Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 36 (2009) 1158–1175
computers & operations research
www.elsevier.com/locate/cor

A two-machine flowshop problem with processing time-dependent buffer constraints—An application in multimedia presentations☆

Feng-Cheng Lin[a], Jen-Shin Hong[a,*], Bertrand M.T. Lin[b]

[a]*Department of Computer Science and Information Engineering, National Chi Nan University, Nantou, Taiwan*
[b]*Department of Information and Finance Management, Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan*

Available online 21 January 2008

Abstract

To have a quality multimedia presentation through networks, its presentation lag needs to be controlled. One way to reduce the lag is to prefetch the media objects before their due dates. This paper explores techniques for optimizing the object sequence in a prefetch-enabled TV-like presentation. An optimal solution is the one with which the presentation lag is minimized. We formulate the problem into a two-machine flowshop scheduling problem with a single chain precedence constraint and a player-side buffer constraint. The player-side buffer is "processing time-dependent" and distinguished from the conventional item-based intermediate buffer constraints discussed in previous flowshop studies. We prove the problem to be strongly NP-hard. A branch and bound algorithm equipped with four lower bounds and an NEH-based upper bound is developed. The simulation results show that the average gaps between the overall lower bounds and the NEH-based upper bound are less than 3% for problems with a large buffer size, and less than 13% for problems with a small buffer size and high density of precedence constraints. For applications where the media objects are delivered through extremely busy servers with which only very restricted CPU resources can be allocated for computation, the CDS-based algorithm provides better sequences than the NEH-based algorithm.

Keywords: Multimedia presentation; Object sequence optimization; Buffer constraint; Two-machine flowshop

1. Introduction

Rapid advances in technologies for media capture, storage and transmission have contributed to an exponential growth of multimedia objects on the Internet. A common way to present the archived media objects in online multimedia communication applications is by a TV-like presentation which continuously plays the media objects in sequence. Applications that prefer a TV-like presentation can originate from any systems that need to combine multiple separate media objects into a continuous presentation. Examples include assembling a TV-like documentary based on queries to multimedia databases, continuously showing the media items in an online multimedia album, presenting a personalized multimedia daily news delivery service, or presenting a multimedia message (MMS) in a mobile phone, etc. A TV-like presentation consists of a set of multimedia objects ordered sequentially. In these dynamically generated presentations, the orders of the objects are dynamically assigned at runtime.

☆ This research was partially supported by the National Science Council of Taiwan, under grant NSC 96-2221-E-260-020-MY3. B.M.T. Lin was also partially supported by the National Science Council of Taiwan under grant NSC-95-2416-H-009-033.
* Corresponding author. Tel.: +886 49 2915225; fax: +886 49 2915226.
E-mail address: jshong@ncnu.edu.tw (J.-S. Hong).

0305-0548/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2008.01.002

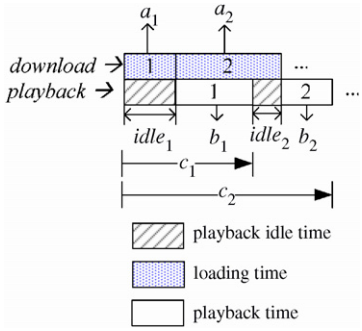

Fig. 1. Graphical illustration of the notations.

Table 1
Correspondence between the media object ordering problem and the two-machine flowshop problem

Media object scheduling for TV-like presentations	Two-machine flowshop scheduling	Notation
A set of media objects	A set of jobs	N
A media object	A job	J_i
Server (send data to the client)	First machine	M_1
Client (receive/playback data from the server)	Second machine	M_2
Download time for a media object	Processing time (download time) on first machine	a_i
Playback time for a media object	Processing time (playback time) on second machine	b_i
Completion time of playback of a media object	Completion time of each job J_i	C_i
Transmission bit rate (bytes per second)		BR
File size of a media object (bytes)		s_i
Buffer size	Total size of jobs allowed in buffer	BUF
The completion of the presentation	Makespan (the maximum of C_i $(1 \leqslant i \leqslant n)$)	$C_{\max}$

To have a quality TV-like presentation through the Internet, the presentation lag, which is defined as the difference between the instant a media object really begins and the instant it was previously scheduled to start, needs to be kept controlled. A commonly used strategy to possibly reduce the total presentation lag of a presentation is to prefetch the objects before their due times. Each prefetched media object occupies the player-side buffer until it finishes the playback (here we assume that the media player provides the first-in, first-out buffer control service). The prefetched objects are stored in the buffer for promptly responding to a user request. Such a prefetch mechanism usually can reduce presentation lags by acquiring the future objects during the current object's view time. When a media object is being played, there could be un-occupied buffer space available for further downloading other subsequent media objects into the buffer.

In typical multimedia presentations, the modalities of the media objects could be text, images (e.g., JPG, GIF), audio (e.g., MP3, MIDI), video (e.g., MPEG, AVI), or vector graphics (e.g., SVG, SWF). Each media type may have a unique data compression ratio. In addition, the objects could be distributed across multiple servers with varied end-to-end bandwidths to the media player. Thus, for the same amount of data transmitted, the anticipated presentation time differs drastically for different media types. If the media objects are ordered appropriately in the server, the total presentation lag perceived by users for a prefetch-enabled TV-like presentation could be significantly reduced.

Since each media object has a download time (a_i) and playback time (b_i) (refer to Fig. 1), so as to reduce the total presentation lag, the order of these media objects should be properly arranged. Our work explores the optimization techniques for sequencing a set of media objects in a delay prone environment such that the presentation lag is minimized. Based on the corresponding elements list in Table 1,we find that the object ordering problem actually can be formulated as a two-machine flowshop problem $F2\|C_{\max}$ [1], in which $F2$ means the machine environment and $C_{\max}$ represents the objective. However, the player-side buffer constraint, as described below, appears to be a new type of constraint which has never been explored in the existing two-machine flowshop problems.

In general, two types of constraints are commonly encountered in online multimedia applications, namely, the inter-object precedence constraint, and the player-side buffer constraint. A precedence constraint (here we adopt the terminology used in conventional scheduling research as described in Brucker [1]) requires one media object to precede the second object without precluding other objects from coming between them. For example, in an educational lecture video of a math course, a video object illustrating the *proof* of a *theorem* must be preceded (but not necessarily immediately) by the video defining the theorem itself. In such a case, other objects (such as example applications or the historical remarks of the theorem) are allowed to be presented between the two objects. Beyond the precedence constraints, for online multimedia applications with which certain portable devices such as PDA are used for the playback, the maximum buffer size allowed for keeping the prefetched objects needs to be considered in the schedule optimization process.

If two jobs (i.e., media objects), J_i and J_j are temporally related by a precedence constraint (denoted by $J_i \rightarrow J_j$), J_i must precede J_j but not necessarily next to J_j. Note that there are two different interpretations of the precedence constraint addressed in the flowshop problems (refer to [2,3]). Assuming that job J_i precedes job J_j, in the first interpretation, J_j cannot be started on the first machine until J_i is completed on the second machine (all machines). The two-machine flowshop scheduling problem with the first type of precedence constraint is NP-hard even if the given graph contains only one arc [4,5].

In the second interpretation, J_j cannot start on the first machine unless J_i is completed on the same machine, that is, J_j can start on the first machine while J_i is being processed on machine two. The precedence constraint considered in the multimedia presentation applications refers to the second interpretation. In general, by a series of concatenation on the constrained objects, the overall precedence constraint for a multimedia presentation might be presented by a general graph. Two-machine flowshop scheduling of makespan minimization in applications with arbitrary precedence constraints (i.e., the $F2|prec|C_{\max}$ problem) has been proved to be NP-hard [6]. To efficiently compute a near-optimal solution using numerical approaches, Hariri and Potts [7] and McMahon and Lim [8] developed a variety of branch and bound algorithms where various lower bounds, upper bounds, and dominance rules were proposed to speed up the enumerative procedure. The two-machine flowshop problem with a number of simplified versions of the precedence constraint, including chain, tree, sp-graph, can be solved in polynomial time. Kurisu [9] gave an algorithm for the case of parallel-chain precedence constraints. A tree-based precedence problem $F2|tree|C_{\max}$ is actually a special case of the series-parallel precedence problem ($F2|sp\text{-}graph|C_{\max}$) which was solved by polynomial algorithms in Monma and Sidney [10], Sidney [11] and Gordon and Shafransky [12]. Valdes et al. [13] presented a linear-time algorithm to recognize series-parallel digraphs. In this paper, we consider only the multimedia presentations where the overall precedence constraint is in the form of a single chain. Problems with more complicated forms of precedence constraint is currently under investigation.

