Appeared in the Proceedings of 17th IEEE Real-Time Systems Symposium, Dec. 1996, Washington, DC

Bounding Completion Times of Jobs with Arbitrary Release Times and Variable Execution Times

Jun Sun
Geoworks, Inc.
2001 Center St, Berkeley, CA 94704
jsun@geoworks.com

Jane W.S. Liu
Department of Computer Science,
University of Illinois, Urbana, IL 61801
janeliu@cs.uiuc.edu

Abstract

In many real-time systems, the workload can be characterized as a set of jobs with linear precedence constraints among them. Jobs often have variable execution times and arbitrary release times. We describe here three algorithms that can be used to compute upper bounds on completion times of such jobs scheduled on a priority-driven basis. The algorithms have different performance and complexity. Simulation was performed to compare their performance.

1. Introduction

The workload of a real-time system oftentimes consists of dependent jobs. For example, an event of pushing a button may trigger several jobs to start execution in order. Meanwhile some other jobs may be executing in response to some other events. In order to guarantee the responsiveness of the system, we must employ a scheduling algorithm that leads to short completion times of these dependent jobs. Moreover, we must have some means to compute upper bounds of the completion times so that we can determine whether every job will indeed complete in time.

Specifically, we assume here that the workload on the processor consists of n chains of jobs, or *job chains*, $\mathbf{J}_1$, $\mathbf{J}_2$, $\ldots$, $\mathbf{J}_n$. We let $J_{i,j}$ denote the jth job on job chain $\mathbf{J}_i$; $J_{i,j}$ cannot execute until its immediate predecessor $J_{i,j-1}$ completes. The execution time $e_{i,j}$ of each job $J_{i,j}$ is in the range $[e^-_{i,j}, e^+_{i,j}]$ where $e^-_{i,j} \leq e^+_{i,j}$. The release time of job $J_{i,j}$ is $r_{i,j}$, which is arbitrary but fixed.

We describe here three algorithms that let us compute upper bounds on the completion times of jobs when they are scheduled according to a given priority-driven algorithm. The first one computes upper bounds on the *effective response times (ERT)* of individual jobs in each chain and is, therefore, called Algorithm ERT. The second algorithm is based on the analysis of the *critical job (CJ)*, a notion which we will define later; it is called Algorithm CJA. The third algorithm iteratively applies Algorithm CJA to yield tighter bounds and is called Algorithm ITR. Algorithms ERT, CJA, ITR yield increasingly tighter upper bounds, but run at increasingly higher complexity. While Algorithm ERT can be used for on-line admission control (i.e., to determine whether the system can accept a new chain of jobs and still ensures on-time completion of all jobs), Algorithm ITR is suitable for off-line schedulability analysis.

A great deal of work has been done on timing analysis for periodic tasks [1, 2, 3, 4, 5]. A periodic task is an infinite stream of identical jobs that are released periodically. The objective of timing analysis is to bound the response times of all jobs in each task. Lehoczky etc [3] proposed a time-demand analysis method for this purpose. Harbour etc. [5] proposed a method to bound the response times of jobs where each periodic task is a chain of subtasks. These existing methods either cannot be applied to bound the completion times of dependent jobs that have arbitrary release times (e.g., the methods based on schedulable utilization bounds [1]) or yield unsatisfactorily loose bounds (e.g., the time-demand analysis method [3]). A reason for the poor performance of existing methods is that they ignore the exact release times of jobs but work with the worst-case combination of release times. While this treatment makes sense in timing analysis for periodic tasks, it causes the algorithms to be very pessimistic for dependent jobs. As a matter of fact, Algorithm ERT proposed in this paper also ignores the release times of jobs. As we will see in Section 6 that the performance of this algorithm is poor compared with the other two algorithms, which make use of the information on job release times.

The problem solved by our algorithm is also related to the *validation problem*, that verifies if all timing constraints are satisfied in a real-time system. Both problems deal with a set of jobs with variable execution times. Ha [6] has studied the validation problem in multiprocessor or distributed systems. In her work, a system is *predictable* if the completion time of a job can be bounded by the completion times of the job in the *maximum schedule* and *minimum schedule*, where the maximum(minimum) schedule is obtained by applying the given priority-driven algorithm to the given set of jobs assuming that all jobs have their maximum(minimum) execution times. As shown by an example in the next section, the execution of a set of dependent, preemptable jobs on a single processor is not predictable. Bounding the completion times of jobs is a reasonable approach to validating the timing constraints for this kind of systems. Our algorithms provide tighter bounds and, thus, more accurate conclusions on the satisfiability of timing constraints than the general bounds provided by algorithms in [6].

The rest of the paper is organized as follows. Section 2 formally defines the problem addressed and introduces the notations used in the paper. Section 3, 4, and 5 present Algorithm ERT, CJA and ITR, respectively. A simulation was performed to compare the performance of these three algorithms, and the simulation results are presented in Section 6. Section 7 discusses modifications of the algorithms when jobs have jittered release times, followed by the conclusions of the paper.

2. Problem Formulation

Again, the problem addressed here is to determine whether every job in n independent job chains can complete in time when the jobs are scheduled on a processor according to a given priority-driven algorithm. By independent chains, we mean that $J_{i,1}$ has no predecessor for every $i = 1, 2, \ldots, n$, and there is no precedence constraint between any pair of jobs in two different chains. We assume that each job $J_{i,j}$ has a fixed priority $\phi_{i,j}$ and is preemptable. As stated earlier, the release time $r_{i,j}$ of job $J_{i,j}$ is arbitrary but fixed. The execution time is in the range $[e^-_{i,j}, e^+_{i,j}]$. Both the maximum execution time $e^+_{i,j}$ and the minimum execution time $e^-_{i,j}$, as well as the release time $r_{i,j}$, of $J_{i,j}$ are known, but the actual execution time $e_{i,j}$ is not known.

We assume that the release time $r_{i,j}$ of every job $J_{i,j}$ is consistent with its precedence constraints. Specifically, the release time $r_{i,j}$ of a job $J_{i,j}$ is no sooner than $r_{i,j-1} + e^-_{i,j-1}$. In other words, the release time of every job $J_{i,j}$ is no sooner than the earliest time at which its immediate predecessor can complete.[1]

An example of such a system is shown in Figure 1. In this example, referred to as Example 1 later, there are two job chains, $\mathbf{J}_1$ and $\mathbf{J}_2$. $\mathbf{J}_1$ has four jobs, and $\mathbf{J}_2$ has two. Each job $J_{i,j}$ is described by a triplet, $(r_{i,j}, \phi_{i,j}, [e^-_{i,j}, e^+_{i,j}])$. We

[1] When the given release times do not satisfy this assumption, we replace them with *effective release times* that do. The effective release time of $J_{i,1}$ is equal to its given release time. The effective release time of $J_{i,j}$ is equal to its given release time or the sum of the effective release time of $J_{i,j-1}$ and $e^-_{i,j-1}$, whichever is larger. We loose no generality by working with the effective release times of jobs.

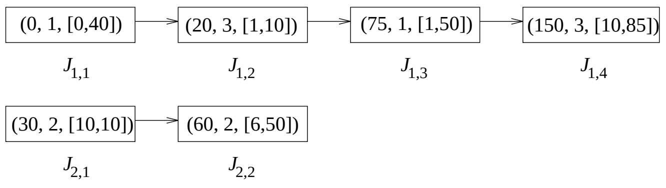

Figure 1. The Simple Job Set in Example 1

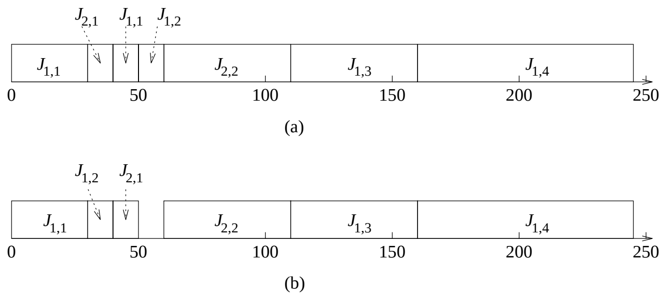

Figure 2. Two Schedules of the System in Example 1

use integers to represent priorities; the greater the integer, the higher the priority.

