

Retrieval schedules based on resource availability and flexible presentation specifications

K. Selçuk Candan¹, B. Prabhakaran², V.S. Subrahmanian³

¹ Department of Computer Science and Engineering, Arizona State University, Box 875406, Tempe, AZ 85287-5406, USA; e-mail: candan@asu.edu

² Department of Information Systems and Computer Science, National University of Singapore, Singapore 119260; e-mail: prabha@iscs.nus.edu.sg

³ Department of Computer Science, Institute for Advanced Computer Studies & Institute for Systems Research, University of Maryland, College Park, MD 20742, USA; e-mail: vs@cs.umd.edu

Abstract. A distributed multimedia document presentation involves retrieval of objects from one or more document servers and their presentation at the client system. The presentation of the multimedia objects has to be carried out in accordance with the specification of temporal relationships between the objects. The retrieval of multimedia objects from the document server(s) is influenced by factors such as temporal specification of objects presentations, throughput offered by the network service provider, and the buffer resources on the client system. Flexibility in the temporal specification of the multimedia document may help in deriving an object retrieval schedule that can handle variations in network throughput and buffer resource availability. In this paper, we develop techniques for deriving a flexible object retrieval schedule for a distributed multimedia document presentation. The schedule is based on flexible temporal specification of the multimedia document using the difference constraints approach. We show how the derived retrieval schedule can be validated and modified to ensure that it can work with the offered network throughput and the available buffer resources.

1 Introduction

A multimedia document consists of different types of media objects that are to be presented at different instants of time for different durations. The time instances and durations of presentations of the objects are specified as either hard or flexible temporal specifications. In the case of hard temporal specification, the time instants and durations of presentations of objects are fixed. In a flexible temporal specification, however, the time instants and durations of presentations of objects are allowed to vary as long as they preserve certain specified relationships. To see this, consider two temporal constraint specifications of the form

This work was supported by the Army Research Office under Grants DAAH-04-95-10174, DAAH-04-96-10297, and DAAH04-96-1-0398, by the Army Research Laboratory under contract number DAAL01-97-K0135, and by an NSF Young Investigator award IRI-93-57756.

Correspondence to: K.S. Candan

- (a) Start showing the image *at* 10 a.m. *for* 10 min.
- (b) Start showing the image *sometime between* 9:58 a.m. and 10:03 a.m. and show it till the audio is played out.

The first statement is a *hard* temporal specification, with the time instant and duration of presentation of the image fixed (at 10 a.m. and for 10 min, respectively). In contrast, the second specification is more flexible in that it allows the start time instant to vary within a range of 5 min. A similar flexibility is allowed for the duration of presentation of the object also, by showing the image till the audio is played out. The temporal constraint specification, in other words, helps in the derivation of a *presentation schedule* that describes the starting times and durations of the presentations of the objects composing the multimedia document. Note that, if the specifications are hard, the presentation schedule would be the same as the temporal constraint specification. Flexible temporal specifications imply that there may be many options in the presentation of the multimedia document, i.e., one can have a *set* of presentation schedules that satisfy the given temporal constraints. Each member of this presentation schedule set describes one possible *view* of the multimedia document. In this work, we deal with flexible temporal constraints.

In a distributed multimedia presentation, the objects composing the document may be dispersed over a computer network. These objects have to be retrieved from their storage sites and presented to the user. With the storage place acting as a *server* and the retrieving system as a *client*, the retrieval process is initiated by the client (as opposed to the server just *delivering* the objects following some schedule of its own). Hence, the retrieval process is composed of the following phases.

- Identify a *presentation schedule* that satisfies the (flexible) temporal specification associated with the multimedia document.
- Identify a *retrieval schedule* that specifies the time instants at which the client should make a request to the server(s) for delivering the objects that compose the multimedia document.

Specification of the time instants for retrieving objects from the server as part of the retrieval schedule is carried out

by determining the time taken to transfer the object from the server to the client. Consider the temporal constraint specification (b). We can derive a presentation schedule that specifies the start time of presentation of object A as 9:58 a.m. If we know that the delay involved in retrieving object A from its server is 3 min, then the retrieval schedule can be fixed at 9:55 a.m. This retrieval schedule is constrained by the following factors

1. Throughput (or the bandwidth) of the communication channel between the server and the client.
2. Buffer availability for the retrieved objects.
3. Size of the object(s) that is (are) to be retrieved from the server.
4. Time duration available for retrieval.