With respect to the player-side buffer constraint, we did not find any existing flowshop scheduling studies that have addressed the problem. In the literature, there are a few studies investigating flowshop problems with limitation of "number of jobs" that can be allocated in the intermediate storage buffer before flowing to the next machine. For the case without any buffer limitation (denoted as the problem ∞-*buffer* problem in [14]), an optimal solution can be obtained by Johnson's rule [23]. For the case in which no any job is allowed to be kept in the intermediate storage buffer (denoted as the *zero-buffer* or *no-wait* problem in [14]), the polynomial algorithm developed by Gilmore and Gomory [15] for a specific TSP can be used to produce optimal solutions. Other than the above two cases, the buffer-constrained machine flowshop scheduling problem has been shown to be strongly NP-hard by Papadimitriou and Kanellakis [14]. To obtain the exact solution of this NP-hard scheduling problem with a finite intermediate buffer, a number of strategies have been proposed; for example, Leisten [16], Smutnicki [17], Dutta and Cunningham [18], Brucker et al. [19], Tang and Xuan [20], Wang et al. [21], etc. In these flowshop studies, the number of jobs allowed to keep in the buffer is fixed throughout the process. In the online multimedia applications addressed in this paper, we fix the "buffer size" instead of number of objects allowed in the player-side buffer. For a given buffer size, the number of objects allowed in the buffer actually depends on the file sizes of the current playing objects and other unscheduled objects. Since, under an environment with a constant transmission bit rate, the download time of an object (i.e., the processing time on M_1) is proportional to its file size, the number of objects allowed in the buffer is actually "processing time-dependent". Hence, the media scheduling problem we want to solve is more complicated than the above-mentioned buffer-constrained flowshop scheduling problems. Another unique feature our buffer model exhibits is that the buffer resides on machine M_2 instead of in an intermediate storage area between two machines. Therefore, a job currently being processed on machine M_2 still occupies the buffer until its completion. In conventional

scheduling problems with buffers, a job released from the buffer for processing immediately frees the space it has acquired.

In summary, two constraints are taken into account in the two-machine flowshop problem to be explored in this paper, including a single chain precedence constraint of a subset of jobs, and a processing time-dependent buffer constraint. We denote the problem addressed in this paper by $F2|a_i\text{-}buffer, chain|C_{\max}$. The jobs that are not constrained by the precedence chain are mutually independent and can be inserted in any place of the chain. Furthermore, since most web servers do not support resumable download with which the transmission of a media object can be temporarily halt and resume later, the jobs are assumed to be *non-preemptive*. For a problem with a buffer capacity larger than the sum of all objects, in the case without a precedence constraint, the optimal sequence can be obtained using Johnson's rule (refer to [22]). In Johnson's rule, job J_i precedes job J_j if $\min\{a_i, b_j\} \leqslant \min\{b_i, a_j\}$. In the case with a single chain precedence constraint, the optimal sequence can be obtained using the algorithm developed in Kurisu [9]. Note that previous works by Allahverdi and Al-Anzi [24], Batra and Eleftheriadis [25] and Blazewicz et al. [26] also addressed the scheduling of objects in multimedia applications. However, the applications and optimization criteria addressed in these studies are different from this paper.

In the following, we introduce the notation which will be referred throughout this paper.

Notation:

$N = \{J_1, J_2, \ldots, J_n\}$	set of n jobs
J_i	a specific job in N
N_{SD}	a subset of N in which the jobs are already scheduled
$N \backslash N_{SD}$	a subset of N in which the jobs are not yet scheduled
S_N	a sequence of jobs of set N
$S_{N_{SD}}$	a sequence of jobs of set N_{SD}
a_i	processing time (download time) of J_i on machine one M_1
b_i	processing time (playback time) of J_i on machine two M_2
BR	transmission bit rate in byte/s
C_i	completion time of J_i
$idle_i$	idle time before J_i on M_2
C_{SD}	maximum completion time of the jobs in sequence $S_{N_{SD}}$
$C_{\max}$	maximum completion time of the jobs in sequence S_N
$N_P = \{J_{P1}, J_{P2}, \ldots, J_{Pk}\}$	a subset of jobs that are subject to a single precedence chain, where J_{P1} must be played before (but not necessarily immediately next to) J_{P2} J_{Pk} must be the last one in N_P to be played
$N_{NSDP} = \{J_{NSDP1}, J_{NSDP2}, \ldots, J_{NSDPk}\}$	a subset of unscheduled jobs (i.e., $N \backslash N_{SD}$) in which jobs are not subject to any precedenceconstraint
N_{BUF}	a subset of jobs that currently reside in the buffer
$a_{\min}$	the minimum processing time (download time) of a_i among the jobs of N
$b_{\min}$	the minimum processing time (playback time) of b_i among the jobs of N
$a_{\text{prec-first}}$	download time of the job with the highest priority in the set of unscheduled jobs of N_p (i.e., $N_p \cap (N \backslash N_{SD})$)
$b_{\text{prec-last}}$	playback time of the job with the lowest priority in the set of unscheduled jobs of N_p (i.e., $N_p \cap (N \backslash N_{SD})$).

2. Strong NP-hardness of the $F2|a_i\text{-}buffer|C_{\max}$ problem

In this section, we show that problem $F2|a_i\text{-}buffer|C_{\max}$ is strongly NP-hard by a reduction from the 3-Partition problem, which is known to be NP-hard in the strong sense (see [31]).

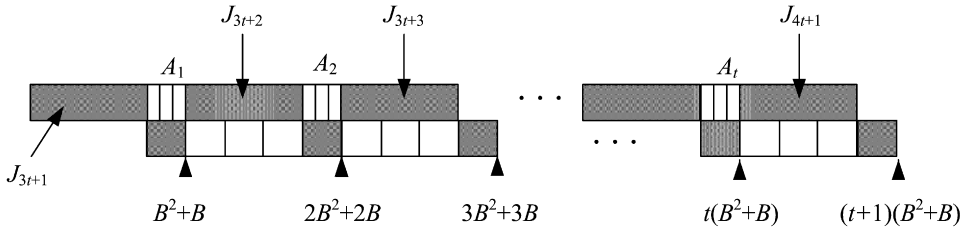

Fig. 2. Gantt chart of the optimal schedule with $C_{\max} = Y$ in Theorem 1.

3-*Partition*: Given an integer B and a set of A of $3t$ positive integers $\{x_1, x_2, \ldots, x_{3t}\}$, $B/4 < x_i < B/2$, $1 \leqslant i \leqslant 3t$, such that $\sum_{i=1}^{3t} x_i = tB$, does there exist a partition $A_1, A_2, \ldots, A_t$ of the set A such that $\sum_{x_i \in A_l} x_i = B$, $1 \leqslant l \leqslant t$?

Theorem 1. *The* $F2|a_i$*-buffer*$|C_{\max}$ *problem is strongly* NP*-hard.*

Proof. It is not hard to see that the decision version of $F2|a_i$*-buffer*$|C_{\max}$ is in the NP class. We next perform a polynomial-time reduction from 3-Partition. Given an instance of 3-Partition, we construct a job set $N=\{1, 2, \ldots, 4t+1\}$ as follows:

Ordinary jobs:

$$a_i = x_i, \quad b_i = Bx_i, \quad 1 \leqslant i \leqslant 3t,$$

Enforcer jobs:

$$a_{3t+i} = B^2, \quad b_{3t+i} = B, \quad 1 \leqslant i \leqslant t+1,$$

$$Y = (t+1)(B^2 + B),$$

Buffer capacity $= B^2 + B$.

The network bit rate, *BR* is assumed to be one unit, i.e., $s_i = a_i$ for all jobs.

With the above instance, we claim that there is a partition for the set A as specified in 3-Partition if and only if there exists an optimal schedule of $F2|a_i$*-buffer*$|C_{\max}$ whose makespan is no greater than Y. Before proceeding to the proof, we assume $t < x_i$ for all elements of A. If it is not the case, we can scale the numerical values in polynomial time to meet the inequality.

If part: Let subsets $A_1, A_2, \ldots, A_t$ be a partition as specified for the set A in 3-Partition. We schedule job J_{3t+1} first, followed by the jobs corresponding to the elements of A_1. Next, we schedule job J_{3t+2} and then the jobs corresponding to the elements of A_2. Continuing the arrangement, we have a schedule as shown in Fig. 2. Inspecting the Gantt chart, we can easily see that the constructed schedule has a makespan Y.

