C_j(g) \ge C_{j-1}(g) + p_j = C_{j-1}(\mathsf{NUM}_k) + p_j \ge C_{j-1}^* + p_j$, by induction. Therefore, $C_j(\mathsf{NUM}_k) \ge \max\{C_j(f), C_j(g)\} \ge C_j^*$.

Now we show the ($\ge$) inequality. In other words, we claim that for any job j and fault sequence $f \in \mathsf{NUM}_k$, $C_j^* \ge C_j(f)$. Again, assume that this holds for jobs $1, 2, \dots, j-1$. By Lemma 4, we can assume that on f some job j' is re-executed k times. If $j < j'$ then j starts at r_j (recall that $r_{i+1} \ge r_i + p_i$ for all i) and executes once, so j is completed no later than C_j^*. For $j = j'$, job j completes at time

$r_j + (k+1)p_j \le C^*_j$. For $j > j'$, job j is completed at time $\max\{r_j, C^*_{j-1}\} + p_j \le C^*_j$. Thus each job j completes no later than at time C^*_j, as claimed.

In conclusion, we obtain the following theorem.

Theorem 5 *For both fault types, Algorithm 1 computes in time $O(n)$ the latest completion times for all jobs j, in the presence of up to k faults (that is, for the fault model* NUM_k*).*

5 Exposed Δ-faults

In this section, we consider the fault model $F = \mathsf{GAP}_\Delta$, in which all fault sequences f satisfy $f_i - f_{i-1} \ge \Delta$ for all i, where Δ is some parameter of the problem. Recall that, as in (Ghosh et al. 1997), we assume that $\Delta \ge 2p_{\max}$. We give a linear-time algorithm for testing fault-tolerance in this model in the case of exposed faults.

As in the previous section, the idea is to show that only some special fault sequences need to be considered.

Lemma 6 *In the greedy schedule, for each $b \in J$ and $f \in \mathsf{GAP}_\Delta$, there is a $g \in \mathsf{GAP}_\Delta$ in which each fault occurs at the completion time of the first execution of some job, and for which $C_b(g) \ge C_b(f)$.*

Proof Pick the largest $a \le b$ such that $S_a(f) = r_a$. Thus jobs $a, a+1, \ldots, b$ execute back-to-back, some possibly twice. Without loss of generality we can assume that all faults in f occur in the interval $(r_a, C_b(f)]$, since we can remove all other faults without affecting the value of $C_b(f)$.

If f satisfies the condition in the lemma, we take $g = f$ and we are done. Otherwise, let f_l be the last fault in f that does not satisfy the condition in the lemma. Let also e, $a \le e \le b$, be the job affected by f_l, that is, $S_e(f) < f_l \le S_e(f) + p_e$. For $\delta = S_e(f) + p_e - f_l$, define a new fault sequence f' where $f'_i = f_i$ if $i < l$ and $f'_i = f_i + \delta$ for $i \ge l$. Then $f' \in \mathsf{GAP}_\Delta$, $C_j(f') \ge C_j(f)$ for all $j = a, \ldots, b$, and f' has more faults satisfying the condition in the lemma than f. So after repeating this process, we transform f into a desired fault sequence g. □

The algorithm For each j, we define $\alpha(j)$ as the minimum index a such that $\sum_{i=a}^{j} p_i \le \Delta$. (In other words, $\sum_{i=a}^{j} p_i \le \Delta$ and either $a = 1$ or $\sum_{i=a-1}^{j} p_i > \Delta$.) Let also $\pi(j) = \sum_{i=\alpha(j)}^{j} p_i$. Note that if a fault occurs during an execution of j and jobs $\alpha(j), \ldots, j$ are executed back-to-back then no fault could have occurred on these jobs.

The algorithm is shown below in pseudocode. It first precomputes the numbers $\alpha(j)$ and $\pi(j)$. Then it uses the numbers $\alpha(j)$ and $\pi(j)$ to compute $C^*_j = C_j(\mathsf{GAP}_\Delta)$, for each j. To determine whether J is GAP_Δ-tolerant, one then only needs to check if $C^*_j \le d_j$ for all j.

Algorithm 2 — Computing $C^*_j = C_j(\mathsf{GAP}_\Delta)$ for exposed faults

```
// Compute the numbers $\alpha_j, \pi_j$
$\alpha(1) \leftarrow 1; \pi(1) \leftarrow p_1$
for $j = 2, \ldots, n$ do
    $a \leftarrow \alpha(j-1); x \leftarrow \pi(j-1) + p_j$
    while $x > \Delta$ do
        $x \leftarrow x - p_a; a \leftarrow a + 1$
    $\alpha(j) \leftarrow a; \pi(j) \leftarrow x$
// Compute the completion times
$C^*_0 \leftarrow r_1$
for $j = 1, \ldots, n$ do
    $C^*_j \leftarrow \max\{C^*_{j-1} + p_j, r_j + 2p_j, C^*_{\alpha(j)-1} + \pi(j) + p_j\}$
```

Running time By a standard amortization argument, it takes time $O(n)$ to compute all numbers $\alpha(j)$ and $\pi(j)$. The linear-time complexity of computing the completion times is obvious.

Correctness We now show that the numbers C^*_j are computed correctly, that is, $C^*_j = C_j(\mathsf{GAP}_\Delta)$ for all j. The proof is by induction. We artificially set $C_0(\mathsf{GAP}_\Delta) = 0$, so that the equality holds in the base case $j = 0$.

We prove the ($\le$) inequality first. Assume that this inequality holds for indices $0, 1, \ldots, j-1$, and consider job j. If f consists of just one fault at the end of the execution of j, then $C_j(f) \ge r_j + 2p_j$. Next, choosing $g \in \mathsf{GAP}_\Delta$ to be the fault that realizes $C_{j-1}(\mathsf{GAP}_\Delta)$, we get $C_j(g) \ge C_{j-1}(g) + p_j = C_{j-1}(\mathsf{GAP}_\Delta) + p_j \ge C^*_{j-1} + p_j$, by induction. Finally, take $h \in \mathsf{GAP}_\Delta$ that realizes $C_{\alpha(j)-1}(\mathsf{GAP}_\Delta)$. Add to h a fault at the end of the execution of j. This new fault sequence h' is still in GAP_Δ, by the definition of $\alpha(j)$ and $\pi(j)$. Therefore, $C_j(h') \ge C_{\alpha(j)-1}(h) + \pi(j) + p_j = C_{\alpha(j)-1}(\mathsf{GAP}_\Delta) + \pi(j) + p_j \ge C^*_{\alpha(j)-1} + \pi(j) + p_j$, by induction. Putting it together, we get $C_j(\mathsf{GAP}_\Delta) \ge \max\{C_j(f), C_j(g), C_j(h')\} \ge C^*_j$.

Next, we prove the ($\ge$) inequality. We need to show that $C^*_j \ge C_j(f)$ for each j and each $f \in \mathsf{GAP}_\Delta$. Assume that the claim holds for $0, 1, \ldots, j-1$. For the current job j, we consider cases depending on whether the last fault occurred. If j is executed without faults, then $C_j = \max\{r_j, C_{j-1}\} + p_j \le \max\{r_j, C^*_{j-1}\} + p_j \le C^*_j$. Suppose now that a fault occurs at job j, so j is executed twice. Without loss of generality, this fault is at the end of its first execution. If j starts at r_j then $C_j(f) = r_j + 2p_j \le C^*_j$. Otherwise j must have been delayed because of a previous fault on a job of index smaller or equal to $\alpha(j)$. So the jobs $\alpha(j) - 1, \ldots, j-1$ must have been executed back-to-back with jobs $\alpha(j), \ldots, j-1$ executing without faults. Therefore, $C_j = C_{\alpha(j)-1} + \pi(j) + p_j \le C^*_{\alpha(j)-1} + \pi(j) + p_j \le C^*_j$. We conclude that the algorithm is correct.

Theorem 7 *For exposed faults, Algorithm 2 computes in linear time the maximum completion times when all faults are separated by gaps of length at least Δ (that is, for the fault model GAP_Δ).*

6 Hidden Δ-faults

The algorithm from (Ghosh et al. 1997) verifies fault-tolerance for hidden Δ-faults if all jobs are ready at the same time. Their method is to divide the sequence of jobs into blocks of length at most Δ, where each block includes an additional unallocated *recovery interval* whose length is at least the longest processing time of the jobs. Then they compute the partition that minimizes the total execution time. This approach does not work for jobs with different release times because some job sequences can be executed fault-tolerantly but do not have such partition into blocks. Consider, for example, a sequence J of jobs in which job j has start time $3j-3$, deadline $3j+1$, and all jobs have execution time equal 2. Let $\Delta = 6$. With the greedy schedules, for all $f \in \mathsf{GAP}_\Delta$, all jobs in J will meet their deadlines, but it will not be possible if we use a schedule where some job is postponed.