A job is *ready* at the instant when it is released or when its immediate predecessor completes, whichever is later. Let $y_{i,j}$ denote the *ready time* of job $J_{i,j}$ and $c_{i,j}$ denote the completion time of $J_{i,j}$. Since the first job in a job chain J_i has no predecessors, it is ready when it is released, i.e., $y_{i,1} = r_{i,1}$. For a later job $J_{i,j}$ $(j > 1)$, we have $y_{i,j} = \max\{r_{i,j}, c_{i,j-1}\}$. The *response time* of $J_{i,j}$ is equal to its completion time less its release time (i.e., the duration of interval $(r_{i,j}, c_{i,j}]$), and the *effective response time* is equal to its completion time less its ready time (i.e., the duration of interval $(y_{i,j}, c_{i,j}]$).

Because the execution times of jobs may vary and the scheduling algorithm is priority driven, there may be many different schedules for a given set of jobs. According to some of these schedules, a job may have its worst-case (i.e., the latest) completion time while other jobs may not. In particular, when all jobs have their maximum execution times, we may not observe the worst-case completion times of all the jobs. For example, Figure 2 shows two schedules of the two job chains in Example 1. In both cases, jobs are scheduled according to their assigned priorities. We have the schedule in Figure 2(a) when all the jobs have their maximum execution times. According to this schedule, job $J_{2,1}$ completes at time 40. However when the execution time of $J_{1,1}$ is reduced from 40 to 30 time units, we obtain the schedule in Figure 2(b). According to this schedule, the completion time of $J_{2,1}$ is 50. As it turns out, this is the worst-case completion time of $J_{2,1}$. This example shows that systems considered in this paper are in general not predictable [6]. To bound the completion times of jobs, we can of course exhaustively simulate the execution of the system and search for the worst-case completion times of jobs. The complexity of a brutal-force search is $O(E^N)$, where E is the length of the range $[e^-_{i,j}, e^+_{i,j}]$ for all i and j and N is the total number of jobs in the system, making this approach impractical for most real-life systems. We focus here on analytical methods which give us upper bounds on the completion times of jobs rather than finding the exact worst case.

3. Algorithm ERT

Algorithm ERT first bounds the effective response times of jobs and then derives the bounds on completion times from the effective response times. To motivate this algorithm, we focus on a job $J_{i,j}$ in job chain $\mathbf{J}_i$. Obviously, the jobs that can execute during the interval $(y_{i,j}, c_{i,j}]$ must be in different job chains from $\mathbf{J}_i$. Furthermore, their priorities must be higher than or equal to the priority $\phi_{i,j}$ of $J_{i,j}$.

Figure 3 illustrates a job chain $\mathbf{J}_k$ ($k \neq i$). The shaded boxes represent the jobs in $\mathbf{J}_k$ whose priorities are lower than $\phi_{i,j}$, and white boxes represent jobs whose priorities are equal to or higher than $\phi_{i,j}$. The lower priority jobs divide the chain $\mathbf{J}_k$ into subchains, each of which contains only jobs with priorities higher than or equal to $\phi_{i,j}$. In this example, there are three such equal or higher priority subchains. We call such a subchain an *interference block* of $J_{i,j}$. In general, an interference block of $J_{i,j}$ is a subchain $\{J_{k,l}, J_{k,l+1}, \ldots, J_{k,l+u}\}$ of $\mathbf{J}_k$, for some $k \neq i$. Priorities $\phi_{k,l}, \phi_{k,l+1}, \ldots,$ and $\phi_{k,l+u}$ are higher than or equal to $\phi_{i,j}$; either $J_{k,l}$ has no predecessor or $\phi_{k,l-1}$ is lower than $\phi_{i,j}$; and either $J_{k,l+u}$ has no successor or $\phi_{k,l+u+1}$ is lower than $\phi_{i,j}$. Only jobs from the interference blocks of $J_{i,j}$ can execute during the interval $(y_{i,j}, c_{i,j}]$. Furthermore, since the interference blocks are separated by one or more jobs with priorities lower than $J_{i,j}$, it is impossible for jobs in more than one interference block of the same chain to execute in $(y_{i,j}, c_{i,j}]$. Consequently, when we want to bound the effective response time of $J_{i,j}$, we only need to consider one interference block from each job chain $\mathbf{J}_k$ ($k \neq i$) that can execute in the interval $(y_{i,j}, c_{i,j}]$. This allows us to bound

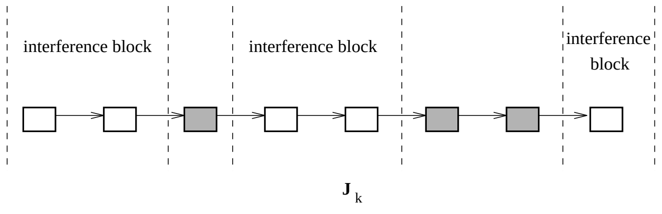

Figure 3. Interference Blocks

```
Algorithm Interference
Input :
    1. A job set $\mathbf{J}$ of jobs where each job $J_{k,l}$ has release
       time $r_{k,l}$, priority $\phi_{k,l}$, and execution time in the
       range $[e^-_{k,l}, e^+_{k,l}]$.
    2. The target job $J_{i,j}$.
Output : $inter(J_{i,j}, \mathbf{J})$, the maximum delay suffered by $J_{i,j}$
    due to executions of other jobs in interval $(y_{i,j}, c_{i,j}]$.
Algorithm :
    1. $inter = 0$
    2. For every job chain $\mathbf{J}_k$ ($k \neq i$)
        (a) Identify the interference blocks in $\mathbf{J}_k$ by comparing
            the priorities of jobs in it with $\phi_{i,j}$;
        (b) Compute $M_{k,l}$, the sum of the maximum execution times of
            jobs in the $l$th interference block in $\mathbf{J}_k$.
        (c) $inter = inter + \max_{1 \leq l \leq m_k}\{M_{k,l}\}$.
    3. return $inter$.
```

Figure 4. The Pseudo-Code of Algorithm Interference

the total possible execution times of jobs that can delay the completion of $J_{i,j}$ once it becomes ready at $y_{i,j}$.

We focus now on finding the maximal effective response time of $J_{i,j}$. Hereafter, we call the job whose completion time we are trying to bound the *target job*. By an interference block, we mean specifically an interference block of the target job. Suppose that a job chain $\mathbf{J}_k$ has m_k interference blocks, and $M_{k,l}$ is equal to the sum of the maximum execution times of jobs in the lth interference block in $\mathbf{J}_k$. As we have discussed in the previous paragraph, the maximum delay of the target job $J_{i,j}$ by $\mathbf{J}_k$ (i.e., the maximum amount of time for which $J_{i,j}$ can be delayed by jobs in $\mathbf{J}_k$) is never more than the maximum of $M_{k,l}$ for all $l = 1, 2, \ldots, m_k$ (i.e., $\max_{1 \leq l \leq m_k}\{M_{k,l}\}$). The sum of the maximum delays of $J_{i,j}$ by all the job chains other than $\mathbf{J}_i$ gives the maximal total execution time of all jobs other than $J_{i,j}$ that can execute in interval $(y_{i,j}, c_{i,j}]$. Algorithm Interference, whose pseudo-code is listed in Figure 4, makes use of this fact. It computes $inter(J_{i,j}, \mathbf{J})$, the maximum total delay which target job $J_{i,j}$ may suffer.