Only if part: Suppose that there is an optimal schedule of $F2|a_i$*-buffer*$|C_{\max}$ with a makespan no greater than $Y = (t+1)(B^2 + B)$. We first note that, without loss of generality, we can assume the enforcer jobs are scheduled in increasing order of their indices. Due to the buffer constraint, no two enforcer jobs are allowed to stay in the buffer at the same time. This implies that for any two enforcer jobs, say for example J_{3t+1} and J_{3t+2}, the machine-two operation of J_{3t+1} and the machine-one operation of J_{3t+2} cannot overlap. This can be illustrated by the Gantt chart given in Fig. 2. With this property and the fact that $\sum_{i=3t+1}^{4t} a_i + \sum_{i=3t+1}^{4t} b_i = (t+1)(B^2 + B) = Y$, no ordinary job can be scheduled before enforcer job J_{3t+1} for otherwise the makespan will be greater than Y. That is, the ordinary jobs need to be allocated to fully fill up the intervals in between any two consecutive enforcer jobs. Moreover, no idle time on machine one is allowed and no idle time between any two machine-two operations is allowed.

Let N_1 be the set of ordinary jobs scheduled between jobs J_{3t+1} and J_{3t+2}. We analyze two disjoint cases.

Case 1: $\sum_{J_i \in N_1} a_i < B$. The completion time of J_{3t+2} on machine M_1 is $B^2 + \sum_{J_i \in N_1} a_i + B^2$. On the other hand, on machine M_2, the completion time of the last job in N_1 is $B^2 + B + \sum_{J_i} a_i B$. Consider the following

chain of derivations

$$\begin{aligned}
&\left(B^2+\sum_{J_i} a_i + B^2\right) - \left(B^2 + B + \sum_{J_i} a_i B\right)\\
&= B^2 + \sum_{J_i} a_i - B - \sum_{J_i} a_i B\\
&= B(B-1) - \sum_{J_i} a_i\,(B-1)\\
&= \left(B - \sum_{J_i} a_i\right)(B-1)\\
&> 0.
\end{aligned}$$

The strict inequality indicates that on machine M_2 a non-zero idle time exists between the last job of N_1 and job J_{3t+2}.

Case 2: $\sum_{J_i \in N_1} a_i > B$. Let $J_{i_1}, J_{i_2}, \ldots, J_{i_k}$ be the ordinary jobs of N_1 as arranged in the schedule. Let J_{i_j}, $1 \leqslant j \leqslant k$, be the first job such that $a_{i_1} + a_{i_2} + \cdots + a_{i_{j-1}} \leqslant B$ and $a_{i_1} + a_{i_2} + \cdots + a_{i_j} > B$.

Case 2.1: $a_{i_1} + a_{i_2} + \cdots + a_{i_{j-1}} < B$. When job J_{i_j} is considered for processing on machine one, the buffer contains enforcer job J_{3t+1} and ordinary jobs $J_{i_1}, J_{i_2}, \ldots, J_{i_{j-1}}$. The residual space in the buffer is less than a_{i_j} because $a_{i_1} + a_{i_2} + \cdots + a_{i_j} > B$. Therefore, job J_{i_j} needs to wait for $B - (a_{i_1} + a_{i_2} + \cdots + a_{i_{j-1}})$ time units for the completion of the playback of job J_{3t+1}. As a sequel, non-zero idle time occurs, a contradiction.

Case 2.2: $a_{i_1} + a_{i_2} + \cdots + a_{i_{j-1}} = B$. The completion time of the first ordinary job J_{i_1} on machine M_2 is $B^2 + B + a_{i_1} B$. On machine M_1, the completion time of the last job J_{i_k} of N_1 is at most $B^2 + tB$, which is smaller than $B^2 + B + a_{i_1} B$ as we have assumed $tB < x_i B$. When job J_{3t+1} is to be dispatched to machine M_1, all jobs of N_1 still occupy the buffer with a total size greater than B. Subject to the capacity constraint, we cannot dispatch job J_{3t+1} onto machine M_1, implying non-zero idle time before J_{3t+1}.

From the analysis of Cases 2.1 and 2.2, we known that $\sum_{J_i \in N_1} a_i > B$ cannot hold.

The above analysis has shown that $\sum_{J_i \in N_1} a_i = B$ must hold. We let the elements that define the jobs of N_1 constitute subset A_1. Continuing the above line of analysis, we can iteratively come up with A_2, A_3 and A_t as required by the 3-Partition problem. The proof is complete. □

The instance constructed in the proof of $F2|a_i\text{-}buffer|C_{\max}$ is a degenerate case of $F2|a_i\text{-}buffer, chain|C_{\max}$ without any precedence constraint. The studied problem $F2|a_i\text{-}buffer, chain|C_{\max}$, with precedence constraints incorporated, is more complicated and thus also computationally intractable. An extreme case of $F2|a_i\text{-}buffer, chain|C_{\max}$ has a chain consisting of all jobs. In this case, the sequence defined by the chain is the solution.

3. Branch and bound algorithm

As the media object scheduling problem is strongly NP-hard, it is unlikely to develop polynomial time algorithms for producing optimal schedules. In this section, we shall develop a branch and bound algorithm for the $F2|a_i\text{-}buffer, chain|C_{\max}$ problem. To support the design of an efficient branch and bound algorithm, we develop several lower bounds to curtail unnecessary branching for reducing the computing efforts. An NEH-based heuristic algorithm will also be proposed to produce approximate solutions that will be used as the initial incumbent value of the exact method.

3.1. Lower bounds based on the precedence constraint

In a node in the branch and bound tree, there should be a set of scheduled jobs (i.e., N_{SD}) and other unscheduled jobs (i.e., $N \backslash N_{SD}$). The scheduled jobs include jobs finished on M_2, the job currently under processing on M_2 (been

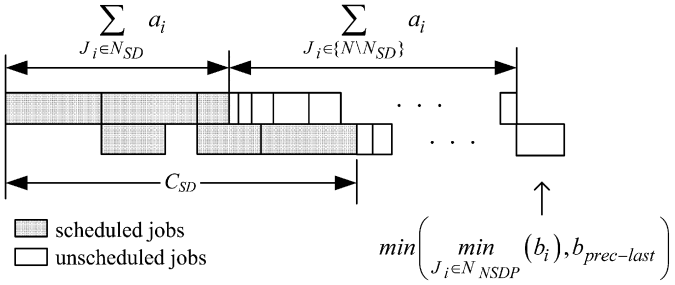

Fig. 3. Lower bound LB_1.

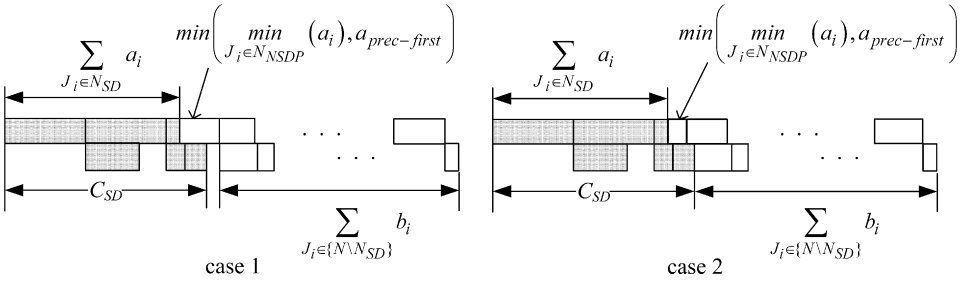

Fig. 4. Lower bound LB_2.

previously downloaded to the buffer), and buffered jobs waiting for playback on M_2. The following two lower bounds, LB_1 and LB_2, are based on certain characteristics of the precedence constraint.

3.1.1. LB_1

Referring to Fig. 3, a lower bound at a node must be larger than the total a_i for all J_i (i.e., $\sum_{J_i \in N_{SD}} a_i + \sum_{J_i \in \{N \setminus N_{SD}\}} a_i$ in Fig. 3) plus the processing time for the final job on M_2. Given N_S as the set of scheduled jobs, the last job in a sequence should be either J_{Pk} (i.e., the last one to be played in J_P) or the job with the smallest b_i in N_{NSDP} ($N_{NSDP} = (N \setminus N_{SD}) \cap (N \setminus N_P)$, i.e., the unscheduled jobs that are not subject to any precedence constraint), whichever has a smaller playback time b_i. Based on the above consideration, a lower bound LB_1 can be given as

$$LB_1 = \sum_{J_i \in N} a_i + \min \left\{ \min_{J_i \in N_{NSDP}} \{b_i\}, b_{\text{prec-last}} \right\}. \tag{1}$$

3.1.2. LB_2

At a node in the search tree, a lower bound must be larger than the total processing time for all unscheduled jobs on M_2 (i.e., $\sum_{J_i \in \{N \setminus N_{SD}\}} b_i$ in Fig. 4), plus the time elapsed before these jobs starting on M_2. The earliest job to be scheduled at this stage should be the one with the highest priority in $(N \setminus N_{SD})$, or the one with the smallest a_i in N_{NSDP}, whichever has a smaller a_i. Therefore, the time elapsed can be given by $\min(\min_{J_i \in N_{NSDP}} (a_i), a_{\text{prec-first}})$. Given C_{SD} as the flowtime of sequence $S_{N_{SD}}$ of job set N_{SD}, a lower bound can be calculated as

$$LB_2 = \max \left(C_{SD}, \sum_{J_i \in N_{SD}} a_i + \min \left(\min_{J_i \in N_{NSDP}} (a_i), a_{\text{prec-first}} \right) \right) + \sum_{J_i \in N \setminus N_{SD}} b_i. \tag{2}$$