General idea The general approach we take is similar to those in the previous section. We identify certain "cruel" fault sequences on which completion times of jobs are maximized. By focusing on these sequences, we can derive a dynamic programming algorithm for computing maximum completion times for all jobs. This algorithm, for each j, will compute pairs (c, δ), where c is a possible completion time of j and $c-\delta$ is the time of the latest fault before c. This is all the information needed to determine the latest completion times of the jobs $j+1, \dots, n$. To achieve polynomial-time, we show that we need to keep track of only those pairs (c, δ) that are not dominated by other—in the sense defined formally below.

To simplify the argument, it is convenient to slightly modify the interpretation of faults as follows: If a fault occurs at a time τ and a job j starts at time τ, then we will assume that this fault affects job j and j is re-executed (not job $j-1$, even if it ends at τ). We claim that this modification does not change the worst-case completion times. Indeed, for any fault sequence f that possibly contains some faults on the beginning of jobs (with the new interpretation), we can choose a sufficiently small $\epsilon > 0$ and increase all fault times by ϵ, so that afterwards all faults are on exactly the same jobs, but not on their start times. The completion times of all jobs on this new sequence f' will be the same as on f. Similarly, for any fault sequence that possibly contains some faults on the end of jobs (with the old interpretation), we can choose a sufficiently small $\epsilon > 0$ and decrease all fault times by ϵ with no change of completion times.

Let J be a set of jobs. A fault sequence $f \in \mathsf{GAP}_\Delta$ is called *cruel for* J if for all $f_i \in f$, either $f_i - f_{i-1} = \Delta$ or f_i occurs at a beginning of some job. The above conditions imply that each cruel fault sequence can be divided into *chains*, where in each chain the faults are at distance exactly Δ. We define CRUEL^0_J to be the set of fault sequences in GAP_Δ that are cruel for J.

Lemma 8 *In the greedy schedule, for each $b \in J$ and $f \in \mathsf{GAP}_\Delta$, there is a $g \in \mathsf{CRUEL}^0_J$ for which $C_b(g) \ge C_b(f)$.*

Proof For convenience, for all fault sequences f we will set $f_0 = -\infty$. Fix b and f. Pick the largest $a \le b$ such that $S_a(f) = r_a$. Thus jobs $a, a+1, \dots, b$ execute back-to-back, some possibly twice. Let f' be the fault sequence obtained from f by removing all faults before S_a and after C_b. Then $f' \in \mathsf{GAP}_\Delta$ and $C_b(f') = C_b(f)$.

Now take the first fault f'_i that does not satisfy the definition of the cruel sequence. Suppose that f'_i occurs when a job j is executed, that is, $f'_i \in (S_j(f'), S_j(f') + p_j)$. By the choice of i, we have $f'_i - f'_{i-1} > \Delta$ (and this holds even if $i = 1$ because $f'_0 = -\infty$). We can modify f' by moving f'_i to $\max\{f'_{i-1} + \Delta, S_j(f')\}$ without affecting the completion time of jobs $j, j+1, \dots, b$. After this change, f' will be still in GAP_Δ. By repeating this process for each fault, we turn f' into a sequence $g \in \mathsf{CRUEL}^0_J$. □

For a job j and a fault sequence f, suppose that f_i is the time of the last fault in f before $C_j(f)$, that is, $f_i = \max_{i'}\{f_{i'} \mid f_{i'} < C_j(f)\}$. As explained earlier, the idea of our algorithm is to keep track of such pairs $(C_j(f), C_j(f) - f_i)$, as these pairs determine the start time of the next job and the earliest possible time when a fault can occur. Lemma 8 implies that we only need to be concerned with cruel fault sequences. This still does not yield a polynomial-time algorithm, as even for cruel sequences the number of such pairs to keep track of could be exponential. We reduce the complexity by discarding from consideration fault sequences that are "redundant", namely those that cannot maximize the completion time of any job after j.

First, we note that, once $C_j(f) - f_i \ge \Delta$, the next fault can occur immediately, and it is not necessary to remember the exact value of the difference. Instead, in such a pair, we always use Δ in the second component. We define $\delta_j(f) = \min(C_j(f) - f_i, \Delta)$.

We now formalize the idea of redundancy. A pair $(\tilde{c}, \tilde{\delta})$ is said to *dominate* a pair (c, δ) if $\tilde{c} \ge c$, $\tilde{\delta} \ge \delta$, and at least one of these inequalities is strict. The dominance relation is clearly a (strict) partial order.

We extend the definition of dominance to fault sequences. For two fault sequences $f, g \in \mathsf{CRUEL}^0_J$ and a job k, we say that f *k-dominates* g if $(C_k(f), \delta_k(f))$ dominates $(C_k(g), \delta_k(g))$. (In case when $(C_k(f), \delta_k(f)) =$

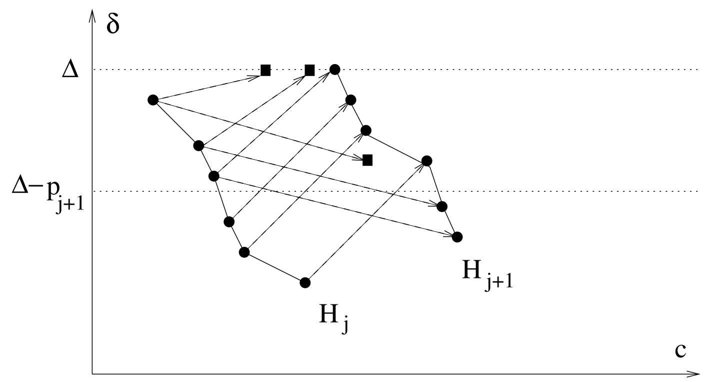


Fig. 1 Building H_{j+1} from H_j.The pairs marked by squares are eliminated

$(C_k(g), \delta_k(g))$, to break the tie, we further require that f is lexicographically smaller than g.) For each k, the k-dominance relation is a partial order on CRUEL^0_J. By $\mathsf{CRUEL}^k_J \subseteq \mathsf{CRUEL}^0_J$ we denote the set of cruel sequences that are not j-dominated by another sequence, for any $j = 1, 2, \ldots, k$. In other words, for $k > 0$, CRUEL^k_J is the set of all $f \in \mathsf{CRUEL}^{k-1}_J$ such that f is not k-dominated by any $g \in \mathsf{CRUEL}^{k-1}_J$. We now prove that, in order to compute the worst-case completion times, it is sufficient to consider only the sequences in the sets CRUEL^k_J.

Lemma 9 *For any $f \in \mathsf{CRUEL}^{k-1}_J - \mathsf{CRUEL}^k_J$ and $b \geq k$, there exists a $g \in \mathsf{CRUEL}^k_J$ such that $C_b(f) \leq C_b(g)$.*

Proof Fix any $f \in \mathsf{CRUEL}^{k-1}_J - \mathsf{CRUEL}^k_J$, and choose a fault sequence $h \in \mathsf{CRUEL}^{k-1}_J$ that k-dominates f. Without loss of generality, we can assume that h is maximal with respect to the k-dominance relation, that is, $h \in \mathsf{CRUEL}^k_J$. Let g be a sequence that contains the faults in h that affect jobs $1, 2, \ldots, k$, as well as the faults from f that affect jobs $k+1, \ldots, n$, appropriately shifted so that they hit the same jobs and at the same places as in f. Then $g \in \mathsf{GAP}_\Delta$ because $f, h \in \mathsf{GAP}_\Delta$ and h k-dominates f. Also, $C_b(g) \geq C_b(f)$ because $C_k(h) \geq C_k(f)$. The gap in g from job k to job $k+1$ could violate the definition of a cruel sequence, but using the method from Lemma 8 we can modify the part of g after job k to satisfy this definition, getting a cruel sequence g for which $C_b(g) \geq C_b(f)$. And finally, since all faults of g up to job k are from h and $h \in \mathsf{CRUEL}^k_J$, we have $g \in \mathsf{CRUEL}^k_J$. □

Corollary 10 *If a job $b \in J$ is CRUEL^b_J-tolerant, then it is CRUEL^0_J-tolerant.*

Proof Let $f \in \mathsf{CRUEL}^0_J$. Let k be the first job for which $f \in \mathsf{CRUEL}^{k-1}_J - \mathsf{CRUEL}^k_J$. Then, according to Lemma 9, there exists a g in CRUEL^k_J with $C_b(g) \geq C_b(f)$. By repeating this argument as many times as necessary, we will find an $h \in \mathsf{CRUEL}^b_J$ with $C_b(h) \geq C_b(f)$. □

The algorithm As before, we view the problem as the optimization problem in which we wish to compute the worst-case completion time, $C^*_j = C_j(\mathsf{GAP}_\Delta)$, for each j. By Lemma 8 and Corollary 10, for each j we have $C^*_j = C_j(\mathsf{CRUEL}^0_J) = C_j(\mathsf{CRUEL}^j_J)$.