Given $inter(J_{i,j}, \mathbf{J})$, the duration of interval $(y_{i,j}, c_{i,j}]$, which is the effective response time of $J_{i,j}$, can be straightforwardly bounded. In particular, the duration of the interval $(y_{i,j}, c_{i,j}]$ satisfies the inequality, $(c_{i,j} - y_{i,j}) \leq e^+_{i,j} +$

$inter(J_{i,j}, \mathbf{J})$. A simple transformation gives

$$c_{i,j} \leq y_{i,j} + e^+_{i,j} + inter(J_{i,j}, \mathbf{J}) \quad (1)$$

For the first job $J_{i,1}$ in the job chain $\mathbf{J}_i$, $r_{i,1} = y_{i,1}$. An upper bound $\hat{c}_{i,1}$ of $c_{i,1}$ is given by

$$\hat{c}_{i,1} = r_{i,1} + e^+_{i,1} + inter(J_{i,1}, \mathbf{J}) \quad (2)$$

For a job $J_{i,j}$ $(j > 1)$ which is not the first job in the chain, its ready time $y_{i,j}$ is equal to $\max\{c_{i,j-1}, r_{i,j}\}$. Therefore an upper bound $\hat{c}_{i,j}$ of $c_{i,j}$ is

$$\hat{c}_{i,j} = \max\{\hat{c}_{i,j-1}, r_{i,j}\} + e^+_{i,j} + inter(J_{i,j}, \mathbf{J}) \quad (3)$$

By applying Eqs.(2) and (3) to jobs in their execution precedence order, we can obtain an bound on the completion time for every job. Algorithm ERT calculates $\hat{c}_{i,j}$ for each target job $J_{i,j}$ in this manner. The complexity of Algorithm Interference is $O(N)$, and the complexity of Algorithm ERT is $O(N^2)$, where N is the total number of jobs in the system.

As an example, we compute $inter(J_{i,j}, \mathbf{J})$ for every job in Example 1. From the perspective of $J_{1,1}$, both jobs in $\mathbf{J}_2$ have higher priorities, and they form one interference block. The maximal execution time of this block is the sum of the maximum execution times of $J_{2,1}$ and $J_{2,2}$, which is 60 time units. This is the value of $inter(J_{1,1}, \mathbf{J})$. From the perspective of job $J_{2,1}$, jobs $J_{1,1}$ and $J_{1,3}$ have lower priorities lower, and jobs $J_{1,2}$ and $J_{1,4}$ have higher priorities. Since $J_{1,2}$ and $J_{1,4}$ are separated by $J_{1,3}$, they form two different interference blocks. The maximal execution times of these two interference blocks are 10 time units and 85 time units, respectively. Consequently, $inter(J_{2,1}, \mathbf{J})$ is 85 time units. Similarly, $inter(J_{i,j}, \mathbf{J})$ of other jobs in Example 1 can be computed, and they are listed in Table 1. Based on these bounds on maximal delays each job can suffer, we proceed to apply Algorithm ERT to obtain bounds on the completion times of jobs.

$$\begin{aligned}
\hat{c}_{1,1} &= r_{1,1} + e^+_{1,1} + inter(J_{1,1}, \mathbf{J}) = 100 \\
\hat{c}_{1,2} &= \max\{\hat{c}_{1,1}, r_{1,2}\} + e^+_{1,2} + inter(J_{1,2}, \mathbf{J}) = 110 \\
\hat{c}_{1,3} &= \max\{\hat{c}_{1,2}, r_{1,3}\} + e^+_{1,3} + inter(J_{1,3}, \mathbf{J}) = 220 \\
\hat{c}_{1,4} &= \max\{\hat{c}_{1,3}, r_{1,4}\} + e^+_{1,4} + inter(J_{1,4}, \mathbf{J}) = 305 \\
\hat{c}_{2,1} &= r_{2,1} + e^+_{2,1} + inter(J_{2,1}, \mathbf{J}) = 125 \\
\hat{c}_{2,2} &= \max\{\hat{c}_{2,1}, r_{2,2}\} + e^+_{2,2} + inter(J_{2,2}, \mathbf{J}) = 260
\end{aligned}$$

$J_{i,j}$	$J_{1,1}$	$J_{1,2}$	$J_{1,3}$	$J_{1,4}$	$J_{2,1}$	$J_{2,2}$
$inter(J_{i,j}, \mathbf{J})$	60	0	60	0	85	85

Table 1. $inter(J_{i,j}, \mathbf{J})$ values for jobs in Example 1

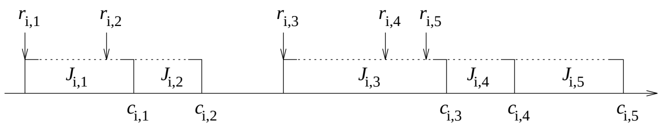

Figure 5. Illustration of the Critical Job on a Schedule

4. Algorithm CJA

Algorithm ERT is simple to understand, easy to implement, and has relatively low complexity. However, it often does not give satisfactory bounds. In the case of job $J_{2,2}$ in Example 1, the bound on its completion time is 260 time units. 125 time units of this bound is contributed by the completion time of its immediate predecessor $J_{2,1}$, and 85 units by the possible interference from $J_{1,4}$. However, its predecessor $J_{2,1}$ completes at time 125 only if $J_{2,1}$ has been delayed by $J_{1,4}$ for 85 time units. Yet this amount of delay from $J_{1,4}$ is counted again in the effective response time of $J_{2,2}$. As a result, the same 85 time units is counted twice in computing $\hat{c}_{2,2}$. Similarly, the interference from $J_{2,1}$ and $J_{2,2}$ is counted twice in the upper bound $\hat{c}_{1,3}$ of the completion time of $J_{1,3}$. Algorithm CJA overcomes this problem by considering the job chain or a subchain that contains the target job as a whole rather than dealing with the target job in isolation.

To motivate Algorithm CJA, suppose that a job chain $\mathbf{J}_i$ has five jobs, and we are interested in bounding the completion time of the target job $J_{i,5}$. Figure 5 shows a possible *worst-case schedule* for $J_{i,5}$, i.e., a schedule according to which $J_{i,5}$ has its worst-case completion time. Obviously, since all the release times are known, the completion time $c_{i,5}$ is equal to $r_{i,k}$ plus the duration of $(r_{i,k}, c_{i,5}]$ for $k = 1, 2, \ldots, 5$. If we can find a tighter bound on the duration of $(r_{i,k}, c_{i,5}]$, we can find a tighter bound on the completion time of $J_{i,5}$. As we will see shortly, for this purpose, we should examine the *critical job* $J_{i,c(j)}$ of each target job $J_{i,j}$. In a schedule, $J_{i,c(j)}$ is the last job in $\mathbf{J}_i$ before and including $J_{i,j}$ whose ready time is equal to its release time. We call the interval $(r_{i,c(j)}, c_{i,j}]$ the *critical interval*. For example, in Figure 5, $J_{i,3}$ is the critical job of $J_{i,5}$, and interval $(r_{i,3}, c_{i,5}]$ is the critical interval.

Lemma 1 states a fact that helps us to bound the duration of the critical interval $(r_{i,c(j)}, c_{i,j}]$.