3.2. Lower bounds based on the buffer constraint

We further propose two more sophisticated lower bounds that exploit certain general characteristics of a buffer-constrained sequence. At any instant during the presentation, if there is certain vacancy in the buffer, more unscheduled

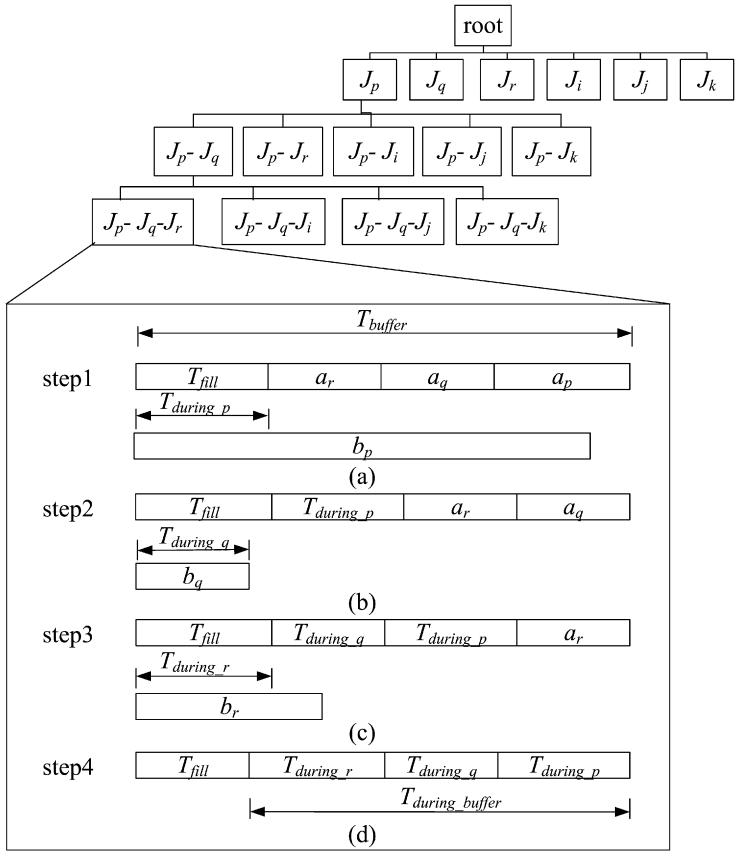

Fig. 5. Buffer status at the four playback steps.

job(s) can be possibly downloaded into the buffer (that is, M_1 can process further unscheduled jobs). Each time a job finishes on M_2, the newly released buffer space can then be used to download more unscheduled job(s). Note that based on the requirements of real world multimedia applications, we assume the download process needs to be suspended unless the currently available free space in the buffer is sufficient for keeping an incoming media object, that is, the download process is not resumable.

3.2.1. LB_3

In a node of the search tree, given the current makespan of the sequence $S_{N_{SD}}$ as C_{SD}, a reasonable estimate on the minimum remaining processing time is the total processing time (a_i) required for all unscheduled jobs minus the total allowable processing time on M_1 before all the buffered jobs finish their playbacks. Fig. 5 illustrates the general idea of LB_3. The calculation is based on the following three values: (1) duration required to fully fill up an empty buffer (denoted by T_{buffer}), (2) duration allowed for further download after J_p starts its playback (denoted by T_{fill}) and (3) duration allowed for further download during the processing of J_p (denoted by $T_{\text{during\_}p}$).

Refer to Fig. 5 and consider an instant when a job J_p is under processing on M_2, with a number of other jobs waiting in the buffer (e.g., J_q and J_r in Fig. 5). Assuming the data transmission bit rate on M_1 (denoted by BR defined above) to be constant, the time required to completely fill up the empty buffer is calculated as

$$T_{\text{buffer}} = \frac{BUF}{BR}. \tag{3}$$

At a node, the current free buffer space (denoted by BUF_{free}) is the total buffer size minus the size of the job being processed on M_2 and the standby job(s) in the buffer. Hence, the duration allowed for further transmission after J_p

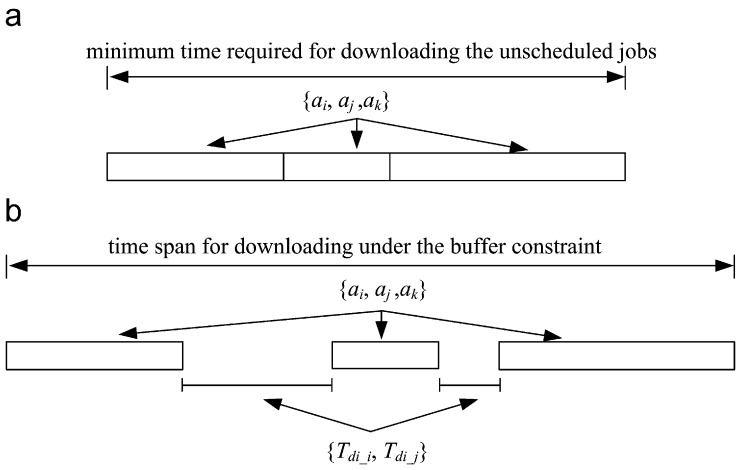

Fig. 6. Calculation of T_R.

starts its playback is given as

$$T_{\text{fill}} = \frac{BUF_{\text{free}}}{BR}. \tag{4}$$

There are two different scenarios for calculating the duration allowed for further download during the processing of a job J_i on M_2 (denoted by $T_{\text{during\_}i}$). In the first case (step 1, Fig. 5(a)), b_p is larger than T_{fill}. So, during the processing of J_p on M_2, the download process on M_1 will be suspended once the buffer is full. The maximum allowable download time is equal to T_{fill}. Hence, $T_{\text{during\_}p} = T_{\text{fill}}$.

In the second case (step 2, Fig. 5(b)), the buffer will not be fully occupied before J_q completes on M_2. The buffer has a free space for downloading the next job, say J_r. In such a case, $T_{\text{during\_}q} = b_q$.

By combining the above two scenarios, we compute the allowable download time $T_{\text{during\_}i}$ during the playback of a buffered job J_i as

$$T_{\text{during\_}i} = \min(T_{\text{fill}}, b_i). \tag{5}$$

The total allowable download time for all buffered jobs at that node can then be given as

$$T_{\text{during\_buffer}} = \sum_{J_i \in N_{BUF}} T_{\text{during\_}i}. \tag{6}$$

We now propose a lower bound based on $T_{\text{during\_buffer}}$. Given C_{SD} as the current makespan of the scheduled jobs, the makespan of the complete sequence N_{SD} should be larger than a presumed ideal sequence with which all the unscheduled jobs can be processed on M_1 during the playback of all the buffer jobs on M_2 (i.e., $\sum_{J_i \in N \setminus N_{SD}} a_i < T_{\text{during\_buffer}}$). In such a case, a lower bound can be given by summing C_{SD} and the total processing time of the unscheduled jobs on M_2. That is, a lower bound is given as $C_{SD} + \sum_{J_i \in N \setminus N_{SD}} b_i$.

In the case where the total processing time on M_1 of the unscheduled jobs is larger than $T_{\text{during\_buffer}}$ (i.e., $\sum_{J_i \in N \setminus N_{SD}} a_i \geqslant T_{\text{during\_buffer}}$), the makespan should then be the sum of the following values: (1) the current makespan C_{SD}, (2) the remaining time required to process all the unscheduled jobs on M_1 after the playback of current buffered jobs and (3) the total download idle time (the time that cannot be allocated for downloading) when an unscheduled job is under playback on M_2. In the following, we elaborate the general idea of this lower bound.

During the playback of the currently buffered jobs on M_2 (i.e., within C_{SD}), there will be a time slot of $T_{\text{during\_buffer}}$ for possible download of the unscheduled jobs on M_1. The minimum remaining time required to download the unscheduled jobs after the completion time of current buffered jobs is given as $T_R = \sum_{J_i \in N \setminus N_{SD}} a_i - T_{\text{during\_buffer}}$.