The algorithm maintains the set $H_j = \{(C_j(f), \delta_j(f)) \mid f \in \mathsf{CRUEL}^j_J\}$, for $j = 1, 2, \ldots, n$. In other words, H_j is the set of pairs (c, δ) such that for some fault sequence $f \in \mathsf{CRUEL}^j_J$ we have $c = C_j(f)$ and $\delta = \delta_j(f)$. Given H_j, the maximum completion time of j can be determined easily, as we have $C^*_j = \max\{c \mid (c, \delta) \in H_j\}$.

To compute the sets H_j, we initially start with $H_0 \leftarrow \{(-\infty, \Delta)\}$. Then for $j = 1, 2, \ldots, n$, since $\mathsf{CRUEL}^j_J \subseteq \mathsf{CRUEL}^{j-1}_J$, we can use the definition of cruel sequences to construct H_j from H_{j-1}. (See the pseudo-code below.)

The complete algorithm is given below in pseudo-code. An example illustrating the recursive construction of the sets H_j (for the case without the release times) is shown in Fig. 1.

Algorithm 3 — Computing the $C^*_j = C_j(\mathsf{GAP}_\Delta)$ for hidden faults

```
$H_0 \leftarrow \{(-\infty, \Delta)\}$
for $j = 1, 2, \ldots, n$ do
    $H_j \leftarrow \emptyset$
    for each $(c, \delta) \in H_{j-1}$ such that $c \geq r_j$ do
        if $\delta + p_j \leq \Delta$ then
            add $(c + p_j, \delta + p_j)$ to $H_j$
        else
            add $(c + 2p_j, \delta + 2p_j - \Delta)$ and $(c + p_j, \Delta)$ to $H_j$
    if $\min\{c \mid (c, \delta) \in H_{j-1}\} < r_j$ then
        add $(r_j + p_j, \Delta)$ and $(r_j + 2p_j, 2p_j)$ to $H_j$
    Eliminate dominated pairs from $H_j$
    $C^*_j \leftarrow \max\{c \mid (c, \delta) \in H_j\}$
```

Correctness Take $f \in \mathsf{CRUEL}^j_J$. We claim that H_j is computed correctly, that is,

$$H_j = \{(C_j(f), \delta_j(f)) \mid f \in \mathsf{CRUEL}^j_J\}. \quad (1)$$

The proof is by induction. Given our definition of H_0, the basis case $j = 0$ is trivial. Suppose that the claim is true up to step $j - 1$, that is, $H_{j-1} = \{(C_{j-1}(f), \delta_{j-1}(f)) \mid f \in \mathsf{CRUEL}^{j-1}_J\}$. We prove that (1) holds.

($\subseteq$) For any $(c', \delta') \in H_j$, we need to find a corresponding $f' \in \mathsf{CRUEL}^j_J$. This is quite straightforward. Suppose that $(c, \delta) \in H_{j-1}$. By induction, there is an $f \in \mathsf{CRUEL}^{j-1}_J$ such that $(c, \delta) = (C_{j-1}(f), \delta_{j-1}(f))$. We extend f to f', depending on which case in the algorithm holds.

Consider $(c, \delta) \in H_{j-1}$ with $c \geq r_j$. If $\delta + p_j \leq \Delta$, then the algorithm adds $(c + p_j, \delta + p_j)$ that corresponds to $f' = f$. Otherwise, we add $(c + 2p_j, \delta + 2p_j - \Delta)$, that corresponds to f' obtained from f by adding a fault at the earliest possible time during the execution of job j, and $(c + p_j, \Delta)$ that corresponds to $f' = f$.

If there exists $(c, \delta) \in H_j$ with $c < r_j$, then the empty fault sequences corresponds to $(r_j + p_j, \Delta)$ and the fault sequence (r_j), with just one fault at r_j, corresponds to $(r_j + 2p_j, 2p_j)$.

($\supseteq$) Take $f \in \mathsf{CRUEL}^j_J$. We argue that $(C_j(f), \delta_j(f))$ will be added to H_j by the algorithm. That this pair will not be eliminated at the end of the jth iteration follows from f being in CRUEL^j_J.

Since $\mathsf{CRUEL}^j_J \subseteq \mathsf{CRUEL}^{j-1}_J$, f is in CRUEL^{j-1}_J as well and $(C_{j-1}(f), \delta_{j-1}(f)) \in H_{j-1}$.

Let $(c, \delta) = (C_{j-1}(f), \delta_{j-1}(f))$, and assume that $c \geq r_j$. If $\delta + p_j \leq \Delta$ then no fault can occur in f on job j. Thus $(C_j(f), \delta_j(f)) = (c + p_j, \delta + p_j)$, consistent with the algorithm. Otherwise, we have $\delta + p_j > \Delta$. If f faults on job j, then, according to the definition of cruel sequences, this fault will occur exactly Δ time units after the previous one, so $(C_j(f), \delta_j(f)) = (c + 2p_j, \delta + 2p_j - \Delta)$. If we do not fault on j, then after j there will be no constraint on the position of the last fault, so $(C_j(f), \delta_j(f)) = (c + p_j, \Delta)$.

Finally, suppose that $c < r_j$. This means that in the greedy schedule j starts at its release time r_j. Note that then we can assume, without loss of generality, that either f is empty, or it only faults at r_j. For if f faults on j but not at its release time, we can remove from f all faults before r_j and move the fault to r_j, and this new fault sequence will j-dominate f. Therefore, if f faults at r_j, then the corresponding pair is $(r_j + 2p_j, 2p_j)$. Otherwise, the corresponding pair is $(r_j + p_j, \Delta)$.

Running time It may seem at first that the size of H_j could double at each step. However, since we eliminated dominated pairs, of all pairs of type (c, Δ) we added, only the one with the biggest c remains. Therefore, the size of H_j increases at most by 1 at each step. Consequently, $|H_j| \leq j$ for all j.

To make the construction of H_j run in linear time, we can keep two lists of the new pairs to be added, one for the pairs of type $(c + p_j, \delta + p_j)$ and one for the pairs of type $(c + 2p_j, \delta + 2p_j - \Delta)$, ordered by increasing c (and thus by decreasing δ). The final list H_j can be obtained by merging two sorted sequences, and adding the dominating pair of type $(\tilde{c}, \Delta)$ and the pair $(r_j + 2p_j, 2p_j)$, if any. Each set H_j can therefore be built from the previous one in time $O(j)$. Thus the overall running time of the algorithm is $O(n^2)$.

Theorem 11 *For hidden faults, Algorithm 3 computes in time $O(n^2)$ the maximum completion times when all faults are separated by gaps of length at least Δ (that is, for the fault model* GAP_Δ).

6.1 Experimental results

As we showed in the previous section, the algorithm for Δ-faults runs in time $O(n^2)$ in the worst case. Note, however, that the algorithm is not data-oblivious, namely, its running time depends on how the size of the sets H_j evolves over time. For the overall running time to be quadratic, the size of H_j would have to increase by 1 in many steps, which means that in many steps no elimination would occur—a scenario that seems very unlikely in random or non-adversarial data sequences. In this section, we confirm this intuition through some experimental studies. We performed three types of experiments, for various probability distributions. In the first one, we show that the expected running time grows linearly with n. Next, we confirm this further by showing that the total size (sum) of the sets H_j is linear in expectation. Finally, we show that, for the uniform distribution, with high probability the size of the sets H_j is bounded by a constant throughout the algorithm. (Indeed, we will prove this fact in the next section.) The experiments and the stochastic analysis are both performed without release times. However, we note that introducing release times can only increase the number of eliminations and thus improve the performance even further.