Lemma 1 *All jobs that are in job chains other than $\mathbf{J}_i$ and execute in the critical interval $(r_{i,c(j)}, c_{i,j}]$ have priorities higher than or equal to $J_{i,low}$, where job $J_{i,low}$ $(c(j) \leq low \leq j)$ is the lowest priority job among jobs $J_{i,c(j)}$, $J_{i,c(j)+1}$, $\ldots$, $J_{i,j}$.*

Proof : Because each of the jobs $J_{i,c(j)}$, $J_{i,c(j)+1}$, $\ldots$, and $J_{i,j}$ is ready to execute immediately after its immediate predecessor completes, any job $J_{k,l}$ that is in another chain and executes in the critical interval $(r_{i,c(j)}, c_{i,j}]$ must execute ahead of one of these jobs. Hence the priority of $J_{k,l}$ is higher than or equal to the priority of this job, which is in turn higher than or equal to $J_{i,low}$. □

According to Lemma 1, the jobs that are in job chains other than $\mathbf{J}_i$ and can execute in the critical interval $(r_{i,c(j)}, c_{i,j}]$ are the same set of jobs that are in job chains other than $\mathbf{J}_i$ and can execute in the interval $(y_{i,low}, c_{i,low}]$. Consequently, their total execution time can be bounded by $inter(J_{i,low}, \mathbf{J})$. The duration of the critical interval is never larger than this amount plus the maximum execution times of $J_{i,c(j)}$, $J_{i,c(j)+1}$, $\ldots$, and $J_{i,j}$. In other words,

$$c_{i,j} \leq r_{i,c(j)} + \sum_{l=c(j)}^{j} e^+_{i,l} + inter(J_{i,low}, \mathbf{J})$$

In the above critical job analysis, we assume that we know the critical job $J_{i,c(j)}$ of each target job $J_{i,j}$ in the worst-case schedule. This assumption in general is not true. To get around this problem, Algorithm CJA computes a bound on the completion time of $J_{i,j}$ by assuming that each of its predecessors, including $J_{i,j}$ itself, is the critical job. Since in the worst-case schedule there must exist a critical job, one of the bounds thus computed must be a correct one, and the maximum of these bounds must be a correct bound as well. The pseudo-code of Algorithm CJA is listed in Figure 6. Its complexity is $O(N^3)$. We again use $\hat{c}_{i,j}$ to denote the upper bound on the completion time of $J_{i,j}$.

For example, we apply Algorithm CJA to bound the completion time of $J_{1,3}$ in Example 1 and obtain the following results.

1. Let $J_{1,1}$ be the critical job. Job $J_{1,1}$ has the lowest priority among $J_{1,1}$, $J_{1,2}$ and $J_{1,3}$, and $b_{1,1} = r_{1,1} + \sum_{k=1}^{3} e^+_{1,k} + inter(J_{1,1}, \mathbf{J}) = 160$.

2. Let $J_{1,2}$ be the critical job. Job $J_{1,3}$ has the lowest priority among $J_{1,2}$ and $J_{1,3}$, and $b_{1,2} = r_{1,2} + \sum_{k=2}^{3} e^+_{1,k} + inter(J_{1,3}, \mathbf{J}) = 140$.

3. Let $J_{1,3}$ be the critical job. Job $J_{1,3}$ has the lowest priority among $J_{1,3}$ itself, and $b_{1,3} = r_{1,3} + \sum_{k=3}^{3} e^+_{1,k} + inter(J_{1,3}, \mathbf{J}) = 185$.

4. The final bound $\hat{c}_{1,3}$ is equal to $\max\{b_{1,1}, b_{1,2}, b_{1,3}\} = 185$.

We note that when computing the bounds $b_{1,1}$, $b_{1,2}$ and $b_{1,3}$, and hence the final bound, the interference from every job on $\mathbf{J}_2$ is counted only once. As a result, the bound obtained by Algorithm CJA for $J_{1,3}$ is tighter than that obtained by Algorithm ERT. For the same reason, the bounds obtained by Algorithm CJA for $J_{1,4}$ and $J_{2,2}$ are tighter than those computed by Algorithm ERT. Table 2 lists the bounds on the completion times computed by Algorithm CJA for all the jobs in Example 1. For the sake of comparison, the bounds computed by Algorithm ERT are also listed in the table.

```
Algorithm CJA
Input :
        A set {J} of jobs where each job $J_{i,j}$ of which
        has the release time $r_{i,j}$, the priority $\phi_{i,j}$, and
        the execution time in the range of $[e^-_{i,j}, e^+_{i,j}]$.
Output :  The bound $\hat{c}_{i,j}$ on the completion time of each job $J_{i,j}$.
Algorithm :
        For each job $J_{i,j}$
          1. For each possible critical job $J_{i,k}$ $(1 \leq k \leq j)$
             (a) Find the job $J_{i,low}$ such that $J_{i,low}$
                 has the lowest priority among job
                 $J_{i,k}$, $J_{i,k+1}$, ..., and $J_{i,j}$.
             (b) Compute
                 $b_{i,k} = r_{i,k} + \sum_{l=k}^{j} e^+_{i,l} + inter(J_{i,low}, \mathbf{J})$
          2. Let $\hat{c}_{i,j} = \max_{1 \leq k \leq j}\{b_{i,k}\}$.
```

Figure 6. The Pseudo-Code of Algorithm CJA

algorithm	$J_{1,1}$	$J_{1,2}$	$J_{1,3}$	$J_{1,4}$	$J_{2,1}$	$J_{2,2}$
ERT	100	110	220	305	125	260
CJA	100	110	185	270	125	195

Table 2. Bounds Computed by Algorithm ERT and CJA

As a matter of fact, every bound obtained by Algorithm CJA is always tighter than the corresponding one obtained by Algorithm ERT. Suppose $\hat{c}_{i,j}$ is a bound computed by Algorithm ERT on the completion time of $J_{i,j}$. From (3), we can deduce that

$$\hat{c}_{i,j} \geq \hat{c}_{i,j-1} + e^+_{i,j} + inter(J_{i,j}, \mathbf{J})$$

and

$$\hat{c}_{i,j} \geq r_{i,j} + e^{+}_{i,j} + inter(J_{i,j}, \mathbf{J})$$

Given any k $(1 \leq k \leq j)$, we expand $\hat{c}_{i,j}$ recursively by using the above two inequalities and obtain

$$\hat{c}_{i,j} \geq r_{i,k} + \sum_{l=k}^{j} e^{+}_{i,l} + \sum_{l=k}^{j} inter(J_{i,l}, \mathbf{J}) \tag{4}$$

On the other hand, Step 1(b) of Algorithm CJA in Figure 6 states

$$b_{i,k} = r_{i,k} + \sum_{l=k}^{j} e^{+}_{i,l} + inter(J_{i,low}, \mathbf{J}) \tag{5}$$

where $1 \leq k \leq j$ and $k \leq low \leq j$. Obviously, from (4) and (5), the bound computed by Algorithm ERT is greater than or equal to every $b_{i,k}$ computed by Algorithm CJA and hence is greater than or equal to the maximum of $b_{i,k}$'s, which is the final bound computed by Algorithm CJA.