Here, the throughput of the communication channel, and the buffer resources are system dependent. The available throughput can vary depending on the type of network and the load on the network. The buffer resources are dependent on their availability in the client system. The last two constraints, sizes of the objects and the times available for retrieval, are application dependent. Size of an object depends on the type of media, as well as on the desired *quality* of presentation (as discussed in Sect. 2.2). For example, an image object may be retrieved as a thumbnail sketch or as a full image. The time available for presentation depends on the derived presentation schedule from the (flexible) temporal constraints specification. The retrieval schedule for a multimedia document presentation has to be derived based on the above four constraints.

1.1 Related work

The use of distributed multimedia information, especially for distributed collaborations [10, 11, 14, 32] and application sharing [39, 40], is becoming increasingly necessary. The related research issues include data retrieval [15], delivery of media objects over a network [3, 4, 8, 9, 12, 29, 30], and temporal reasoning and media synchronization [1, 25, 27, 33, 36, 37]. Also, multimedia authoring and presentation schedule creation have been studied by many researchers [2, 6, 16, 17, 19, 20, 21, 24, 26, 34, 38]. Similarly, derivation of retrieval schedules for distributed multimedia presentation has also been studied in many works, such as [17, 19, 28, 29, 30, 31, 35]. In [19], the presentation schedule is based on Petri nets' description of the temporal specification. This presentation schedule is fixed before the generation of the retrieval schedule. The retrieval schedule is derived by assuming a certain throughput to be provided by the network service provider. Based on the derived retrieval schedule and the assumed network throughput, estimates for the buffer resource requirements on the client system are made. However, the proposed algorithm does not check whether the estimated buffer resources are available or not. Also, it does not handle the variations in the throughput offered by the network service provider. Li et al. [17] use time-flow graphs to capture interval-based *fuzzy* presentation schedules, and synchronization of independent sources. Their algorithms guarantee that there will be no gaps in the source's schedules. However, they do not address the issue

of constraints on resources such as throughput and buffer. As in [19], in [28, 29, 30, 31], authors use Petri net models to describe temporal specifications, and they base the retrieval schedules on the fixed presentation schedules. Thimm et al. [35] describe a method which adapts the presentation schedule to the changes in the resource availability by modifying the overall quality of the presentation.

1.2 Our approach

In this paper, we have developed techniques for deriving flexible object presentation and retrieval schedules for a distributed multimedia document presentation. Our approach is to use flexible temporal constraint specifications for deriving a possible presentation schedule [5, 6]. Based on this presentation schedule, we suggest techniques for deriving the retrieval schedule. The derived retrieval schedule is validated by checking whether it satisfies all the system availability constraints. If it does not, we do the following.

1. Modify the retrieval schedule. We try to find a different retrieval schedule which fits into the presentation schedule at hand.
2. If no such change in the retrieval schedule is possible, then modify the presentation schedule. We identify the portions of the retrieval schedule which do not satisfy the system resource availabilities. Based on the unsatisfied retrieval schedule, we suggest a feedback mechanism for modifying the presentation schedule appropriately.
3. If everything else fails, then modify the quality of presentation. Such a modification is possible in the case of certain objects such as *gif* images, etc. Also, the reduced quality of presentation should be acceptable to the viewer.

2 Multimedia document presentation

A multimedia document is composed of objects that are to be presented at different time instances and for different durations. Based on the way the objects are to be retrieved from the server(s), we classify them as

- **Atomic Objects.** These objects need to be received as a whole at the client side before the presentation starts. For example, still-image files are atomic objects.
- **Stream Objects.** These objects can be presented to the viewer as soon as some portion of them is received. The rest of the object is then continuously fed to the viewer. Video and audio objects are generally considered to be stream objects. However, in systems which cannot handle display-while-retrieving operation, video can be retrieved as a whole, and be displayed afterwards. In such systems, video objects must be considered atomic objects.

Atomic objects must be present in the buffers of the client as a whole at the start of their presentations. They can then be consumed from the buffers during the presentation (as in the case of atomic video objects) or can be kept in the buffers as a whole till the end of the presentation (as

in the case of the still images). We use $C_a(o)$ to denote the consumption rate of an atomic object o from the buffers. Note that, for objects like still images, $C_a(o) = 0$.