Running time We have tested the running time of the algorithm for uniform and normal random distributions of the job lengths, without release times. In both cases, the job lengths are drawn from the interval $(0, \Delta/2)$, where we arbitrarily choose $\Delta/2 = 10$. We tested several normal distributions, with different values of the mean and variance (discarding the values that did not fall between 0 and $\Delta/2$). The number n of jobs ranges from 1 to 20,000. In all cases, the simulations show that the running time increases linearly with the size of the instance—see Fig. 2.

Note, however, that the slope of the linear curve depends on the distribution. The intuition here is quite simple. For ex-

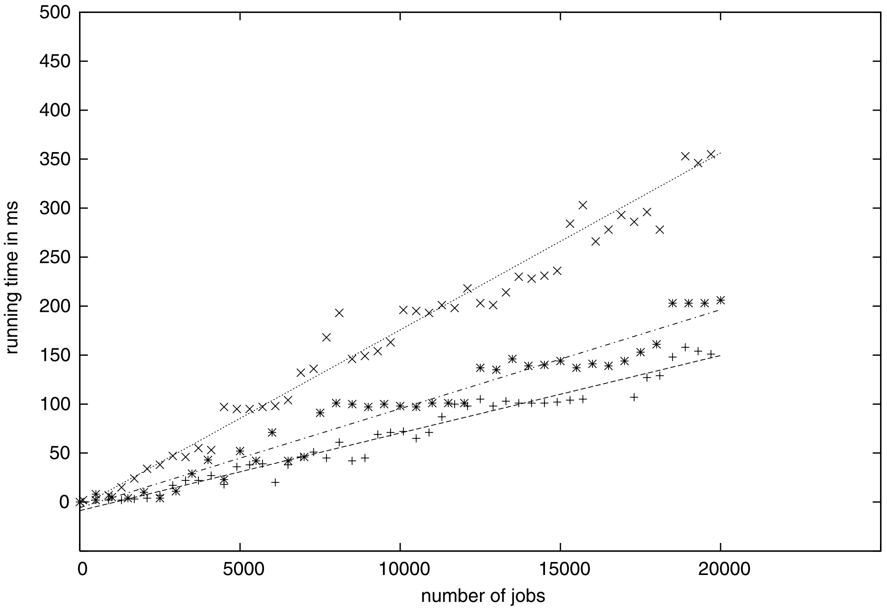


Fig. 2 Running time of Algorithm 3 in milliseconds. The x-axis represents the number n of jobs and the y-axis represents the running time. The results are shown for the uniform distribution (marked with "+"), and two normal distributions, one with mean 9 and variance 0.5 (marked with "$\times$") and one with mean 6 and variance 2 (marked with "$*$"). The *lines* show the corresponding linear interpolations

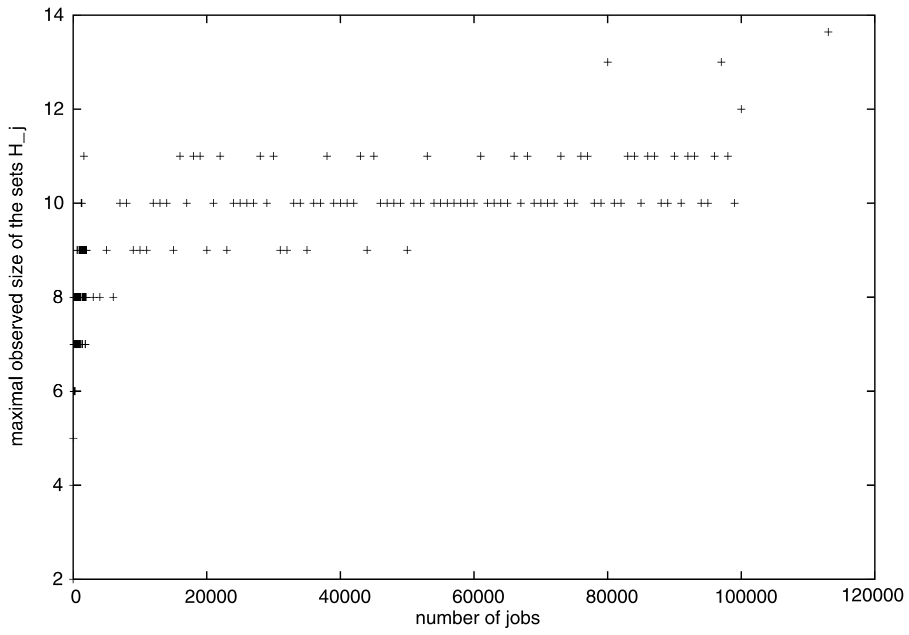


Fig. 3 Maximum size of the sets H_j. The x-axis represents the number n of jobs, and the y-axis represents the maximum cardinality of a set H_j

ample, for the normal distribution with mean 9 and variance 0.5, most of the generated job lengths are between 8 and $10 = \Delta/2$, decreasing the probability of elimination, and thus increasing the average number of pairs in the sets H_j.

The size of the sets H_j The running time of the algorithm is proportional to $\sum_{j=0}^{n} |H_j|$, the total number of pairs (c, δ) in the sets H_j. In the second batch of experiments we measured the expectation of the total size of sets H_j for instances of different size n ranging from 1 to 20,000. In our experiments, this value also grows linearly with n. (Results not shown.)

We have also run experiments where we computed the maximum size of the sets H_j, for various values of n, ranging from 0 to 120,000, and for the uniform distribution of job lengths. For each n, we run the simulation, and the value plotted for n is the maximum cardinality of sets $H_1, \ldots, H_n$ for the whole run. The results (see Fig. 3) show that this quantity grows very slowly, and appears to level off at around 11. Even for very large values of n, we did not find any sets H_j with more than 13 pairs. In the next section, we will prove that for the uniform distribution the expected size of the sets H_j is indeed $O(1)$.

6.2 Probabilistic analysis

In this section, we show that if the job lengths are drawn uniformly at random from the interval $(0, \Delta/2)$ then the expected running time of Algorithm 3 is $O(n)$. In fact, we prove something stronger—namely that the running time of the algorithm is $O(n)$ with very high probability.

Without loss of generality we can assume that $\Delta = 1$, and thus the job lengths are uniformly distributed in the interval $(0, 1/2)$.

Although, for the sake of simplicity, we carry out the calculations for the uniform distribution, without the release dates, our proof works for any distribution where each interval $[\frac{5}{26}, \frac{6}{26}]$ $(\frac{13}{52}, \frac{15}{52})$, $[\frac{15}{52}, \frac{17}{52}]$, and $(\frac{6}{13}, \frac{1}{2}]$ has strictly positive probability, and with release dates taken into account.

The idea of the proof is to show that, with high probability, the size of the sets H_j remains constant throughout the computation. To simplify the analysis, we only exploit certain types of elimination in the proof. As a result, the constant bound we get is higher than what one would expect based on the empirical study from the previous section.

Random sets Q_j For a given j, let $\omega_j = \min\{c \mid (c, \delta) \in H_j\}$. It is easy to show (by induction on j) that $H_j \subseteq [\omega_j, \omega_j + 1/2] \times [0, 1]$ and also $(\omega_j, 1) \in H_j$. The idea of the proof is to define a sequence of random sets Q_j which are essentially supersets of the sets H_j, offset leftwards by ω_j so that they are contained in the rectangle $[0, 1/2] \times [0, 1]$. This way, each step of the algorithm can be modeled as a mapping from $[0, 1/2] \times [0, 1]$ to $[0, 1/2] \times [0, 1]$. Another difference between the sets Q_j and H_j is that when computing Q_j we only do one type of elimination, and thus more points from Q_{j-1} may survive when mapped into Q_j than when H_j is computed from H_{j-1} in the actual algorithm. Nevertheless, we still show that with high probability the size of the Q_j remains constant.