5. Algorithm ITR

Both of the previous two algorithms use Algorithm Interference in a straightforward way to bound the maximal total execution time of jobs in job chains other than $\mathbf{J}_i$ that can execute in some interval. An obvious drawback of this approach is that the release times of jobs are not taken into account. For example, when the maximum delay suffered by $J_{1,1}$ in Example 1 is computed by Algorithm Interference, the execution time of $J_{2,2}$ is counted in the delay. However, we notice that $J_{2,2}$ will not be released until time 60, by which time $J_{1,1}$ should have already completed even when $J_{1,1}$ has its maximum execution time and is preempted by $J_{2,1}$. Hence one possible improvement is to leave out from consideration the jobs (such as $J_{2,2}$ in this example) that cannot possibly interfere with the execution of the target job. In other words, in Step 1(b) of Algorithm CJA, if we can prune some jobs from the job set $\mathbf{J}$ that cannot possibly execute in the critical interval and apply Algorithm interference to the pruned job set, we can obtain a tighter bound on the maximum possible delay job $J_{i,j}$ might suffer. Clearly, the pruning process must be done for every pair of a target job and an assumed critical job because different jobs may be pruned for different combinations.

The next question is how to obtain the information we need to prune jobs properly. One approach is called the *pessimistic iteration*. We use Algorithm CJA to obtain an initial bound on the completion time of every job. This is called the initial step. We then iteratively apply the modified Algorithm CJA to obtain a new bound on the completion time of each job. When bounding the duration of the critical interval for each pair of target job $J_{i,j}$ and assumed critical job $J_{i,c(j)}$, the modified Algorithm CJA excludes from consideration any job $J_{k,l}$ whose interval $(r_{k,l}, \hat{c}_{k,l}]$ does not overlap with the interval $(r_{i,c(j)}, \hat{c}_{i,j}]$, where $\hat{c}_{k,l}$ and $\hat{c}_{i,j}$ are the bounds on the completion times of $J_{k,l}$ and $J_{i,j}$, respectively, obtained in the initial step or the previous iteration step. This pruning process is safe because $\hat{c}_{i,j}$'s computed in the initial step and each of the previous iteration step are correct upper bounds on the completion times of jobs and, hence, all the pruned jobs have no chance to execute in the critical interval $(r_{i,c(j)}, c_{i,j}]$. The iteration will terminate when all the new bounds obtained in the current step are equal to the corresponding bounds obtained in the previous step. Obviously, during each iteration before the termination, at least one bound on the completion time of a job is strictly smaller than its corresponding previous one. Since bounds cannot be arbitrarily small, the iteration will terminate in a finite number of steps.

Although the pessimistic iteration approach improves the bounds in general, it does not help in our example. We notice that in Example 1 job $J_{1,1}$ is both the target job and the critical job. The initial bound on the completion time of $J_{1,1}$ is 90. Based on this bound, interval $(r_{1,1}, \hat{c}_{1,1}]$ overlaps with the critical interval $(r_{2,2}, \hat{c}_{2,2}]$ of job $J_{2,2}$. Consequently $J_{2,2}$ will not be pruned from consideration using the pessimistic iteration approach, and the bound on the completion time of $J_{1,1}$ will not be improved.

A more aggressive approach is called the *optimistic iteration*. Contrary to the pessimistic iteration, the optimistic iteration starts as an initial step with an optimistic bounds on the completion times of jobs, obtained by assuming that each job is interfered only by jobs in the same job chain. During each subsequent iteration step, we use the modified CJA algorithm to obtain a new bound on the completion time of each job $J_{i,j}$ based on bounds obtained in either the previous iteration step or the initial step. Like the pessimistic iteration, for each pair of critical job $J_{i,c(j)}$ and the target job $J_{i,j}$, we prune any job $J_{k,l}$ whose interval $(r_{k,l}, \hat{c}_{k,l}]$ does not overlap with the interval $(r_{i,c(j)}, \hat{c}_{i,j}]$. The iteration will terminate when all the new bounds are equal to the corresponding bounds obtained in the earlier step.

Figure 7 lists the pseudo-code of Algorithm ITR, which uses the optimistic iteration approach. It is essentially a loop which is preceded by an initial step. Inside the loop, Algorithm CJA is applied but is preceded by two extra steps. Step 2(biA) and Step 2(biB) are inserted to prune the jobs $J_{k,l}$ whose intervals $(r_{k,l}, \hat{c}_{k,l}]$ do not overlap with the interval $(r_{i,c(j)}, \hat{c}_{i,j}]$. Because of the extra pruning steps, the bounds obtained at the end of the loop body are always no larger than the corresponding bounds obtained by Algorithm CJA without any pruning. So are the final bounds when the iteration terminates.

The correctness of Algorithm ITR is stated formally by the following two theorems. Their proofs can be found in the appendix.

Theorem 1 *Algorithm ITR terminates after a finite number of iterations.*

Theorem 2 *The bounds obtained in the last iteration of Algorithm ITR are the correct upper bounds on the completion times of jobs.*

```
Algorithm ITR
Input :
        A set $\{\mathbf{J}\}$ of jobs where each job $J_{i,j}$ has the
        release time $r_{i,j}$, the priority $\phi_{i,j}$, and the exe-
        cution time in the range of $[e^-_{i,j}, e^+_{i,j}]$.

Output :  A bound $\hat{c}_{i,j}$ on the completion time of each job $J_{i,j}$.

Algorithm :
    1. For each job $J_{i,j}$
        (a) if $j = 1$, $\hat{c}_{i,j} = r_{i,j} + e^+_{i,j}$;
        (b) otherwise, $\hat{c}_{i,j} = \max\{\hat{c}_{i,j-1}, r_{i,j}\} + e^+_{i,j}$.
        (c) $\hat{c'}_{i,j} = 0$. ($\hat{c'}_{i,j}$ is the bound on the completion
            time of $J_{i,j}$ computed in the previous iteration
            step.)
    2. Repeat until ($\hat{c}_{i,j} = \hat{c'}_{i,j}$) for every job $J_{i,j}$
        (a) For each job $J_{i,j}$,
                $\hat{c'}_{i,j} = \hat{c}_{i,j}$
        (b) For each target job $J_{i,j}$
            i. For each possible critical job $J_{i,k}$ $(1 \le k \le j)$
                A. $\mathbf{J}' = \mathbf{J}$
                B. Purge from $\mathbf{J}'$ any job $J_{u,v}$ $(u \ne i)$ for
                   which the interval $(r_{u,v}, \hat{c'}_{u,v}]$ does not
                   overlap with the interval $(r_{i,k}, \hat{c'}_{i,j}]$.
                C. Find the job $J_{i,low}$ such that $J_{i,low}$
                   has the lowest priority among job $J_{i,k}$,
                   $J_{i,k+1}, \ldots, J_{i,j}$.
                D. Compute the bound $b_{i,k}$ by
                   $b_{i,k} = r_{i,k} + \sum_{l=k}^{j} e^+_{i,l} + inter(J_{i,low}, \mathbf{J}')$
            ii. $\hat{c}_{i,j} = \max\{b_{i,k}\}, k = 1, 2, \ldots, j$.
```

Figure 7. The Pseudo-Code of Algorithm ITR

From the proof of Theorem 1, we can see that there can be no more than $O(N^3)$ number of iteration steps. Since each iteration step has the same complexity of Algorithm CJA, which is $O(N^3)$, the complexity of Algorithm ITR is thus $O(N^6)$.

As an example, we apply Algorithm ITR to bound the completion time of $J_{1,1}$ in Example 1. The initial optimistic bound is equal to 40, which is equal to the release time of $J_{1,1}$ plus its maximum execution time. During the first iteration, the interval $(r_{2,1}, \hat{c}_{2,1}]$ overlaps with $(r_{1,1}, \hat{c}_{1,1}]$, and therefore $J_{2,1}$ is retained in $\mathbf{J}'$ at Step 2(biB). The interval $(r_{2,2}, \hat{c}_{2,2}]$, however, does not overlap with the interval $(r_{1,1}, \hat{c}_{1,1}]$. Therefore $J_{2,2}$ is pruned at Step 2(biB). The new bound on the completion time of $J_{1,1}$ becomes 50. During the second and later iterations, the interval $(r_{2,2}, \hat{c}_{2,2}]$ still does not overlap with the interval $(r_{1,1}, \hat{c}_{1,1}]$, and job $J_{2,2}$ is always pruned. Consequently the bound on the completion time of $J_{1,1}$ remains 50.