Stream objects do not need to be delivered as a whole before their presentations. However, in order to reduce jitter and to smooth their display, such objects usually require some fraction to arrive at the display site before the start of their presentations. In this paper, we use $b_{init}(o)$ to denote the size of this fraction for a stream object o , and we use $C_s(o)$ to denote the consumption rate of a stream object o from the buffers. Note that, in order to prevent the underflow of the buffer $B(o)$, the consumption rate $C_s(o)$ must be equal to the average delivery rate (throughput) of the object o (throughput of an object o is denoted as $th(o)$). However, if the network is not capable of providing the consumption rate, then we can reduce the throughput requirement by buffering a larger portion of the object at the client site. This process will be explained in more detail in Sect. 7.1.

2.1 Flexible multimedia presentation

In practical circumstances, one may encounter a situation where the derived retrieval schedule cannot be satisfied. For example, the network service provider might offer a very low throughput for the application. The retrieval schedule based on the throughput offered by the network service provider may overshoot the buffer availability on the client. Hence, the derived retrieval schedule cannot be used for the multimedia presentation. However, we may be able to modify the retrieval schedule by relaxing (one or both of) the application-dependent constraints. For example, one can reduce the size of the multimedia objects to be retrieved by reducing the quality of the presentation (if the reduced quality is acceptable to the viewer). If the reduction in the quality of the presentation is not acceptable, then we can modify the duration for retrieving the objects. This can be performed by selecting a different presentation schedule that satisfies the given set of (flexible) temporal constraints.

As an example, consider the (previously described) temporal constraint specification (b): start showing the image sometime between 9:58 a.m. and 10:03 a.m. and show it till the audio is played out. Let the chosen presentation schedule be such that the image presentation starts at 9:58 a.m. If we find that the retrieval schedule based on this presentation schedule does not satisfy all the system constraints, then we can try another presentation schedule, say, image presentation starts at 10:03 a.m. This change in the presentation schedule gives more time for the retrieval of the object (in this example, it gives 5 min more for retrieval). We can derive a new retrieval schedule based on the modified presentation schedule and check whether the new retrieval schedule satisfies all the constraints.

In the above example, we (seem to have) arbitrarily picked up another value for the start of presentation of the image. Instead, we can use feedback from the retrieval schedule to give us an idea of which temporal values can be changed by what factor. In other words, *we can use the flexibility of the multimedia presentation to find other solutions within the application-dependent constraints, such as the quality of presentation and the presentation schedule.*

| Media Type | Reduction in size | Reduction in quality |
|------------|-------------------|----------------------|
| A | 0.2 | 0.1 |
| A | 0.5 | 0.7 |
| B | 0.1 | 0.8 |
| ... | ... | ... |

Fig. 1. An example table relating quality with reduction in object size

This new solution for the retrieval schedule is selected in such a way that it can handle the changes in the system-dependent constraints such as throughput offered by the network and the buffer resources available on the client.

2.2 Quality of the presentation

In this paper, we suggest the use of quality reduction as an option for satisfying the system resources. Note that this option is used only when there is no other way to satisfy the constraints.

The quality of a presentation has two main aspects.

- **Response time of the presentation.** The response time is defined as the amount of time that elapses between the time at which the first object request is issued by the system and the time at which the presentation starts. The smaller this value is, the higher the quality of the presentation.
- **Quality of objects.** In this paper, we assume that a quality is associated with each object presentation, and a reduction in the size of an object is accompanied by a reduction in its quality. Different media types observe different quality reductions when their size is reduced. A table of the form shown in Fig. 1 provides a means by which a system can possibly relate quality with reduction in object size. Note that a reduction in the quality of an object causes a reduction in the quality of the overall presentation. Hence, it is better to use higher quality objects as long as their sizes do not violate system constraints.

2.3 Object priorities

When there are two or more objects competing for the same system resource, it may be necessary to change the schedules or reduce the qualities of their presentations. Note that, if objects o_1 and o_2 are competing for a resource, then o_1 , o_2 , or both may be affected if the available resources are inadequate to serve them both. We assign priorities to the objects in a presentation to minimize the number of objects that will be affected from a resource shortage. The priority of an object o is calculated using user preferences, the number of other objects whose presentations depend on o , and cost of modifying the quality of o .