We define first two auxiliary functions $F(\cdot)$ and $\hat{\alpha}(\cdot)$. For all $p \in (0, \frac{1}{2})$ and $\alpha, \beta \in [0, \frac{1}{2}] \times [0, 1]$, define

$$F(p, \alpha, \beta) = \begin{cases} (\alpha + p, \beta + 2p - 1) & \text{if } \beta \geq 1 - p, \\ (\alpha, \beta + p) & \text{if } \beta < 1 - p, \end{cases}$$

and for $Q \subseteq [0, \frac{1}{2}] \times [0, 1]$ and $p \in (0, \frac{1}{2})$, let

$$\hat{\alpha}(Q, p) = \max\{\alpha \mid (\alpha, \beta) \in Q \,\&\, \beta \geq 1 - p\}.$$

Intuitively, $F(p, \alpha, \beta)$ represents the mapping from H_{j-1} to H_j (except that all points are additionally offset leftwards by p), while $\hat{\alpha}(Q, p)$ represents the maximum α-coordinate of a point in Q to which the first option in the definition of $F(p, \alpha, \beta)$ applies. (See Fig. 4.)

In the rest of the proof we consider a random sequence $p_1, p_2, \ldots, p_n$ of job lengths, where each p_j is chosen uniformly from $(0, \frac{1}{2})$, and we prove that for this sequence the size of all sets H_j remains constant with high probability.

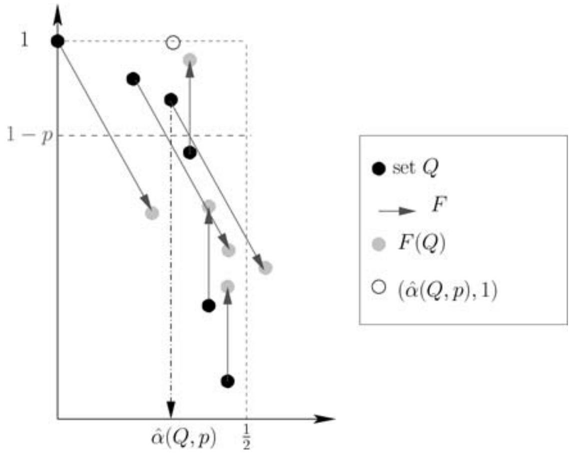


Fig. 4 Interpretation of $F(p, \alpha, \beta)$ and $\hat{\alpha}(Q, p)$

To avoid cumbersome notation, from now on we fix the values of $p_1, p_2, \ldots, p_n$. The sets Q_j are defined recursively. For $j = 0$, let $Q_0 = \{(0, 1)\}$. For $j \geq 1$, suppose that Q_{j-1} has been defined. To simplify notation, denote $F_j(\alpha, \beta) = F(p_j, \alpha, \beta)$ and $\hat{\alpha}_j = \hat{\alpha}(Q_{j-1}, p_j)$. As before, for $(\alpha, \beta), (\alpha', \beta') \in [0, \frac{1}{2}] \times [0, 1]$ we say that (α, β) *dominates* (α', β') if and only if $\alpha \geq \alpha'$ and $\beta \geq \beta'$. Then

$$Q'_j = \{(\alpha, \beta) \in F_j(Q_{j-1}) \mid (\alpha, \beta) \text{ is not dominated by } (\hat{\alpha}_j, 1) \text{ or } (p_j, 2p_j)\} \cup \{(\hat{\alpha}_j, 1)\},$$
$$Q_j = \{(\alpha - \hat{\alpha}_j, \beta) \mid (\alpha, \beta) \in Q'_j\}.$$

The reader needs to keep in mind that F_j, Q_j and $\hat{\alpha}_j$, as well as all other values dependent on the sequence $\{p_j\}$ are actually random variables. Observe that, by definition, the point $(\hat{\alpha}(Q, p), 1)$ dominates all points $(\alpha, \beta) \in Q$ with $\alpha \leq \hat{\alpha}(Q, p)$. Also, by induction, for all $j \geq 1$ we have $(0, 1) \in Q_{j-1}$, and therefore $(p_j, 2p_j) \in F_j(Q_{j-1})$.

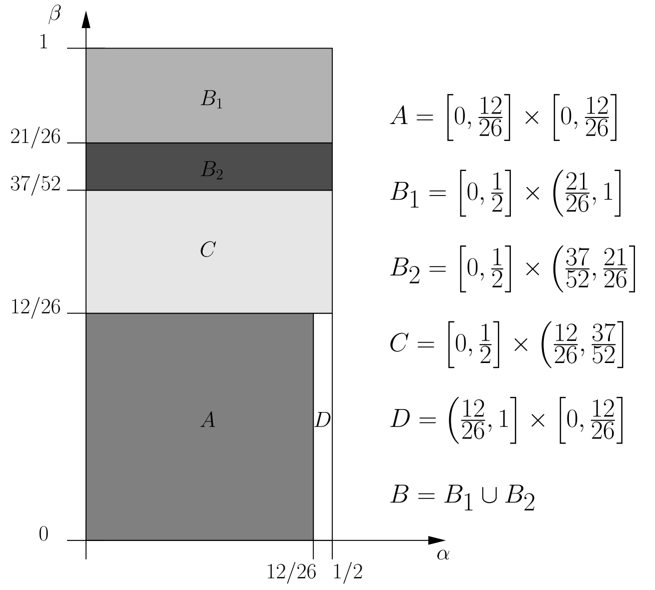
Lemma 12 *For all $j = 1, 2, \ldots, n$ and for any $(\alpha, \beta) \in F_j(Q_{j-1})$ we have $\alpha \leq \hat{\alpha}_j + \frac{1}{2}$.*

Proof Choose $(\alpha', \beta') \in Q_{j-1}$ such that $(\alpha, \beta) = F_j(\alpha', \beta')$. We have two cases. If $\beta' < 1 - p_j$, then $\alpha = \alpha' \leq \hat{\alpha}_j + \frac{1}{2}$, since both $\hat{\alpha}_j, \alpha' \in [0, \frac{1}{2}]$. If $\beta' \geq 1 - p_j$, then $\alpha = \alpha' + p_j \leq \hat{\alpha}_j + p_j \leq \hat{\alpha}_j + \frac{1}{2}$, by the definition of $\hat{\alpha}_j$. □

Observe that, according to Lemma 12 and from the definition of F_j, we have $Q_j \subseteq [0, \frac{1}{2}] \times [0, 1]$ for all j.

Lemma 13 $|H_j| \leq |Q_j|$ *for all $j = 1, 2, \ldots, n$.*

Proof For all j define $\hat{c}_j = \hat{\alpha}(H_j, p_j)$, that is, $\hat{c}_j$ is the analogue of the $\hat{\alpha}_j$ for sets H_j instead of Q_j. Let $\omega_j = \sum_{i=1}^{j-1} (\hat{\alpha}_i + p_i)$.



β
1
21/26
37/52
12/26
0
B_1
B_2
C
A
D
12/26
1/2
α
$A = \left[0, \frac{12}{26}\right] \times \left[0, \frac{12}{26}\right]$
$B_1 = \left[0, \frac{1}{2}\right] \times \left(\frac{21}{26}, 1\right]$
$B_2 = \left[0, \frac{1}{2}\right] \times \left(\frac{37}{52}, \frac{21}{26}\right]$
$C = \left[0, \frac{1}{2}\right] \times \left(\frac{12}{26}, \frac{37}{52}\right]$
$D = \left(\frac{12}{26}, 1\right] \times \left[0, \frac{12}{26}\right]$
$B = B_1 \cup B_2$

Fig. 5 Partition of $[0, \frac{1}{2}] \times [0, 1]$ into five zones

We want to prove that $(c, \delta) \in H_j$ implies $(c - \omega_j, \delta) \in Q_j$. The proof is by induction on j. For $j = 0$, $Q_0 = H_0 = \{(0, 1)\}$ and $\omega_0 = 0$. In the inductive step, suppose the property is true for $j - 1$, and let $(c, \delta) \in H_j$. We have two cases. If $\delta = 1$, (c, δ) originates from the point $(\hat{c}_{j-1}, \delta') \in H_{j-1}$. By induction, we have $(\hat{c}_{j-1} - \omega_{j-1}, \delta') \in Q_{j-1}$, and there is no point $(c', \beta') \in H_{j-1}$ with $c' \geq \hat{c}_{j-1}$ and $\beta' \geq 1 - p_j$ (because such a point would have been nondominated in H_{j-1} and thus would not have been eliminated in the process leading to creation of set H_{j-1}). Therefore, $(\hat{c}_{j-1} - \omega_{j-1}, \delta') = (\hat{\alpha}_{j-1}, \delta')$ and $(c - \omega_j, 1) = (0, 1) \in Q_j$.