Referring to Fig. 6(b), due to the buffer constraint, the time span for downloading all the unscheduled jobs could be larger than T_R. When a job J_i is processed on M_2, the buffer is at least occupied by J_i itself. Hence, the maximum

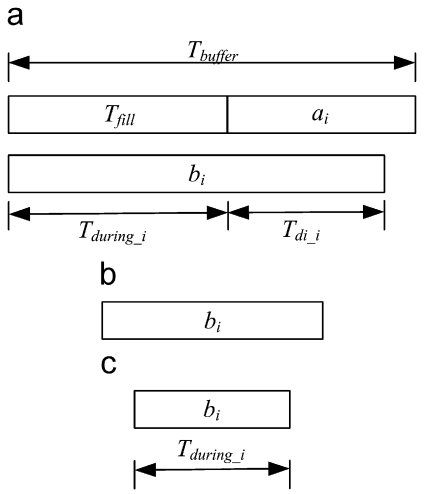

Fig. 7. Calculation of the minimum downloading idle time T_{di_i} for J_i.

allowable download time on M_1 during its playback is $T_{\text{buffer}} - a_i$. When $T_{\text{during\_}i} < b_i$ (Fig. 7(a)), the minimum download idle time $T_{di\_i} = b_i - (T_{\text{buffer}} - a_i)$. Otherwise, $T_{di\_i} = 0$ (Figs. 7(b) and (c)).

Therefore, T_{di_i} can then be given as

$$T_{di\_i} = \max\{b_i - (T_{\text{buffer}} - a_i), 0\}. \tag{7}$$

The minimum download idle time T_{DI} for all the unscheduled jobs is the sum of T_{di_i} of all unscheduled jobs

$$T_{DI} = \sum_{J_i \in N \setminus N_{SD}} T_{di\_i} = \sum_{J_i \in N \setminus N_{SD}} \max\{b_i - (T_{\text{buffer}} - a_i), 0\}. \tag{8}$$

Thus, a lower bound can be given by $C_{SD} + T_R + T_{DI}$.

In summary, we have the following lower bound:

$$LB_3 = \begin{cases} C_{SD} + \sum_{J_i \in N \setminus N_{SD}} b_i & \text{if } \sum_{J_i \in N \setminus N_{SD}} a_i < T_{\text{during\_buffer}} \\ C_{SD} + T_R + T_{DI} & \text{otherwise.} \end{cases}$$

3.2.2. LB_4

With further elaboration, we can improve the tightness of the previous lower bound LB_3 by calculating the best possible job combination so as to optimize the utilization of the residual buffer space in any instance.

Given a partial schedule with J_p starting its playback, there are two cases to consider for calculating $T_{\text{during\_}p}$ (duration allowed for further download during the processing of J_p). In the first case (Fig. 8(a)), b_q is smaller than T_{fill}. The buffer will not be fully occupied before the completion of J_q. In such a case, $T_{\text{during\_}q} = b_q$. In the second case (Fig. 8(b)), b_p is larger than the T_{fill}. The maximum allowable download time $T_{\text{during\_}p}$ will depend on the possible combinations of the unscheduled jobs (e.g., in Fig. 8(b), $\{a_i, a_k\}$ occupies more buffer space than a_j). The best allocation can be calculated using techniques of the 0-1 Knapsack problem [27]. In the studied case, we can compute $T_{\text{during\_}i}$ using algorithms of the 0-1 Knapsack problem by mapping a_i to the item values and T_{fill} to the knapsack capacity. The objective is then to determine an optimal combination of a_i's among the unscheduled jobs with which $T_{\text{during\_}i}$ is maximized. The 0-1 Knapsack problem can be solved by dynamic programming algorithms (denoted by DP) in $O(nT_{\text{fill}})$ time, where n is the number of jobs. We denote the solution obtained from the Knapsack problem by $T_{\text{during\_knapsack}}$.

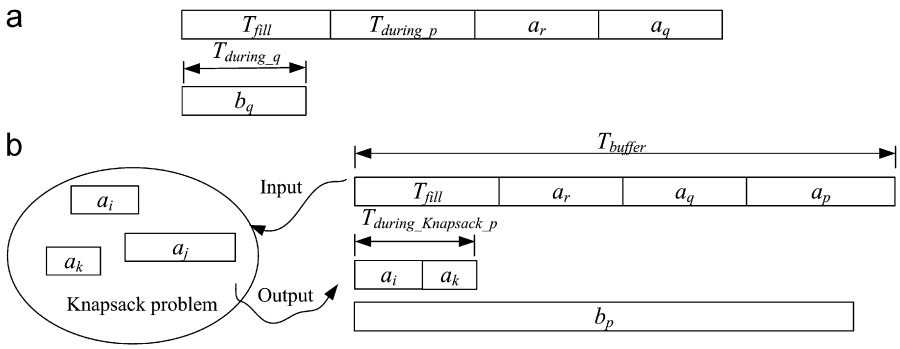

Fig. 8. Maximum allowable download time.

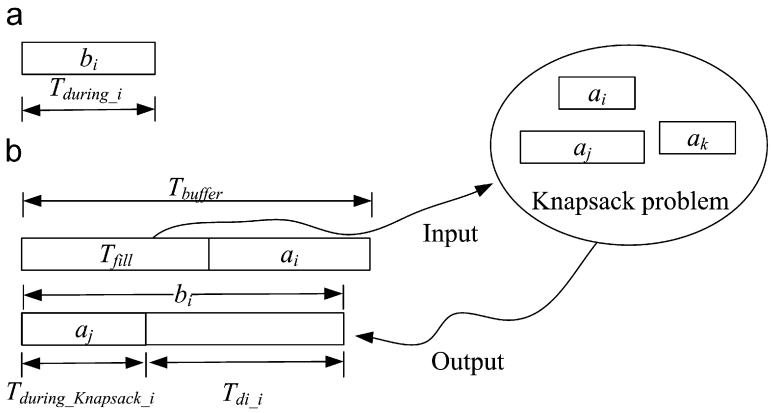

Fig. 9. Calculation of $T_{\text{during\_}i}$ by dynamic programming.

By combining the above two cases, we can compute the allowable download time during the playback of a buffered job J_i by

$$\begin{aligned} &\text{If } T_{\text{fill}} > b_i \text{ (Fig. 9(a)), then } T_{\text{during\_}i} = b_i, \\ &\text{Else } T_{\text{during\_}i} = T_{\text{during\_Knapsack\_}i} \text{ (Fig. 9(b)).} \end{aligned} \tag{9}$$

The total allowable download time for all buffered jobs in the partial schedule is given as

$$T_{\text{during\_buffer}} = \sum_{J_i \in N_{BUF}} T_{\text{during\_}i}. \tag{10}$$

The minimum remaining time required to download the unscheduled jobs after the playback of the currently buffered jobs is

$$T_R = \max\left\{ \sum_{J_i \in N \setminus N_{SD}} a_i - T_{\text{during\_buffer}}, 0 \right\}. \tag{11}$$

Under the buffer constraint, for any sequence, the time span for downloading all unscheduled jobs should be no less than T_R due to the possible idle time in the download process. When J_i is being played, the buffer is at least occupied by J_i. Hence, the maximum of allowable download time during its processing is $T_{\text{buffer}} - a_i$. When $T_{\text{fill}} > b_i$ (Fig. 9(a)), $T_{di\_i} = 0$. When $T_{\text{fill}} \leqslant b_i$ (Fig. 9(b)), $T_{\text{during\_}i}$ can be computed by using the dynamic programming algorithms for solving the Knapsack problem. The download idle time for J_i can then be given as

$$T_{di\_i} = b_i - T_{\text{during\_Knapsack\_}i} \tag{12}$$

By combining the above two cases, T_{di_i} can be given as

$$\text{If } T_{\text{fill}} > b_i, \quad \text{then } T_{di\_i} = 0, \quad \text{else } T_{di\_i} = b_i - T_{\text{during\_Knapsack\_}i}. \tag{13}$$

The minimum download idle time for all the unscheduled jobs is the sum of the download idle times of all the unscheduled jobs, given as

$$T_{DI} = \sum_{J_i \in N \setminus N_{SD}} T_{di\_i} = \sum_{J_i \in N \setminus N_{SD}} \max\{b_i - T_{\text{during\_Knapsack\_}i}, 0\}. \tag{14}$$

Summarizing the above discussions, we have

$$\begin{aligned} \text{LB}_4 &= C_{SD} + T_R + T_{DI} \\ &= C_{SD} + \max\left\{ \sum_{J_i \in N \setminus N_{SD}} a_i - \sum_{J_i \in N_{BUF}} T_{\text{during\_}i}, 0 \right\} + T_{DI}. \end{aligned} \tag{15}$$

3.3. Upper bounds

We apply the NEH algorithm proposed by Nawaz et al. [28] to obtain an upper bound. The method has been widely applied in various flowshop scheduling problems with pretty good performances [29,30].

We outline the major steps in the NEH method as follows:

NEH-based Algorithm—*UB*:

Step 1: For each J_i, compute $c_i = a_i + b_i$. Sort the jobs by non-increasing c_i.