3 Flexible multimedia presentation architecture

Figure 2 shows the architecture we propose for a flexible multimedia presentation. The temporal specifications associated with a given multimedia document are used by the Presentation Schedule Generator module to generate a possible presentation schedule. Based on this schedule and other

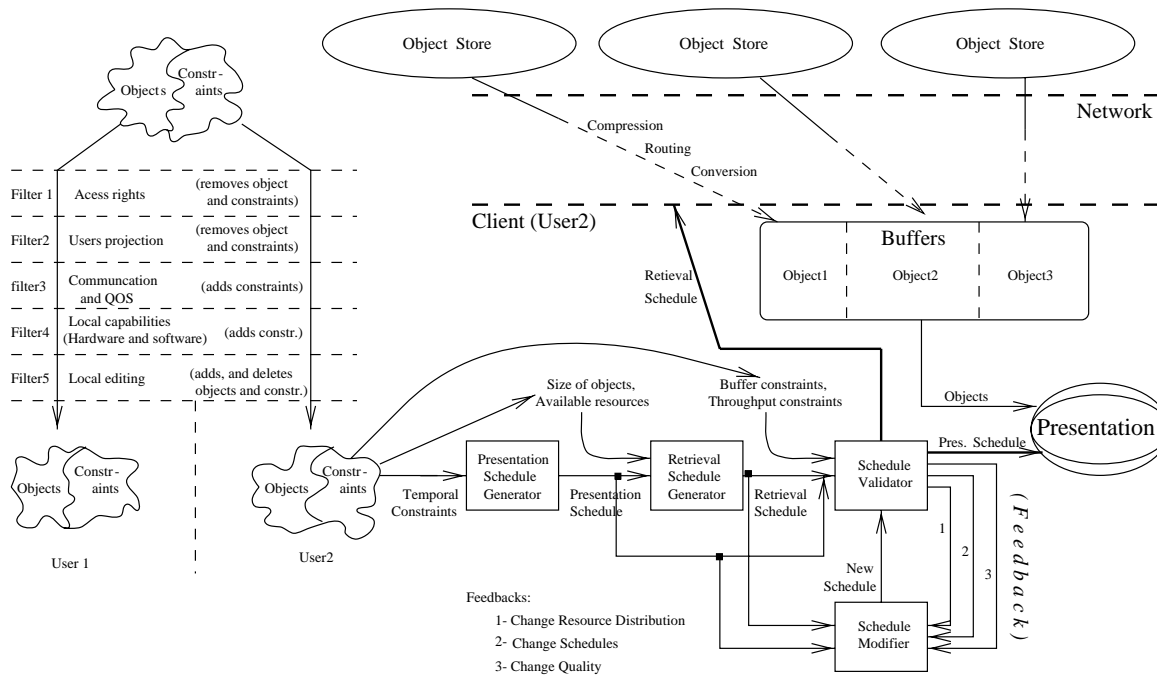


Fig. 2. Flexible multimedia presentation architecture

constraints such as available throughput and the size of the objects, the retrieval schedule generator determines a possible retrieval schedule. The schedules are checked by the schedule validator to determine whether all the associated constraints (buffer availability, throughput) are satisfied. If some of the constraints are not satisfied, one or more of the following modifications are made in order to satisfy all the constraints.

- Change the buffer resource distribution.
- Pick a different presentation or retrieval schedule.
- Change the quality of the presentation

In the above modifications, modifying the buffer resource distribution is most desirable, while changing the quality of presentation is least desirable. Based on this discussion, we can say that the input and output of the flexible multimedia presentation system are as follows.

Input. The input to the system consists of

- a set of temporal specifications,
- a list of available system resources: throughput and buffer,
- a list of object sizes and object locations,
- a list of presentation quality requirements,
- a list of object priorities.

Output. The output is a presentation and a retrieval schedule that satisfies the input specifications, and requirements.

Flexible Multimedia Presentation System Components.

1. **Filters for Accessing the Multimedia Document.** Depending on their interests and their needs, authors editing a document may wish to *view* a document in different ways, through the use of *filters*. These filters describe the specific portions of the multimedia document that can be

viewed or edited by an author/user. The filters that can possibly be applied to a document depend on

- user access rights,
- local system capabilities, and
- user's projection of the document.