If $\delta < 1$ then (c, δ) originated from a point $(c', \delta') \in H_{j-1}$, that is, either $(c, \delta) = (c' + p_j, \delta' + p_j)$ for $\delta' < 1 - p_j$ or $(c, \delta) = (c' + 2p_j, \delta' + 2p_j - 1)$ for $\delta' \geq 1 - p_j$. Then, by induction, $(c' - \omega_{j-1}, \delta') \in Q_{j-1}$. We only need to make sure that $F_j(c' - \omega_{j-1}, \delta')$ is not eliminated in Q_j. This cannot happen, for otherwise (c, δ) would have been eliminated in H_j as well: Indeed, the two points $(0, 1)$ in respectively Q_j and Q_{j-1} correspond in H_j and H_{j-1} to the points $(\omega_{j-1}, 1)$ and $(\omega_j, 1)$. □

We are now going to study how the cardinality of Q_j changes while j varies. We view these changes as a random process where at each step a point $(\alpha, \beta) \in Q_{j-1}$ is mapped into a point $(\alpha', \beta') = F_j(\alpha, \beta) - (\hat{\alpha}_j, 0)$. If $(\alpha', \beta') \notin Q_j$ (because $F_j(\alpha, \beta)$ is dominated by $(\hat{\alpha}_j, 1)$ or $(p_j, 2p_j)$), we say that (α, β) is *eliminated* in step j. Otherwise, we say that (α, β) *migrates* to (α', β'). For each $j = 1, 2, \ldots, n - 4$, and each point in Q_j, we show that in four steps with constant probability it is eliminated (more precisely, one of the subsequent points to which it migrates is eliminated).

We partition the rectangle $[0, \frac{1}{2}] \times [0, 1]$ into five zones A, B_1, B_2, C, D defined as in Fig. 5.

Lemma 14 *Fix some step j, $1 \leq j \leq m$. Then*: (a) *with probability at least $\frac{1}{13}$, all points in $Q_{j-1} \cap A$ will be eliminated*, (b1) *with probability at least $\frac{1}{13}$, all points in $Q_{j-1} \cap B_1$ that are not eliminated will migrate to $Q_j \cap A$*, (b2) *with probability at least $\frac{1}{13}$, all points in $Q_{j-1} \cap B_2$ that are not eliminated will migrate to $Q_j \cap A$*, (c) *with probability at least $\frac{1}{13}$, all points in $Q_{j-1} \cap C$ that are not eliminated will migrate to $Q_j \cap B$, and* (d) *with probability at least $\frac{1}{13}$, all points in $Q_{j-1} \cap D$ that are not eliminated will migrate to $Q_j \cap (B \cup C)$.*

Proof We prove each claim separately. Let $(\alpha, \beta) \in Q_{j-1}$ and $(\alpha', \beta') = F_j(\alpha, \beta) - (\hat{\alpha}, 0)$.

(a) Suppose that $p_j \in [\frac{12}{26}, \frac{1}{2}]$. If $(\alpha, \beta) \in Q_{j-1} \cap A$ then $F_j(\alpha, \beta)$ is dominated by $(p_j, 2p_j)$, so (α, β) will be eliminated. The probability that $p_j \in [\frac{12}{26}, \frac{1}{2}]$ is $\frac{1}{13}$.

Now, for the following cases, we assume that the point (α', β') is not eliminated.

(b1) Suppose that $p_j \in [\frac{5}{26}, \frac{12}{52}]$. If $(\alpha, \beta) \in Q_{j-1} \cap B_1$ then $\beta \geq 1 - p_j$, so $\alpha' = \alpha + p_j - \hat{\alpha}_j \leq p_j \leq \frac{12}{26}$ and $\beta' = \beta + 2p_j - 1 \leq \frac{12}{26}$ so $(\alpha', \beta') \in Q_j \cap A$. The probability that $p_j \in [\frac{5}{26}, \frac{12}{52}]$ is $\frac{1}{13}$.

(b2) Suppose that $p_j \in [\frac{15}{52}, \frac{17}{52}]$. If $(\alpha, \beta) \in Q_{j-1} \cap B_2$ then $\beta \geq 1 - p_j$, so $\alpha' = \alpha + p_j - \hat{\alpha}_j \leq p_j \leq \frac{12}{26}$ and $\beta' = \beta + 2p_j - 1 \leq \frac{21}{26} + 2 \cdot \frac{17}{52} - 1 = \frac{12}{26}$, so $(\alpha', \beta') \in Q_j \cap A$. The probability that $p_j \in [\frac{15}{52}, \frac{17}{52}]$ is $\frac{1}{13}$.

(c) Suppose that $p_j \in (\frac{13}{52}, \frac{15}{52})$. If $(\alpha, \beta) \in Q_{j-1} \cap C$ then $\beta < 1 - p_j$ so $\beta' = \beta + p_j > \frac{12}{26} + \frac{13}{52} = \frac{37}{52}$. So $(\alpha', \beta') \in Q_j \cap B$. The probability that $p_j \in [\frac{13}{52}, \frac{15}{52}]$ is $\frac{1}{13}$.

(d) Suppose that $p_j \in (\frac{12}{26}, \frac{1}{2}]$. If $(\alpha, \beta) \in Q_{j-1} \cap D$ $\beta < 1 - p_j$, so $\beta' = \beta + p_j > \frac{12}{26}$. Thus $(\alpha', \beta') \in Q_j \cap (B \cup C)$. The probability that $p_j \in (\frac{12}{26}, \frac{1}{2}]$ is $\frac{1}{13}$. □

Let $\lambda = \frac{1}{13^4}$. Looking at an individual point in Q_j, Lemma 14 implies that in fours steps $j + 1$, $j + 2$, $j + 3$, $j + 4$ (starting from Q_j and ending in Q_{j+4}) it is eliminated with probability at least λ. Consequently, each point on average contributes to at most $4/\lambda$ sets Q_j. Since one new point is introduced in each step, the expected total size of sets $Q_1, \ldots, Q_n$ is at most $4n/\lambda = O(n)$. By Lemma 13, this also bounds the running time of the algorithm, and we obtain the following theorem.

Theorem 15 *Suppose that the job lengths are drawn from a uniform distribution in $(0, \Delta/2)$. Then the expected running time of Algorithm 3 is $O(n)$.*

We now prove a stronger statement, namely that the size of each Q_j is small with high probability. Since the eliminations of different elements of Q_j are not independent, this needs some more work. In the following lemma, we show that with constant probability a constant fraction of points is eliminated. This is sufficient to calculate the desired bound.

Lemma 16 *Let $0 \le j \le n-4$. In four steps $j+1$, $j+2$, $j+3$, $j+4$ (starting from Q_j and ending in Q_{j+4}), with probability at least $\lambda = \frac{1}{13^4}$, at least one ninth of the points in Q_j will be eliminated.*

Proof If $|Q_j \cap A| \ge \frac{1}{9}|Q_j|$ then, according to Lemma 14(a), with probability at least $\frac{1}{13}$, all points in $Q_j \cap A$ will be eliminated in step $j+1$.

If $|Q_j \cap B| \ge \frac{2}{9}|Q_j|$ then $|Q_j \cap B_a| \ge \frac{1}{9}|Q_j|$ for $a = 1$ or $a = 2$. According to Lemma 14(b1) and (b2), with probability at least $\frac{1}{13}$ each point in B_a either will be eliminated in step $j+1$ or will migrate to A and then be eliminated in step $j+2$ with probability $\frac{1}{13}$. So with probability at least $\frac{1}{13^2}$, at least one ninth of points in Q_j will be eliminated in steps $j+1$ and $j+2$.

If $|Q_j \cap C| \ge \frac{2}{9}|Q_j|$ then, according to Lemma 14(c), with probability at least $\frac{1}{13}$ all points in $C \cap Q_j$ either will be eliminated or will migrate to B, and then, applying the argument from the previous case, with probability at least $\frac{1}{13^2}$ at least half of them will be eliminated in steps $j+2$ and $j+3$. Thus with probability at least $\frac{1}{13^3}$ at least one ninth of the points in Q_j will be eliminated in steps $j+1$, $j+2$ or $j+3$.