The final bounds on the completion times of all jobs in Example 1 obtained by (optimistic) Algorithm ITR are listed in Table 3. We also list the bounds obtained by Algorithm ERT and Algorithm CJA, as well as the actual worst-case completion times of jobs. We note that although Algorithm ITR gives fairly tight bounds compared with other two algorithms, it may still fail to find the actual worst-case completion times. Take job $J_{1,3}$ for example. Although Algorithm ITR has correctly determined that the completion of $J_{1,3}$ is delayed only by $J_{2,2}$, it fails to see, however, that the maximal amount of time delayed by $J_{2,2}$ is less than the maximum execution time of $J_{2,2}$, because $J_{2,2}$ is released 15 time units earlier than $J_{1,3}$.

algorithm	$J_{1,1}$	$J_{1,2}$	$J_{1,3}$	$J_{1,4}$	$J_{2,1}$	$J_{2,2}$
ERT	100	110	220	305	125	260
CJA	100	110	185	270	125	195
ITR	50	60	175	260	50	110
worst-case	50	60	160	245	50	110

Table 3. Bounds Computed by the Three Algorithms and the Actual Worst-Case Completion Times

6. Performance of the Algorithms

From the previous discussion, we know that bounds yielded by Algorithm ITR are tighter than those yielded by Algorithm CJA, which in turn are tighter than those yielded by Algorithm ERT, but we do not know by how much. To quantify their relative merits and to determine the way their relative performance depends on the characteristics of jobs, we perform a series of simulation experiments. This section discusses the criterion used to evaluate their performance, the method used to generate the workload, and finally the simulation results.

6.1. Performance Criterion

The performance criterion we use to compare two algorithms, say A and B, is the *bound ratio* or the *average bound ratio* of A over B. The ratios are defined as follows. For a given system of jobs, the bound ratio of (algorithm) A over (algorithm) B for a job is the ratio of the upper bound on the response time of the job obtained by A over the corresponding bound obtained by B. The bound ratio of the system is the average of the bound ratios for all the jobs in the system. In our experiment, we generate many synthetic systems and compute the bound ratio for each system. The average bound ratio is the average of the bound ratios of all the systems with the same characteristics examined in the experiment. Obviously the smaller the average bound ratio of A over B, the better algorithm A is compared with algorithm B, provided that the ratio is less than 1.

6.2. The generation of workload

Through preliminary experiment, we found that performance of the algorithms depends almost entirely on three factors. They are the number of job chains in the system, the number of jobs in each job chain and the *density of the schedule*, or the *schedule density*. Intuitively, the density of a schedule indicates how "sparse" the schedule is. It can be quantized by the *density factor*, which is the total maximum execution time of all jobs divided by the range of release times of jobs. For example, if the release times of jobs are distributed in the range of $[1, 1000]$ and the total maximum execution time of all jobs is equal to 1500, then the density factor is equal to 1.5. Obviously, the smaller the density factor, the "sparser" the schedule.

A *configuration* is a unique combination of values of the above three factors. Synthetic systems have the same configuration when they have the same number of job chains, number of jobs in each job chain and schedule density. In our simulation experiment, we examined configurations with the number of job chains ranging from 5, 10, or 15, the number of jobs in each job chain being 1, 2, 5, or 10, and the schedule density being 0.5, 1, or 2. We thus have 36 configurations. For each configuration, we generated 1000 systems to yield negligibly small confidence intervals for all the average values presented below.

Each system of a configuration with x job chains, y jobs per chain and schedule density z, is generated as follows. For each of the x job chains and each of the y jobs in the chain, we choose the release time of the job from the uniform distribution in the range [1,1000000]. We then sort the jobs in each job chain in the increasing order of their release times and add a precedence constraint to each pair of adjacent jobs in the job chain.

To choose the execution times of the jobs, we first compute the total maximum execution time of all jobs by multiplying the schedule density z by the range of job release times, 1000000. We then randomly divide the total maximum execution time among the xy jobs. This is done by first generating an *execution factor* for each job, which is uniformally distributed in range $[0.001, 1]$. We obtain the normalized execution factor for each job by dividing the execution factor over the sum of execution factors of all jobs. The maximum execution time of each job is then equal to the normalized execution factor times the total maximum execution time. We let the minimum execution time of every job to be 0. Finally, the priority of every job is randomly distributed in range $[1, 1000]$.

6.3. Comparison of Algorithm ERT and Algorithm CJA

In this subsection, by "bound ratio" we mean the average bound ratio of Algorithm CJA over Algorithm ERT. The simulation results show that bound ratios are not sensitive to the number of job chains in the system. For this reason, we only present in Figure 8 the bound ratio as a function of the number of jobs in each chain and the schedule density. Each value in the figure is the average value of the bound ratios of all systems of a configuration. The overall average bound ratio for all configurations is 0.77, which indicates that on average the bounds on the job response times yielded by Algorithm CJA are 23% shorter than the bounds yielded by Algorithm ERT.

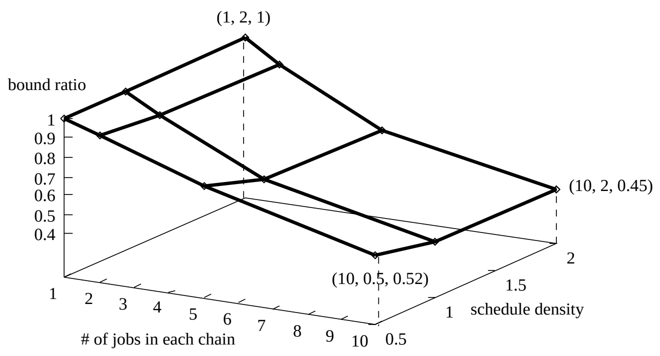

Figure 8. Bound Ratio of Algorithm CJA over Algorithm ERT

In the figure, we notice that the average bound ratio decreases as the number of jobs in each job chain increases. A closer examination reveals that this is because the bound ratio for an individual job is strongly correlated with the position of the job in the job chain. Figure 9 depicts the average bound ratio of jobs as a function of their position in the job chains. The value on the horizontal-axis is the position of jobs in their corresponding job chains, and the corresponding value on the vertical-axis is the average bound ratio for all jobs in that position. We notice that for the first job on every job chain, Algorithm CJA and Algorithm ERT yield the same bound. The is due to the fact that both algorithms in fact do the same computation for these jobs. The average bound ratio decreases as the number of predecessors of the target job increases, largely due to the fact that Algorithm ERT sometimes counts the interference of jobs multiple times. The later a job is in a job chain, the more likely Algorithm ERT does so. As a result, a system with longer job chains has a smaller average bound ratio. Figure 8 shows that the bound ratio also decreases a little as the schedule density increases, mainly for a similar reason.