Here, the user access rights and local system capabilities are system-defined filters. Access rights describe the portions of the multimedia document accessible to the user. Local system capabilities, on the other hand, describe the facilities the user has for multimedia object handling. For example, if the local system cannot handle MPEG video streams, the author might filter out MPEG video objects and view the rest of the document. The users' projection of the document is a user-defined filter that helps the author to access the portion of the document that they are interested in. The presentation must adapt itself to the changes, such as to the omissions or additions of new objects. Authors can perform certain edit operations that modifies only the local view of the multimedia document. When they are satisfied with the edit operations they have performed, the changes can be realized on the system's document view.

2. **Presentation Schedule Generator.** This component of the system provides a solution to the given temporal specifications. It picks a schedule which satisfies the temporal specifications. We provide an overview of the temporal constraint solver in Sect. 4.
3. **Retrieval Schedule Generator.** This module takes as input the presentation schedule, the list of available system resources, and the object sizes. It outputs a retrieval schedule. Section 5 describes how the retrieval schedule is generated from the listed inputs.
4. **Schedule Validator.** Given a temporal schedule, system constraints, and a retrieval schedule, this module checks the validity of the generated retrieval schedule based on

the input constraints. If the schedules are valid with respect to the specified constraints, then the *validator* returns them as the final solution. However, if the schedules do not satisfy the system constraints, this module suggests modifications that can be made to the current solution in order to satisfy the system constraints. These suggestions are used by the *Schedule Modifier* module to find a modified solution that can satisfy the constraints. Section 6 describes in detail the functionality of the Schedule Validator module.

5. **Schedule Modifier.** This module modifies the current solution for retrieval and presentation schedules, based on the suggestions made by the Schedule Validator module. The modified solution is given back to the Validator module to check the solution against the system constraints. We discuss the details of the Schedule Modifier in Sect. 7.3.

This process of solution-feedback and validation is repeated till a valid schedule is generated. In case a valid schedule cannot be arrived at, then the best schedule found so far can be used as the solution. Objects whose schedules do not satisfy the system constraints can be dropped from the presentation, provided the viewer agrees.

3.1 Segmented validation of schedules

In our approach, we first generate a presentation schedule based on the specified temporal constraints. Then we generate a retrieval schedule for a *segment of time* (the duration of the time segment is chosen based on the implementation requirements). The process of segmented retrieval schedule generation (and validation) is done for the following reasons.

- In the case of a long presentation (say, a 1-h presentation), system constraints such as throughput and buffer can vary considerably. Hence, an initial schedule generated for the entire presentation may become invalid at a later point in time.
- Creating a complete schedule for the whole presentation can be time consuming. It might lead to a long wait time for the user before the presentation can start. (And after all that, the schedules might become invalid!)

The retrieval schedule for the time segment is then validated with respect to the system constraints. If the retrieval schedule is found valid, then the multimedia document presentation for the validated time segment is started. (Otherwise, we go through the process of solution-feedback to generate another schedule, as discussed above). Then, the retrieval schedule for the next time segment is generated and validated. Hence, we follow a *segmented validation* of the generated schedules.

It should be noted here that the segmentation of the presentation is done with respect to the presentation schedules. The retrieval schedule for a segment might fall into the previous segment. This can modify the throughput and buffer requirements. In our approach, we allow these segments to overlap by a chosen duration, say t . For example, if SE_{i-1} denotes the end of segment $i-1$, then SS_i , the segment start time of the i^{th} segment will be: $SS_i = SE_{i-1} - t$.