If none of the cases above holds then $|Q_j \cap D| \ge \frac{4}{9}|Q_j|$. Then, according to Lemma 14(d), with probability at least $\frac{1}{13}$, in step $j+1$ each point in $Q_j \cap D$ either will be eliminated or will migrate to B or to C. By applying the previous cases to either $Q_{j+1} \cap B$ or $Q_{j+1} \cap C$, whichever is bigger, we conclude that with probability at least $\frac{1}{13^4}$ at least one fourth of the points in $Q_j \cap D$ (and thus at least one ninth of the points in Q_j) will be eliminated in steps $j+1$, $j+2$, $j+3$, or $j+4$. □

Lemma 17 *Recall that $\lambda = \frac{1}{13^4}$. For $t = 0, 1, \dots, \lfloor n/4 \rfloor$, let $P_t(k) = \text{Prob}[|Q_{4t}| \ge k]$. Then $P_t(k) \le (1-\lambda/2)^{(k-k_0)/4}$ where $k_0 = -\frac{32}{\log(1-\lambda/2)} + 4$.*

Proof For $k \le k_0$ we have $(1-\lambda/2)^{(k-k_0)/4} \ge 1$, so the condition is trivially satisfied. Assume now that $k > k_0$. In this case, the proof is by induction on t. In the base case, for $t < k_0/4$, we have $k > 4t$ and $P_t(k) = 0$ (because $|Q_j| \le j$ for all j with probability 1), and the theorem holds.

In the inductive step, let $t \ge k_0/4$ and suppose the property is true for $t' = t-1$ and all values of k. By Lemma 16, in steps $4t-3$, $4t-2$, $4t-1$, and $4t$, with probability at least λ at least one ninth of points from $Q_{4(t-1)}$ have been eliminated. At the same time, at most four points have been added. Thus if $|Q_{4t}| \ge k$ then either we had $|Q_{4(t-1)}| \ge \frac{9}{8}(k-4)$, or $k-4 \le Q_{4(t-1)} < \frac{9}{8}(k-4)$ and fewer than one ninth of the points in $Q_{4(t-1)}$ were eliminated in steps $4t-3$, $4t-2$, $4t-1$, and $4t$—an event whose probability is at most $1-\lambda$. Therefore,

$$\begin{aligned}
P_t(k) &\le P_{t-1}\left(\frac{9}{8}(k-4)\right) \\
&\quad + (1-\lambda)\left[P_{t-1}(k-4) - P_{t-1}\left(\frac{9}{8}(k-4)\right)\right] \\
&= (1-\lambda)P_{t-1}(k-4) + \lambda P_{t-1}\left(\frac{9}{8}(k-4)\right) \\
&\le (1-\lambda)(1-\lambda/2)^{(k-4-k_0)/4} \\
&\quad + \lambda(1-\lambda/2)^{(\frac{9}{8}(k-4)-k_0)/4} \\
&= (1-\lambda/2)^{(k-4-k_0)/4}\big(1-\lambda+\lambda(1-\lambda/2)^{(k-4)/32}\big) \\
&\le (1-\lambda/2)^{(k-4-k_0)/4} \\
&\quad \times \left[1-\lambda+\lambda(1-\lambda/2)^{-1/\log(1-\lambda/2)}\right] \\
&\le (1-\lambda/2)^{(k-4-k_0)/4}[1-\lambda+\lambda/2] \\
&= (1-\lambda/2)^{(k-k_0)/4},
\end{aligned}$$

and the lemma follows. □

Theorem 18 *Suppose that the job lengths are drawn from a uniform distribution in $(0, \Delta/2)$. Then the size of each set H_j is constant with high probability. Specifically, for $j = 1, 2, \dots, n$, we have $\text{Prob}[|H_j| \ge k] \le C_1(C_2)^k$ where C_1 and C_2 are constants and $C_2 < 1$.*

Proof Fix j. By Lemma 13, we have $|H_j| \le |Q_j|$. Thus this claim follows from Lemma 17 with constants $C_1 = (1-\lambda/2)^{-k_0/4}$ and $C_2 = (1-\lambda/2)^{1/4}$. Since $\lambda > 0$, we have $C_2 < 1$. □

Other distributions Theorem 18 holds, in fact, for more general distributions on job lengths. To obtain an $O(n)$ bound, all we need is that all intervals discussed in the five parts of the proof of Lemma 14 are hit with non-zero probability. Changing these probabilities from $\frac{1}{13}$ to other positive values will only affect the constant in the big-O notation. Thus we have:

Theorem 19 *Suppose that the job lengths are drawn from a distribution in $(0, \Delta/2)$ in which each interval $[\frac{5}{26}, \frac{6}{26}]$, $(\frac{13}{52}, \frac{15}{52})$, $[\frac{15}{52}, \frac{17}{52}]$, and $(\frac{6}{13}, \frac{1}{2}]$ has strictly positive probability. Then the expected running time of Algorithm 3 is $O(n)$.*

6.3 An $\Omega(n^2)$ lower bound

Algorithm 3 gives rise to a data structure problem where we need to maintain a dynamic set H_j of pairs (c, δ) under a sequence of conditional offset operations. At each step j, to obtain H_{j+1} from H_j, we are given a threshold τ and two offset vectors (α,β), (α',β'), and we perform the following operation: For each $(c,\delta)\in H_j$ let

$$(c,\delta)\leftarrow\begin{cases}(c+\alpha,\delta+\beta) & \text{if } \delta\le\tau,\\ (c+\alpha',\delta+\beta') & \text{if } \delta>\tau.\end{cases}$$

(In Algorithm 3, we have $\tau=\Delta-p_j$, $(\alpha,\beta)=(p_j,p_j)$ and $(\alpha',\beta')=(2p_j,2p_j-\Delta)$. In fact, the algorithm creates other points as well, but we ignore them here, for simplicity.)

Is there a data structure to implement a sequence of m conditional offset operations so that the overall running time will be less than $O(mn)$? In this section, we consider a simple abstraction of this problem and show that its complexity in the model of algebraic computation trees is $\Omega(mn)$.

We stress that this lower bound does not imply a lower bound for the original problem of fault tolerant scheduling, but rather on a class of algorithms that use our dynamic approach and attempt to maintain the sets H_j using some data structures. It is conceivable (although, in our view, not likely) that a completely different approach may lead to a faster algorithm.

The 1-dimensional version We focus on a simplified problem, where we maintain a set of numbers (instead of pairs of numbers), and one of the offset values is 0. The input consists of three vectors of real numbers:

$$\bar{x}=(x_1,x_2,\ldots,x_n),\qquad \bar{\tau}=(\tau_1,\tau_2,\ldots,\tau_m),$$
$$\bar{\beta}=(\beta_1,\beta_2,\ldots,\beta_m),$$

where $\bar{x}$ represents the input values and $\bar{\tau}$, $\bar{\beta}$ represent m operations on $\bar{x}$. In the jth *conditional offset* operation, we do the following: For each $i=1,2,\ldots,n$, if $x_i\le\tau_j$ then we set $x_i\leftarrow x_i+\beta_j$ (otherwise x_i remains unchanged). The task is to compute the vector $\bar{y}$ resulting from applying these m conditional offset operations successively to $\bar{x}$.

Algebraic computation trees We now show that the above problem requires time $\Omega(mn)$ in the algebraic computation tree model. This computation model is an extension of the standard comparison tree model, where algebraic operations on the variables are allowed. The computation is represented by a tree that has two types of nodes: computation nodes and decision nodes. In a computation node (that has one child), an operation $o\in\{+,-,\times,/,\sqrt{}\}$ is applied to some variables. In a decision node, a comparison between two variables is made, and such a node has two children corresponding to the outcome (true or false). Leaves are labeled as either "accept" leaves or "reject" leaves. Each decision problem is modeled by a set $V\subseteq\mathbb{R}^d$. An algebraic computation tree solves the decision problem "given $\bar{v}\in\mathbb{R}^d$, is $\bar{v}\in V$?" if, for any given $\bar{v}$, the computation on $\bar{v}$ leads to an "accept" leaf if and only if $\bar{v}\in V$. The complexity is measured by the maximum tree depth. (See (Ben-Or 1983).)

Let $W\in\mathbb{R}^{2m+2n}$ be the set of vectors

$$\bar{v}=(x_1,\ldots,x_n,\tau_1,\ldots,\tau_m,\beta_1,\ldots,\beta_m,y_1,\ldots,y_n)$$

such that $\bar{y}=(y_1,y_2,\ldots,y_n)$ is the result of applying the m conditional offsets with parameters (τ_j,β_j), $j=1,\ldots,m$, to the input vector $\bar{x}=(x_1,x_2,\ldots,x_n)$. We consider a decision problem where, given a vector $\bar{v}\in\mathbb{R}^{2n+2m}$, we wish to determine if $\bar{v}\in W$, and we will show that any algorithm for this problem requires time $\Omega(mn)$ in the algebraic computation tree model. This implies that, given $\bar{x}$, $\bar{\tau}$, $\bar{\alpha}$, computing the output vector $\bar{y}$ requires time $\Omega(mn)$ as well.