Step 2: Take the first two jobs in the list of sorted jobs in Step 1, and order them (without violating the precedence constraint) so as to minimize the partial makespan.

Step 3: For $k = 3$ to n do. Insert the kth job into the partial schedule. There are k possible positions for this insertion. Select the position which minimizes the partial makespan while compliant to the precedence constraint.

Step 4: Output the final sequence from step 3.

Note that for applications where the multimedia servers typically handle a huge number of requests simultaneously, the CPU resource that can be allocated for each request to calculate the heuristic sequence is extremely limited. Therefore, the computation time required by the proposed NEH-based heuristic solutions needs to be addressed. In such a case, a faster algorithm giving a potentially worse schedule than NEH-based solutions might be able to give a smaller value of the summation of the computation time and the scheduled makespan. Therefore, for application with busy servers, we also apply a much more efficient algorithm—the CDS algorithm, which was proposed by Campbell et al. [32] in 1970. The CDS algorithm is a generalization of Johnson's algorithm and has been frequently used for obtaining approximate solutions to m-machine flowshop scheduling problems without precedence constraints.

4. Computational results

This section presents the computational experiments designed to evaluate the effectiveness and efficiency of the proposed lower bounds and upper bound for producing an optimal media sequence. For each job set, processing times a_i and b_i were randomly generated from the uniform interval [1, 100]. The transmission bite rate was set to be 160 KB/s (i.e., $BR = 160$ K). The simulation codes were written in C++ language, and the experiments were performed on IBM ×Series ×206 m computers running Microsoft Window Server 2003. All numerical values of time reported are in CPU-seconds(s). We conducted a series of computational experiments with different problem sizes (small and big size). The instance generation procedure yields test instances encompassing a wide variety of real life scenarios of online multimedia applications. In each media set, we randomly assigned certain precedence constraints among a portion (0%, 20% and 40%) of the objects. For each experiment, 50 different media sets were used to investigate the proposed algorithms. Two buffer size constraints, 16,000 KB and 30,720 KB were given for the experiments.

Tables 2–4 list the results for problems with a relatively large buffer size, 30,720 KB. Table 2 shows that, without any precedence constraints, in more than 26 experiments out of the 50 experiments conducted, the overall lower bound

Table 2
Buffer = 30, 720 KB

Deviation (%)															
n	$LB = UB$	UB vs. Opt		Opt vs. LB		UB vs. LB		LB_4				LB_3			
		Avg.	Max.	Avg.	Max.	Avg.	Max.	avg_nodes	max_nodes	avg_time	Unfound	avg_nodes	max_nodes	avg_time	Unfound
8	29	1.04	7.34	0.46	5.52	1.45	10.09	191	2288	0.000	0	337	8293	0.000	0
10	27	0.74	5.89	0.16	3.02	0.87	8.42	5083	99,002	0.008	0	8965	219,171	0.003	0
12	28	0.80	5.08	0.21	2.92	0.98	5.05	8.43E + 04	3.67E + 06	0.037	0	84701	3.67E+06	0.036	0
14	26	0.50	3.12	0.12	1.78	0.61	3.58	2.15E + 07	1.07E + 09	9.930	0	2.19E + 07	1.09E + 09	9.695	0
16	32	0.61	3.94	0.03	0.87	0.62	4.61	NA	NA	NA	1	NA	NA	NA	1
18	26	0.74	5.31	0.07	0.90	0.79	5.04	NA	NA	NA	3	NA	NA	NA	4

No precedence constraint. UB vs. $Opt = \frac{UB-Opt}{Opt} \times 100\%$, Opt vs. $LB = \frac{Opt-LB}{Opt} \times 100\%$, UB vs. $LB = \frac{UB-LB}{UB} \times 100\%$.

Table 3
Buffer = 30, 720 KB

Deviation (%)															
n	$LB = UB$	UB vs. Opt		Opt vs. LB		UB vs. LB		LB_4				LB_3			
		Avg.	Max.	Avg.	Max.	Avg.	Max.	avg_nodes	max_nodes	avg_time	Unfound	avg_nodes	max_nodes	avg_time	Unfound
8	27	1.33	10.67	0.68	7.62	1.93	10.54	162	2606	0.000	0	205	2606	0.000	0
10	22	1.15	10.07	0.39	5.73	1.49	9.96	7683	136,248	0.004	0	9012	136,259	0.004	0
12	21	0.91	8.88	0.19	2.92	1.07	8.15	7.49E + 04	2.95E + 06	0.032	0	7.56E + 04	2.96E + 06	0.032	0
14	21	0.64	4.60	0.14	2.04	0.77	4.40	1.27E + 07	6.07E + 08	8.528	0	1.29E + 07	6.17E + 08	8.637	0
16	25	1.14	7.62	0.06	1.40	1.15	7.08	NA	NA	NA	1	2.83E + 07	1.18E + 09	12.666	0
18	23	1.10	6.73	0.04	0.90	1.10	6.72	NA	NA	NA	4	NA	NA	NA	4

Precedence constraint = 20%.

Table 4
Buffer = 30, 720 KB

Deviation (%)															
n	$LB = UB$	UB vs. Opt		Opt vs. LB		UB vs. LB		LB_4				LB_3			
		Avg.	Max.	Avg.	Max.	Avg.	Max.	avg_nodes	max_nodes	avg_time	Unfound	avg_nodes	max_nodes	avg_time	Unfound
8	22	1.68	10.64	1.55	12.33	3.11	14.47	273	2956	0.000	0	284	2956	0.000	0
10	13	2.84	14.10	1.31	12.36	3.93	14.03	2956	31,294	0.001	0	2996	31,294	0.001	0
12	14	2.25	11.13	0.48	4.21	2.60	12.17	6.51E + 04	1.01E + 06	0.022	0	6.56E + 04	1.01E + 06	0.022	0
14	10	1.68	8.38	0.34	3.84	1.95	7.73	1.06E + 06	1.84E + 07	0.580	0	1.08E + 06	1.87E + 07	0.557	0
16	11	1.94	11.45	0.46	10.10	2.29	11.11	1.77E + 07	2.08E + 08	6.658	0	1.81E + 07	2.14E + 08	6.159	0
18	13	2.53	9.10	0.19	2.65	2.59	8.34	NA	NA	NA	3	NA	NA	NA	3

Precedence constraint = 40%.

Table 5
The deviation between *UB* and *LB*

Precedence	0%			20%			40%		
Gap	*UB* vs. *LB*		*UB* = *LB*	*UB* vs. *LB*		*UB* = *LB*	*UB* vs. *LB*		*UB* = *LB*
Avg./Max.	Avg.	Max.	No.	Avg.	Max.	No.	Avg.	Max.	No.
20	0.652	3.097	19	0.744	3.607	17	2.418	9.626	6
50	0.163	1.030	21	0.407	2.735	18	1.125	4.483	8
100	0.092	0.915	21	0.258	1.478	15	0.728	2.824	3
200	0.058	0.682	19	0.135	1.112	9	0.557	2.166	5
300	0.043	0.370	21	0.109	0.697	7	0.519	1.892	0
500	0.031	0.270	9	0.064	0.341	7	0.332	1.246	1

Buffer = 30, 720 KB.

equals to the NEH-based upper bound ($LB = UB$) at the root node, i.e., optimal solutions were found. For all the experiments, the average error ratios between the optimal solutions (*Opt*) and *UB* or *LB* are less than 1.45%. For problems with more than 14 jobs, the branch and bound algorithm is not very computationally efficient and there were instances for which the branch and bound algorithm cannot find optimal solutions within 10 min. Tables 3 and 4 show that as the density of the precedence constraints increases, the number of instances for which $LB = UB$ decreases as compared to the cases without precedence constraints. On the other hand, the average error ratios between the optimal solutions (*Opt*) and *UB* or *LB* increase accordingly. These results indicate that the proposed lower bounds are more effective in problems with sparse precedence constraints among the jobs. Further, in cases where the buffer size is relatively large, the total number of nodes traversed before the optimal solutions can be obtained is slightly reduced if LB_4 (with the dynamic programming algorithm) is applied. Therefore, we conclude that LB_4 only slightly outperforms LB_3 for problems with a large buffer size.

As the problem size grows, as listed in Table 5, *UB* has an average gap to *LB* less than 2.418%. This result indicates that *UB* can be used as a pretty good approximate solution in real time multimedia applications.

For problems with a small buffer, Tables 6–8 show that the possibility for $LB = UB$ at the root node is minimal. Generally speaking, the gaps between *LB* and *UB* are less than 13% on average. As the number of jobs increases, the average CPU time increases significantly. As the percentages of the precedence constraints increase, the average CPU time decreases. For the problems with 40% precedence constraints, the branch and bound algorithm can handle up to 14 jobs. Furthermore, the total number of nodes traversed for a problem is reduced by about 50% if LB_4 is applied. However, the computation time is roughly doubled in each case. As the problem size increases, Table 9 shows that the average gap between *UB* and *LB* could get to 7.919% for problems without precedence constraints. This gap seems to decrease as the problem size increases. But the gap increases slightly as the density of precedence constraints increases. Still, this result indicates that the *UB* can be used as a rather good approximation solution in real time multimedia applications.