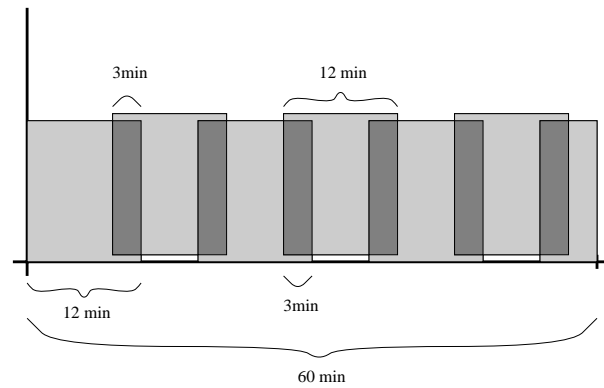


Fig. 3. Segmented validation: an example

| Symbol | Meaning |
|----------------|---|
| Th_{max} | Maximum throughput available at a communication line. |
| Th_{tot} | The total throughput required by the objects sharing a communication line. |
| $th(o)$ | The amount of throughput used by object o . |
| Buf_{max} | Maximum buffer available for the objects sharing a communication line. |
| $Buf_{tot}(t)$ | The total buffer required at time t by the objects sharing a communication line. |
| $buf(o, t)$ | The amount of buffer used by object o at time t . |
| $b_{init}(o)$ | The size of the buffer required by object o before the start of its presentation. |
| $st(o)$ | The time at which the display of object o starts. |
| $et(o)$ | The time at which the display of object o ends. |
| $req(o)$ | The time at which the request for the object o is issued by the client. |
| $rec(o)$ | The time at which the first bit of the object o is received at the client. |
| $sz(o)$ | The size of the object o . |

Fig. 4. Notation and terminology

While validating one segment, its overlap with the previous segment is *revalidated*. Hence, this takes care of the overlap of the retrieval schedules.

Figure 3 shows an example of this segmented validation of a 1-h presentation. The document is divided into five segments of size 12 min each, and each of the segments are handled separately. The consecutive segments have an overlap of 3 min: the schedule of the last 3 min of segments are reprocessed at the beginning of the following segments, as discussed above.

3.2 Notation used in the paper

The major symbols we use in this paper are explained in Fig. 4.

4 Presentation schedule generation

Our approach for temporal specification is to use a small class of the language of real-valued linear constraints called *difference constraints*. While generalized linear constraints [13] have the form

$$a_1x_1 + a_2x_2 + \dots + a_nx_n \leq b, \quad (1)$$

where a_1, \dots, a_n, b are *rational numbers* (positive and negative), and x_1, \dots, x_n range over the *real numbers* (positive and negative), difference constraints have the form

$$x_1 - x_2 \leq b. \quad (2)$$

Thus, difference constraints are a special case of linear constraints, where

1. there are only two variables (i.e., $n = 2$ in Eq. 1), and
2. one variable has coefficient 1 (i.e., $a_1 = 1$), while the other has coefficient -1 (i.e., $a_2 = -1$).

With each object O in a multimedia document D , we associate a set, T_O , of temporal constraints. As is customary in operations research [13], constraints are constructed from *variables*. In the case of multimedia documents, we associate with each multimedia object O in the document the following *temporal variables*:

- $st(O)$: denotes the start time of the display of the object O ,
- $et(O)$: denotes the end time of the display of the object O

Later, in Sect. 5, we will associate some other variables with objects, but for now, these two types of variables are adequate for specifying presentation constraints. There are four types of temporal presentation constraints:

- $\mathcal{T}(o) - t \leq \delta t$ • $\mathcal{T}(o) - t \geq \delta t$
- $t - \mathcal{T}(o) \leq \delta t$ • $t - \mathcal{T}(o) \geq \delta t$

where

1. $\mathcal{T}(o) \in \{st(o), et(o)\}$ and
2. $t \in \bigcup_j \{et(o_j), et(o_j)\} \cup \{st_p, et_p\}$ and
3. st_p and et_p denote the start and end of the presentation, respectively.

Example 4.1. Let us assume that there exist two objects o_1 and o_2 that we want to display simultaneously, i.e., we want them to start and finish simultaneously. This requirement can be described using the following constraints:

$$\begin{aligned} st(o_1) - st(o_2) &\leq 0, \\ st(o_2) - st(o_1) &\leq 0, \\ et(o_1) - et(o_2) &\leq 0, \\ et(o_2) - et(o_1) &\leq 0. \end{aligned}$$

Note that using these constraints, not only can we specify Allen's 13 temporal relationships [1] between events, but also specify more complex quantitative relationships that cannot be expressed in Allen's framework. Suppose D is any document and T_D is the set of temporal constraints associated with D . With this set of difference constraints, we may associate a graph $G = (V, E)$ defined as follows.