For a set $W\subseteq\mathbb{R}^d$, let $\#(W)$ denote the number of connected components of W. By a well-known result of Ben-Or (1983), the algebraic computation tree complexity of W is at least $\log_6\max\{\#(W),\#(\mathbb{R}^{2m+2n}-W)\}-\Theta(m+n)$.

In fact, we will consider a fixed sequence of operations. We take $\bar{\tau}$, $\bar{\beta}$ where $\tau_j=\sum_{i=1}^{j}2^{m-i}$ and $\beta_j=2^{m-j}$ for $j=1,2,\ldots,m$. Let $W'\subseteq\mathbb{R}^{2n}$ be the set of vectors $(\bar{x},\bar{y})$ where $\bar{y}$ is obtained from $\bar{x}$ by applying the sequence of operations (τ_j,β_j), $i=1,2,\ldots,m$, defined above. Since $\#(W)\ge\#(W')$, to prove our lower bound it is sufficient to show the following inequality:

$$\#(W')\ge 2^{mn}. \tag{2}$$

We prove (2) by presenting a set of 2^{mn} points in W' that all must be in different connected components of W'. Define $K\subseteq W'$ to be the set of vectors $(\bar{k},\bar{l})\in W'$ where $\bar{k}\in\{0,1,\ldots,2^m-1\}^n$. Clearly, $|K|=2^{mn}$.

Lemma 20 *If $(\bar{k},\bar{l}),(\bar{k}',\bar{l}')\in K$ and $\bar{k}\ne\bar{k}'$ then $(\bar{k},\bar{l})$, $(\bar{k}',\bar{l}')$ are in different connected components of W'.*

Proof For integers $0\le k\le 2^m-1$ and $1\le i\le m$, let $k[i]$ denote the ith bit of k in its binary representation. Note that if $(\bar{k},\bar{l})\in K$ and $\bar{k}=(k_1,k_2,\ldots,k_n)$, $\bar{l}=(l_1,l_2,\ldots,l_n)$, then for each $i=1,2,\ldots,n$ we have $l_i=k_i+\sum_{j=1}^{m}(1-k_i[j])\beta_j$.

Choose an i such that $k_i\ne k'_i$ and let j the smallest integer such as $k_i[j]\ne k'_i[j]$. Without loss of generality, we can assume that $k_i[j]=0$ and $k'_i[j]=1$. Denote $A_j=\sum_{r=1}^{j-1}(1-k_i[r])2^{m-r}$, $R_j=2^{m-j}+\sum_{r=j+1}^{m}(1-k_i[r])2^{m-r}$, and $R'_j=\sum_{r=j+1}^{m}(1-k'_i[r])2^{m-r}$. Then we have:

$$l_i-k_i-A_j=R_j\ge 2^{m-j},$$
$$l'_i-k'_i-A_j=R'_j\le 2^{m-j}-1.$$

Define the function $f_{i,j}:W'\to\mathbb{R}$ so that $f_{i,j}(\bar{x},\bar{y})=y_i-x_i-A_j-2^{m-j}$. This is a continuous function, and it satisfies $f_{i,j}(\bar{k},\bar{l})\ge 0$ and $f_{i,j}(\bar{k}',\bar{l}')\le -1$. According to the intermediate value theorem of calculus, any connected path from $(\bar{k}, \bar{l})$ to $(\bar{k}', \bar{l}')$ must contain a point $(\bar{x}, \bar{y})$ for which $f_{i,j}(\bar{x}, \bar{y}) = -\frac{1}{2}$. But, by definition of W', if $(\bar{x}, \bar{y})$ were in W', then $y_i - x_i$ would be integer. Therefore, $(\bar{x}, \bar{y}) \notin W'$ and $(\bar{k}, \bar{l})$ and $(\bar{k}', \bar{l}')$ must be in different connected components of W', as claimed. □

Since $|K| = 2^{mn}$, Lemma 20 implies that $\#(W') \geq 2^{mn}$. Therefore, $\#(W) \geq 2^{mn}$ as well. By the lower bound of Ben-Or (Ben-Or 1983), the algebraic computation tree complexity of W is then at least $\Omega(\log \#(W)) = \Omega(mn)$. This, in turn, implies the following lower bound.

Theorem 21 *Any algorithm for maintaining a set of n numbers under a sequence of m conditional offset operation requires time $\Omega(mn)$ in the algebraic computation tree model.*

7 Final comments

We presented algorithms for testing fault tolerance of sequenced jobs in several fault models. For the model where the number of faults is bounded by a constant k we gave an $O(n)$-time fault-tolerance testing algorithm. For the model where the time between the faults is lower bounded by a constant Δ, our algorithms run in time $O(n)$ for exposed faults (detectable immediately) and in time $O(n^2)$ for hidden faults (detectable after the job completes.) We also show that this last algorithm runs in expected time $O(n)$ for a wide class of probability distributions on job lengths, and that the $O(n^2)$ worst-case running time cannot be improved in the algebraic computation tree model.

Our method can be extended to a yet more general model where two bounds k and Δ are given, and in any fault sequence there can be at most k gaps between faults of length greater than Δ. The dynamic programming approach from Algorithm 3 can be generalized to this model (and hidden faults) to yield an algorithm with worst case running-time $O(kn^2)$, although experiments show and actual running time of $O(kn)$, which can be explained by the same intuition.

For the future work, it would be of great interest to study the model where the fault sequences are drawn from a given probability distribution. Then the goal would be to compute (or estimate) the probability of failure for a given sequence of jobs.

References

Aydin, H. (2004). On fault-sensitive feasibility analysis of real-time task sets. In *Proceedings of the 25th IEEE international real-time systems symposium (RTSS'04)* (pp. 426–434). Washington: IEEE Computer Society.

Ben-Or, M. (1983). Lower bounds for algebraic computation trees. In *Proceedings of the 15th ACM symposium on theory of computing (STOC)* (pp. 80–86).

Egan, E., Kutz, D., Mikulin, D., Melhem, R., & Mossé, D. (1999). Fault-tolerant RT-Mach (FT-RT-Mach) and an application to real-time train control. *Software: Practice and Experience*, *29*, 379–395.

Girault, A., Kalla, H., & Sorel, Y. (2004). A scheduling heuristics for distributed real-time embedded systems tolerant to processor and communication media failures. *International Journal of Production Research*, *42*(14), 2877–2898.

Ghosh, S., Melhem, R., & Mossé, D. (1995). Enhancing real-time schedules to tolerate transient faults. In *Proceedings IEEE real-time systems symposium* (pp. 120–129).

Ghosh, S., Melhem, R., & Mossé, D. (1997). Fault-tolerance through scheduling of aperiodic tasks in hard-real time multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, *8*, 272–284.

Kalyanasundaram, B., & Pruhs, K. (1997). Fault-tolerant real-time scheduling. In *Proceedings of the 5th European symposium on algorithms (ESA)* (pp. 296–307).

Liberato, F., Lauzac, S., Melhem, R., & Mossé, D. (1999). Fault tolerant real-time global scheduling on multiprocessors. In *Proceedings of the Euromicro workshop in real-time systems*.

Liberato, F., Melhem, R., & Mosse, D. (2000). Tolerance to multiple transient faults for aperiodic tasks in hard real-time systems. *IEEE Transactions on Computers*, *49*, 906–914.

Manimaran, G., & Siva Ram Murthy, C. (1998). A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis. *IEEE Transactions on Parallel and Distributed Systems*, *9*(11), 1137–1152.

Mosse, D., Melhem, R., & Ghosh, S. (2003). A nonpreemptive real-time scheduler with recovery from transient faults and its implementation. *IEEE Transactions on Software Engineering*, *29*, 752–767.

Qi, X., Jiang, H., & Swanson, D. R. (2002). An efficient fault-tolerant scheduling algorithm for real-time tasks with precedence constraints in heterogenous systems. In *Proceedings of the 13th international conference on parallel processing* (pp. 360–368).

Qin, X., Han, Z., Jin, H., Pang, L., & Li, S. (2000). Realtime fault-tolerant scheduling in heterogeneous distributed systems. In *Proceedings of the international conference on parallel and distributed processing techniques and applications* (pp. 421–427).