For applications where the multimedia servers typically handle a huge number of simultaneous requests, we have also conducted experiments to compare the performances of the NEH-based algorithm with the CDS-based algorithm. Table 10 lists the computation results of the NEH-based algorithm and the CDS-based algorithm for problems without any precedence constraint. Two different settings of the CPU resources were used in the experiments. The first case simulates a rather busy server with which only 1/100,000 of the CPU time can be allocated to the scheduling task. The second case simulates an extremely busy server with which only 1/1,000,000 of the CPU time can be allocated. The results clearly indicate that for the second case (applications with extremely busy servers), the CDS-based algorithm provides better solutions than the NEH-based algorithm.

5. System implementation

In current web environments, there are a number of technologies available for providing prefetchable multimedia presentations that integrate different modalities of media objects. The most popular ones include SMIL, Flash,

Table 6
Buffer = 16, 000 KB

16, 000 KB	Deviation (%)													
n	$LB = UB$	UB vs. Opt		Opt vs. LB		UB vs. LB		LB_4				LB_3			
		Avg.	Max.	Avg.	Max.	Avg.	Max.	avg_nodes	max_nodes	avg_time	Unfound	avg_nodes	max_nodes	avg_time	Unfound
8	3	1.96	10.93	5.67	17.00	7.45	17.00	5073	29630	0.010	0	11735	81880	0.005	0
10	1	2.28	6.89	5.26	13.46	7.36	14.25	1.75E + 05	2.06E + 06	0.322	0	4.56E + 05	3.38E + 06	0.194	0
12	1	1.95	8.54	6.57	22.75	8.34	22.75	6.48E + 06	9.44E + 07	15.915	0	2.14E + 07	1.69E + 08	9.833	0
14	0	1.94	6.28	6.84	17.83	8.62	17.83	NA	NA	NA	18	NA	NA	NA	13
16	0	1.39	5.22	6.93	15.62	8.21	15.62	NA	NA	NA	46	NA	NA	NA	45

No precedence constraint.

Table 7
Buffer = 16, 000 KB

16, 000 KB	Deviation (%)													
n	$LB = UB$	UB vs. Opt		Opt vs. LB		UB vs. LB		LB_4				LB_3			
		Avg.	Max.	Avg.	Max.	Avg.	Max.	avg_nodes	max_nodes	avg_time	Unfound	avg_nodes	max_nodes	avg_time	Unfound
8	3	2.68	11.04	5.91	16.00	8.32	19.50	3376	16,248	0.007	0	7356	48,819	0.002	0
10	0	2.87	7.49	5.76	14.69	8.37	16.87	5.17E + 04	5.53E + 05	0.090	0	1.14E + 05	5.53E + 05	0.047	0
12	1	3.52	9.50	6.86	22.75	9.99	23.57	2.02E + 06	2.50E + 07	4.555	0	5.87E + 06	4.11E + 07	2.595	0
14	0	2.77	10.13	6.99	17.83	9.48	17.83	NA	NA	NA	10	NA	NA	NA	7
16	0	2.62	7.65	7.36	15.62	9.71	15.69	NA	NA	NA	36	NA	NA	NA	29
18	1	1.98	6.97	6.51	13.69	8.31	15.16	NA	NA	NA	46	NA	NA	NA	44

Precedence constraint = 20%.

Table 8
Buffer = 16, 000 KB

16, 000 KB	Deviation (%)													
n	$LB = UB$	UB vs. Opt		Opt vs. LB		UB vs. LB		LB_4				LB_3			
		Avg.	Max.	Avg.	Max.	Avg.	Max.	avg_nodes	max_nodes	avg_time	Unfound	avg_nodes	max_nodes	avg_time	Unfound
8	1	3.25	19.96	8.11	17.83	10.88	22.88	1007	2549	0.002	0	1747	5788	0.001	0
10	1	3.68	14.38	7.36	16.28	10.53	21.40	1.07E + 04	3.56E + 04	0.018	0	1.98E + 04	6.03E + 04	0.007	0
12	0	4.30	12.54	8.00	22.75	11.71	27.06	3.03E + 05	1.72E + 06	0.686	0	7.23E + 05	3.45E + 06	0.285	0
14	0	4.15	13.75	8.72	18.72	12.27	21.63	6.38E + 06	6.81E + 07	25.982	0	1.97E + 07	3.09E+08	11.254	0
16	0	4.08	15.29	8.75	14.60	12.24	21.16	NA	NA	NA	9	NA	NA	NA	6
18	0	4.33	14.78	7.35	17.44	11.12	25.39	NA	NA	NA	28	NA	NA	NA	20

Precedence constraint = 40%.

Table 9
Deviation between *UB* and *LB*

Precedence	0%			20%			40%		
Gap	*UB* vs. *LB*		*UB* = *LB*	*UB* vs. *LB*		*UB* = *LB*	*UB* vs. *LB*		*UB* = *LB*
Avg./Max.	Avg.	Max.	No.	Avg.	Max.	No.	Avg.	Max.	No.
20	7.919	14.143	0	9.847	17.582	0	12.773	21.930	0
50	6.191	12.985	0	7.957	13.739	0	10.594	16.068	0
100	4.412	6.653	0	6.173	8.409	0	9.342	12.675	0
200	3.632	6.052	0	5.364	7.760	0	8.731	11.246	0
300	3.300	4.821	0	4.944	6.905	0	8.028	9.459	0
500	2.811	4.349	0	4.249	5.584	0	7.433	8.728	0

Buffer = 16, 000 KB.

Table 10
Comparison of the NEH-based and the CDS-based heuristic solutions

N	Case 1: 1/100,000 CPU time allocated						Case 2: 1/1,000,000 CPU time allocated					
	16,000 KB (s)	NEH(UB)	CDS	30,720 KB (s)	NEH(UB)	CDS	16,000KB (s)	NEH	CDS	30,720KB (s)	NEH(UB)	CDS
10	CPU_time	35.40	0.85	CPU_time	27.90	0.60	CPU_time	304.00	7.80	CPU_time	365.70	11.80
	Makespan	732.18	841.66	Makespan	546.16	592.62	Makespan	732.18	841.66	Makespan	546.16	592.62
	Total time	767.58	842.51	Total time	574.06	593.22	Total time	1036.18	849.46	Total time	911.86	604.42
20	CPU_time	76.50	1.90	CPU_time	63.80	1.90	CPU_time	732.30	20.30	CPU_time	591.00	20.10
	Makespan	1460.72	1757.10	Makespan	1069.36	1252.24	Makespan	1460.72	1757.10	Makespan	1069.36	1252.24
	Total time	1537.22	1759.00	Total time	1133.16	1254.14	Total time	2193.02	1777.40	Total time	1660.36	1272.34
50	CPU_time	346.28	6.90	CPU_time	293.50	7.20	CPU_time	3399.00	72.00	CPU_time	2806.00	71.00
	Makespan	3592.92	4550.98	Makespan	2628.66	3290.94	Makespan	3592.92	4550.98	Makespan	2628.66	3290.94
	Total time	3939.20	4557.88	Total time	2922.16	3298.14	Total time	6991.92	4622.98	Total time	5434.66	3361.94

Quicktime, Real Player, etc. With these players and a standard web server, a prefetch-enabled TV-like presentation services can be realized using script languages or APIs supported by the players. Fig. 10 shows a snapshot of a prototype Flash-based implementation incorporating the object ordering algorithms developed in this paper.

6. Conclusion

In this paper, we mapped the media object scheduling problem aiming to minimize the presentation span to a conventional two-machine flowshop scheduling problem. Two constraints are considered in this paper, including a single chain precedence constraint and a player-side buffer constraint. In particular, the characteristics of the player-side buffer constraints have never been explored in previous flowshop scheduling problems. The player-side buffer constraint is termed as processing time-dependent buffer so as to distinguish it from the conventional item-based intermediate buffer in flowshop scheduling problems. We have proved that the studied problem is strongly NP-hard in Section 2.