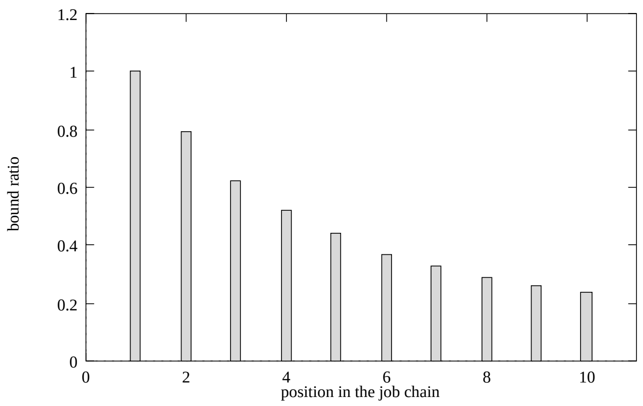

Figure 9. Bound Ratio as Function of the Position of Jobs

6.4. Comparison of Algorithm CJA and Algorithm ITR

In this subsection we focus on Algorithm CJA and ITR, and by "bound ratio" we mean the average bound ratio of Algorithm ITR over Algorithm CJA. The overall average bound ratio is 0.49 for all configurations, which indicates that on average the bounds on job response times computed by Algorithm ITR are about half the bounds computed by Algorithm CJA.

Figure 10 depicts the average bound ratio as a function of number of job chains in the system and shows that bound ratio of Algorithm ITR over Algorithm CJA varies slightly but noticeably with the number of job chains in the system. When the number of job chains increases while the other two parameters remain constant, the delay due to interference from jobs in difference chains increases. Due to the pruning step, Algorithm ITR can better estimates the effect of the increase than Algorithm CJA. As a result, Algorithm ITR obtains tighter bounds as the number of job chains in a system increases.

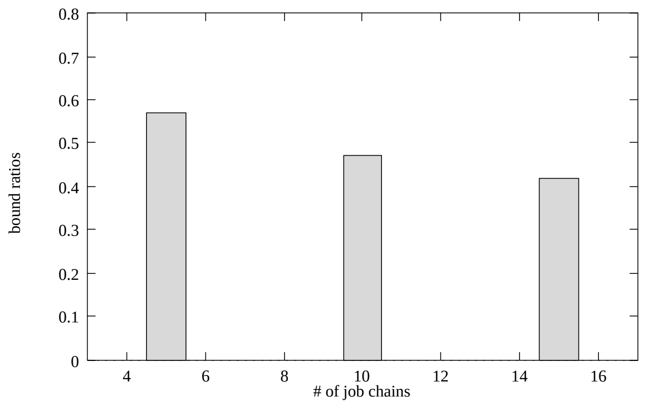

Figure 10. Bound Ratio as a Function of the Number of Job Chains in the System

Figure 11 shows the average bound ratio as a function of number of jobs in each job chain and the schedule density. The bound ratios are much smaller when the schedule densities are smaller, indicating that Algorithm ITR is much more effective for sparse schedules. When the schedule is sparse, many jobs execute in isolation and do not interfere each other. The pruning step in Algorithm ITR can correctly detect this and obtain tighter bounds, while Algorithm CJA does not have this capability.

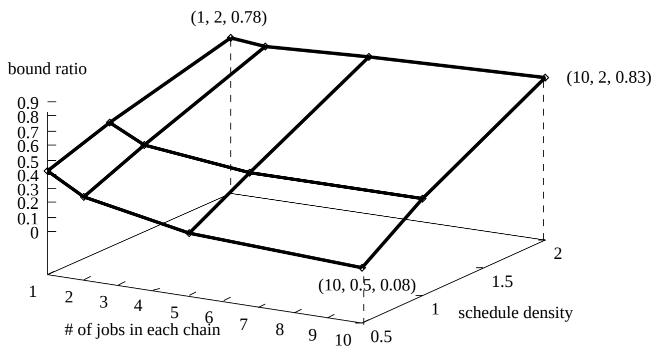

Figure 11. Bound Ratio of Algorithm ITR over Algorithm CJA

6.5. Summary of the Simulation Results

Figure 12 shows the bound ratio of Algorithm ITR over Algorithm ERT, as a function of the number of jobs in each job chain and the schedule density. As we expect, the bounds yielded by Algorithm ITR are much tighter than those by Algorithm ERT. In summary, we see a great reduction of the upper bounds on job response times by Algorithm ITR over Algorithm CJA and ERT. Furthermore, Algorithm

ITR is more effective when the schedule is sparse and the number of jobs in the system is large. When the schedule is "dense", the performance of Algorithm CJA is close to that of Algorithm ITR.

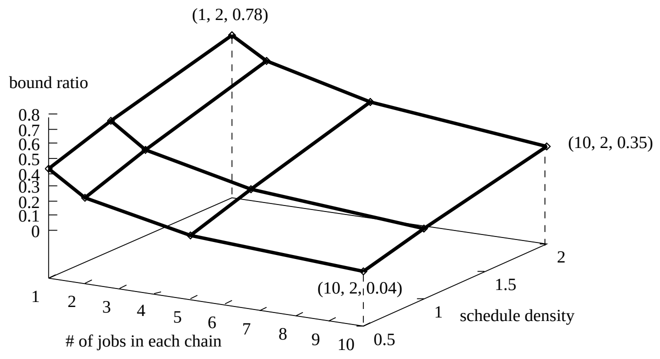

Figure 12. Bound Ratio of Algorithm ITR over Algorithm ERT

7. Conclusion and Extension

We have described three algorithms to bound the completion times of jobs in a set of chains when jobs have variable execution times, arbitrary release times and fixed priorities. The algorithms have different complexities and yield different performance. Our simulation results show that Algorithm ITR consistently produces tighter bounds than the other two algorithms. The example presented here suggests that the bounds obtained by Algorithm ITR are close the actual worst-case completion times. The complexity of Algorithm ITR is $O(N^6)$, where N is the total number of jobs in a system. This complexity is not a problem for off-line analysis. When this complexity is too high, e.g., for the purpose of on-line admission control, Algorithm CJA is a good alternative choice, especially when the schedule is expected to be "dense".

All three algorithms can be modified to deal with jittery release times, i.e., the release time of each job is in a range of $[r^-_{i,j}, r^+_{i,j}]$, Algorithm Interference will not be affected by release time jitters, because it does not use the information on job release times. In the case of Algorithm ERT, we simply replace all $r_{i,j}$'s with corresponding $r^+_{i,j}$'s. The bounds computed by the modified Algorithm ERT are correct.

In the case of Algorithm CJA, we first need to redefine the critical job of the target job as follows: For a target job $J_{i,j}$, the critical job $J_{i,c(j)}$ is the last job in $\mathbf{J_i}$ before and including $J_{i,j}$ whose ready time is in its release time range $[r^-_{i,c(j)}, r^+_{i,c(j)}]$. By the new definition, the critical job analysis remains correct in bounding the duration of interval $(r^+_{i,c(j)}, c_{i,j}]$. Consequently, the pseudo-code of Algorithm CJA remains correct if we replace $r_{i,k}$ with $r^+_{i,k}$ at Step 1(b) in Figure 6.

To see how to take into account release time jitters in the case of Algorithm ITR, we note that at the initial steps, Step 1(a) and 1(b) in Figure 7, we should replace the release time $r_{i,j}$ with the latest possible release time $r^+_{i,j}$. As a consequence, the initial bounds are conservative. At Step 2(biB), we need to prune non-interfering jobs to the target job $J_{i,j}$ from $\mathbf{J}'$. Due to the release jitters, the interval of a job $(J_{u,v})$ becomes $(r^-_{u,v}, \hat{c'}_{u,v}]$. Similarly, the critical interval between the critical job $(J_{i,k})$ and the target job $(J_{i,j})$ becomes $(r^-_{i,k}, \hat{c'}_{i,j}]$. We can thus test if a job $J_{u,v}$ is interfering based on whether these two intervals overlap. Lastly, when we compute each individual $b_{i,k}$ for each assumed critical job $J_{i,k}$ at Step 2(biD), we simply replace $r_{i,k}$ with $r^+_{i,k}$, and the final bound will be correct.

A problem related to this work is to find the exact worst-case completion time. Specifically, the pruning technique used in Algorithm ITR can effectively reduce a large number of combinations when exhaustive searching is used to find the worst-case completion time.

References

[1] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[2] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, December 1989.

[3] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *11th IEEE Real-Time Systems Symposium*, pages 201–209, December 1990.

[4] N. Audsley, A. Burns, K. Tindell, M. Richardson, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[5] M. G. Harbour, M. H. Klein, and J. P. Lehoczky. Timing analysis for fixed-priority scheduling of hard real-time systems. *IEEE Transactions on Software Engineering*, 20(1):13–28, January 1994.

[6] R. Ha. *Validating Timing Constraints in Multiprocessor and Distributed Systems*. PhD thesis, University of Illinois, Urbana-Champaign, Department of Computer Science, 1995.

Appendix A - Proof of Theorem 1

The bound of the completion time of each job obtained in the first iteration is no smaller than the corresponding bound obtained in the initial step, due to the optimistic estimation used in the initial step. As a consequence, for each pair of target job and critical job considered at Step 2(bi), fewer or the same number of jobs are pruned at Step 2(biB) in the second iteration step, making the value of $b_{i,k}$ greater than the corresponding one obtained in the first iteration step. Hence every bound obtained in the second iteration step is no smaller than the corresponding bound obtained in the first iteration step. It is easy to see that in each iteration step the bounds are monotonically non-decreasing and the total number of jobs pruned at Step 2(biB) is monotonically decreasing.

The iteration will continue if and only if for at least one job the new bound is greater than the corresponding bound obtained in the previous step. For a new bound of a job to be greater than its previous bound, fewer jobs must have been pruned from set $\mathbf{J}'$ at Step 2(biB) in the current iteration step than in the previous step. Overall, in each iteration step, the total number of jobs pruned at Step 2(biB) must decrease at least by one. Since the total number of jobs that can be pruned at Step 2(biB) cannot exceed $N^2(N-1)$ in the first iteration step, the iterative procedure will terminate in a finite number of iterations. □

Appendix B - Proof of Theorem 2

We will prove this theorem by an induction over the jobs in the increasing order of their release times. By convention, let $\hat{c}_{i,j}$ denote the bound on the completion time of $J_{i,j}$ obtained by Algorithm ITR, i.e., the bound obtained by the last two iteration steps, and let $c_{i,j}$ denote the actual completion time of $J_{i,j}$ in the schedule.

Induction basis : Because the release times of all jobs are consistent with their precedence constraints, the job with the earliest release time among all jobs must be $J_{i,1}$ for some i. Suppose that the actual completion time $c_{i,1}$ of $J_{i,1}$ is greater than the bound $\hat{c}_{i,1}$. Let Γ denote the total amount of execution times of all jobs, excluding $J_{i,1}$, that execute in interval $(r_{i,1}, \hat{c}_{i,1}]$. We must have $\Gamma + e_{i,1} > \hat{c}_{i,1} - r_{i,1}$ (Otherwise job $J_{i,1}$ would have completed by time $\hat{c}_{i,1}$). Hence $\Gamma + e^+_{i,1} > \hat{c}_{i,1} - r_{i,1}$.

Now let us focus on Steps 2(bi) during the last (outermost) iteration step in Algorithm ITR, specifically when $J_{i,1}$ is the target job and the critical job. In the last iteration step, for every job $J_{x,y}$, the bound $\hat{c}_{x,y}$ obtained is the same as the bound obtained in the second last iteration step, which is copied to $\hat{c'}_{x,y}$. Consequently the jobs pruned at Step 2(biB) are those whose release times are later than $\hat{c}_{i,1}$. Job set $\mathbf{J}'$ obtained at this step thus gives all the jobs that can execute in interval $(r_{i,1}, \hat{c}_{i,1}]$.

Obviously, every job that executes in interval $(r_{i,1}, \hat{c}_{i,1}]$ must have priority higher than or equal to $J_{i,1}$. Thus $Inter(J_{i,1}, \mathbf{J}')$ gives an upper bound on Γ. By Step 2(biD) and 2(bii), we have

$$\Gamma + e^+_{i,1} \leq Inter(J_{i,1}, \mathbf{J}') + e^+_{i,1} = b_{i,1} - r_{i,1} = \hat{c}_{i,1} - r_{i,1}$$

This is a contradiction to the conclusion stated in the previous paragraph. Therefore the hypothesis must be wrong, and for job $J_{i,1}$ Algorithm ITR yields a correct upper bound on its completion time.

Induction : Now we let $J_{i,j}$ be the job whose release time is later than the release times of k other jobs. As an induction hypothesis, we assume that the completion time of every job released before $J_{i,j}$ is no larger than the upper bound on its completion time obtained by Algorithm ITR. We will now prove that $c_{i,j}$ is no larger than $\hat{c}_{i,j}$ either.

We, again, prove this by contradiction. Suppose that $c_{i,j}$ is larger than $\hat{c}_{i,j}$. Let $J_{i,c}$ $(1 \leq c \leq j)$ be the critical job for $J_{i,j}$ in this schedule, and Γ be the total amount of execution times of all jobs from chains other than $\mathbf{J}_i$ that execute in interval $(r_{i,c}, \hat{c}_{i,j}]$. Since job $J_{i,j}$ is not completed by $\hat{c}_{i,j}$, we must have

$$\Gamma + \sum_{l=c}^{j} e^+_{i,l} \geq \Gamma + \sum_{l=c}^{j} e_{i,l} > \hat{c}_{i,j} - r_{i,c}$$

Now let us focus on Steps 2(bi) during the last (outermost) iteration step in Algorithm ITR, specifically when $J_{i,j}$ is the target job and $J_{i,c}$ is the critical job. If job $J_{u,v}$ is pruned at Step 2(biB), then either (1) $\hat{c}_{u,v} < r_{i,c}$, or (2) $r_{u,v} > \hat{c}_{i,j}$. If a job $J_{u,v}$ is pruned due to the first reason, its release time must be earlier than that of $J_{i,j}$. By induction hypothesis, the bound $\hat{c}_{u,v}$ is a correct upper bound on the completion time. Hence we are certain that job $J_{u,v}$ cannot execute in interval $(r_{i,c}, \hat{c}_{i,j}]$. On the other hand, if $J_{u,v}$ is pruned due to the second reason, it cannot execute in interval $(r_{i,c}, \hat{c}_{i,j}]$ either. Thus the new job set $\mathbf{J}'$ obtained at Step 2(biB) contains all the possible jobs that can execute in interval $(r_{i,c}, \hat{c}_{i,j}]$.

Since no job with priority lower than $J_{i,low}$, obtained at Step 2(biC), can execute in interval $(r_{i,c}, \hat{c}_{i,j}]$, $Inter(J_{i,low}, \mathbf{J}')$ will give an upper bound on Γ, the total amount of execution times of jobs that can execute in interval $(r_{i,c}, \hat{c}_{i,j}]$. By Step 2(biD) and Step 2(bii), we will have

$$\begin{aligned}\Gamma + \sum_{l=c}^{j} e^+_{i,l} &\leq Inter(J_{i,low}, \mathbf{J}') + \sum_{l=c}^{j} e^+_{i,l} = b_{i,c} - r_{i,c} \\ &\leq \hat{c}_{i,j} - r_{i,c}\end{aligned}$$

This contradicts the conclusion we obtained in the previous paragraph. The hypothesis must be wrong; we must have $c_{i,j} \leq \hat{c}_{i,j}$. By induction, we know that for every job in this schedule its completion time is no longer than the corresponding bound computed by Algorithm ITR. □