Based on the characteristics associated with the precedence and the buffer constraints, we have proposed four lower bounds and an NEH-based upper bound for a branch and bound solution method. From the simulation experiments, the average gap between the overall lower bound and the NEH-based upper bound is less than 13% for problems with small buffer sizes and high density of precedence constraints, and 3% for problems with large buffer sizes. The performance of the proposed NEH-based heuristic solutions should be satisfactory for typical real time online multimedia applications. For applications where the media objects are delivered through extremely busy servers with

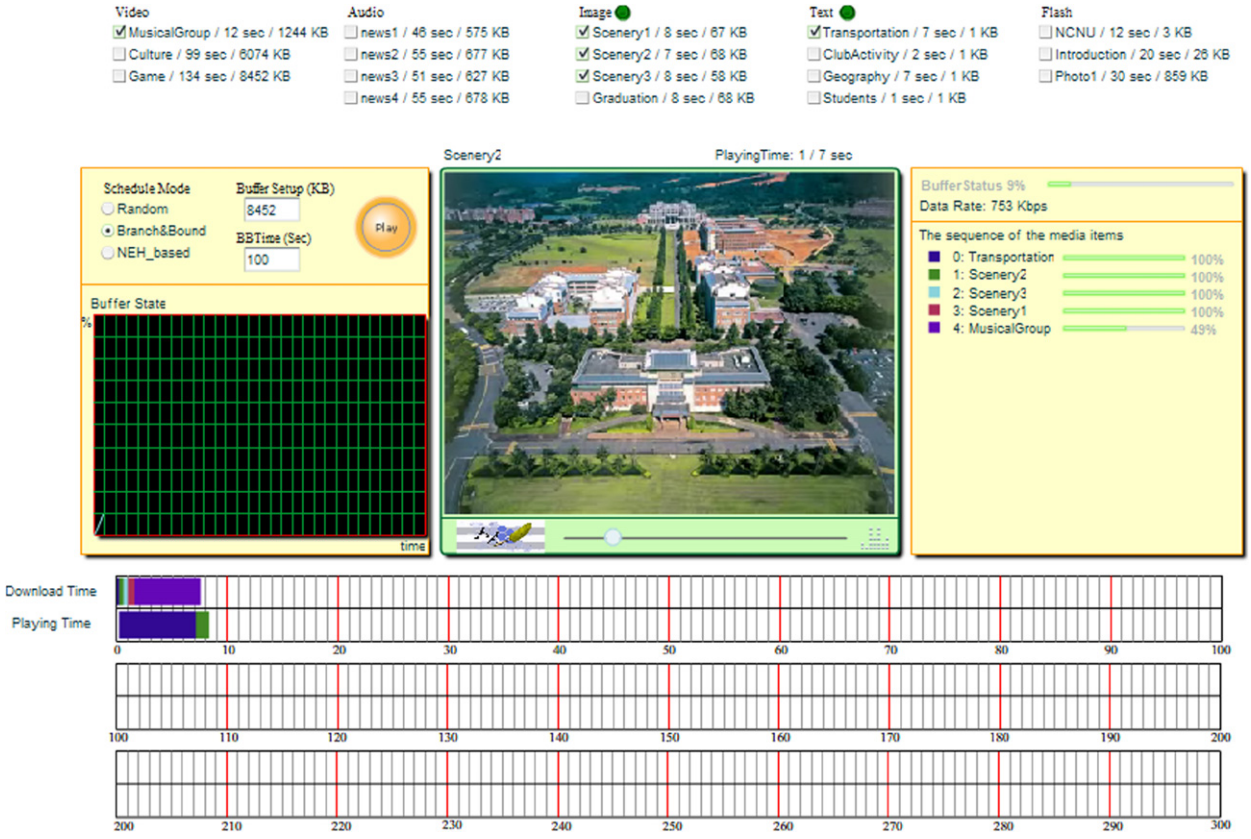

Fig. 10. Snapshot of a prototype prefetch-enabled media player that incorporates the scheduling algorithms discussed in this paper.

which only very restricted CPU resources can be allocated for computation, the CDS-based algorithm provides better sequences than the NEH-based algorithm.

Acknowledgments

The authors are grateful to the anonymous referees for their constructive comments that have improved the presentation of this paper.

References

[1] Brucker P. Scheduling algorithms. 3rd ed., Berlin: Springer; 2001.
[2] Strusevich VA. Shop scheduling problems under precedence constraints. Annals of Operations Research 1997;69(1):351–77.
[3] Gladky AA, Shafransky YM, Strusevich VA. Flow shop scheduling problems under machine-dependent precedence constraints. Journal of Combinatorial Optimization 2004;8(1):13–28.
[4] Lenstra J, Rinnooy Kan AHG, Brucker P. Complexity of machine scheduling problem. Annals of Discrete Mathematics 1977;1:343–62.
[5] Lenstra J, Rinnooy Kan AHG. Complexity of scheduling under precedence constraints. Operations Research 1978;26(1):22–35.
[6] Monma CL. Sequencing to minimize the maximum job cost. Operations Research 1980;28(4):942–51.
[7] Hariri AMA, Potts CN. Algorithms for two-machine flow-shop sequencing with precedence constraints. European Journal of Operational Research 1984;17(2):238–48.
[8] McMahon GB, Lim CJ. The two-machine flow shop problem with arbitrary precedence relations. European Journal of Operational Research 1993;64(2):249–57.
[9] Kurisu T. Two-machine scheduling under required precedence among jobs. Journal of the Operations Research Society of Japan 1976;19(1):1–13.
[10] Monma CL, Sidney JB. Sequencing with series-parallel precedence constraints. Mathematics of Operations Research 1979;4(3):215–24.
[11] Sidney JB. The two-machine maximum flow time problem with series parallel precedence relations. Operations Research 1979;27(4):782–91.

[12] Gordon VS, Shafransky YM. Optimal sequencing under series-parallel precedence constraints. Doklady Akademii Nauk BSSR 1978;22: 224–47 (in Russian).
[13] Valdes J, Tarjan RE, Lawler EL. The recognition of series parallel digraphs. SIAM Journal on Computing 1982;11(2):298–313.
[14] Papadimitriou CH, Kanellakis PC. Flowshop scheduling with limited temporary storage. Journal of the Association for Computing Machinery (JACM) 1980;27(3):533–49.
[15] Gilmore PC, Gomory RE. Sequencing a one state-variable machine: a solvable case of the traveling salesman problem. Operations Research 1964;12(5):655–79.
[16] Leisten R. Flowshop sequencing problems with limited buffer storage. International Journal of Production Research 1990;28(11):2085–100.
[17] Smutnicki C. A two-machine permutation flow shop scheduling problem with buffer. OR Spectrum 1998;20(4):229–35.
[18] Dutta SK, Cunningham AA. Sequencing two-machine flow-shops with finite intermediate storage. Management Science 1975;21(9):989–96.
[19] Brucker P, Heitmann S, Hurink J. Flow-shop problems with intermediate buffers. OR Spectrum 2003;25(4):549–74.
[20] Tang L, Xuan H. Lagrangian relaxation algorithms for real-time hybrid flowshop scheduling with finite intermediate buffers. Journal of the Operational Research Society 2006;57:316–24.
[21] Wang L, Zhang L, Zheng DZ. An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. Computers & Operations Research 2006;33(10):2960–71.
[22] Lin FC, Hong JS. Using Johnson's rule to optimize media object sequence for multimedia autoplay services. NCNU Technical Report TR20060005, 2006. ⟨http://tr.csie.ncnu.edu.tw/⟩.
[23] Johnson SM. Optimal two- and three-stage production schedules with setup time included. Naval Research Logistics Quarterly 1954;1:61–8.
[24] Allahverdi A, Al-Anzi FS. Using two-machine flowshop with maximum lateness objective to model multimedia data objects scheduling problem for WWW applications. Computers & Operations Research 2002;29(8):971–94.
[25] Batra P, Eleftheriadis A. A framework for optimal scheduling of structured and streaming media. CU/ADVENT Technical Report 2000-03, 2000. ⟨http://www.ee.columbia.edu/∼pbatra/tr00.pdf⟩.
[26] Blazewicz J, Dell'Olmo P, Drozdowski M. Scheduling of client-server applications. International Transactions in Operational Research 1999;6(4):345–63.
[27] Martello S, Toth P. Knapsack problems: algorithms and computer implementations. Chichester: Wiley; 1990.
[28] Nawaz M, Enscore E, Ham I. A heuristic algorithm for the *m*-machine, *n*-job flow shop sequencing problem. OMEGA 1983;11:91–5.
[29] Watson JP, Barbulescu L, Howe AE, Whitley LD. Algorithm performance and problem structure for flow-shop scheduling. In: The 16th national conference on artificial intelligence (AAAI-99) 1999. p. 688–95.
[30] Agarwal A, Colak S, Eryarsoy E. Improvement heuristic for the flow-shop scheduling problem: an adaptive-learning approach. European Journal of Operational Research 2006;169(3):801–15.
[31] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness. San Franciso, CA: Freeman; 1979.
[32] Campbell HG, Dudek RA, Smith ML. A heuristic algorithm for *n* job, *m* machine sequencing problem. Management Science 1970;16(10): 630–7.