1. **Vertices.** For each constraint variable τ_i occurring in the set of difference constraints T_D , V contains a vertex v_i representing that variable. In addition, V contains two special vertices v_s (document "start" node) and v_e (document "end" node).
2. **Edges.** If $\tau_j - \tau_i \leq \delta t$ is a constraint in the set of difference constraints being considered, then E contains an edge from v_i to v_j and the weight associated with this

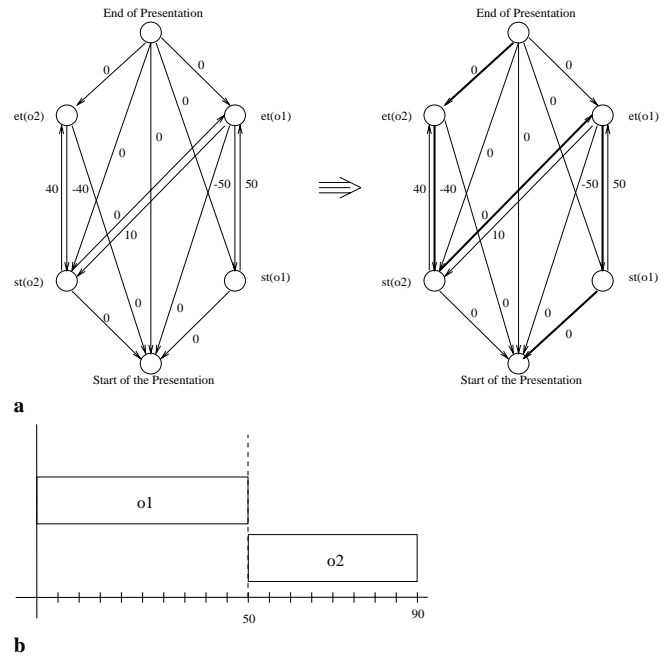


Fig. 5a. The constraint graph and **b** the corresponding solution

edge is δt . Furthermore, for each node v_i , there is an edge from v_i to v_s with weight 0 and from v_e to v_i with weight 0.

Thus, given any document D , we have one graph associated with its temporal specifications. The shortest path solution of this graph (from the end of presentation node to the start of presentation node) results in a schedule that satisfies the temporal specification. Example 4.2 shows how the solution works.

Example 4.2. Let us assume that there exist two objects o_1 (50 s) and o_2 (40 s). We want to display one after the other (i.e., o_2 after o_1). But, we also want the display of o_2 to start within 10 s after o_1 finishes. This requirement is captured using the following constraints:

$$\begin{aligned} st(o_1) - et(o_1) &\leq -50, \\ et(o_1) - st(o_1) &\leq 50, \\ st(o_2) - et(o_2) &\leq -40, \\ et(o_2) - st(o_2) &\leq 40, \\ et(o_1) - st(o_2) &\leq 0, \\ st(o_2) - et(o_1) &\leq 10. \end{aligned}$$

Figure 5 shows the corresponding graph, and the corresponding shortest path solution. At the end, the vertices of the graph have the following values:

$$\begin{array}{ll} \text{End of the presentation} = 0 & \text{Start of the presentation} = -90 \\ et(o_2) = 0 & st(o_2) = -40 \\ et(o_1) = -40 & st(o_1) = -90 \end{array}$$

The corresponding presentation schedule is also shown in Fig. 5. Note that the time values in the schedule are shifted by 90 to make them positive.

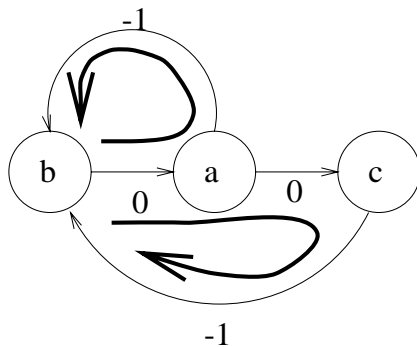


Fig. 6. A constraint graph with a negative cycle

4.1 Solving difference constraints

In general, the number of constraints in the graph is linear in the number of temporal specifications provided by the multimedia authors. It is well known that solving a set of difference constraints is equivalent to finding the shortest path in the graph associated with those constraints [7]. The constraint graphs may contain edges with negative weights. As an example, consider the following constraint specification.

Example 2. Assume the following set of specifications:

$$\begin{aligned} (1a) \quad a - b &\leq 0 & (2a) \quad b - a &\leq -1 \\ (3a) \quad c - a &\leq 0 & (4a) \quad b - c &\leq -1 \end{aligned}$$

Here, (1a)-(2a), and (1a)-(3a)-(4a) are in conflict. The best way to handle this problem is to remove (1a), because both conflicts will be resolved by the deletion of a single constraint. However, any other solution would include at least two deletions, such as the removal of (2a) and (3a), which is undesirable.

Such conflicting constraints are captured by the existence of negative cycles in the constraint graph. A negative cycle is one where sum of the weights assigned to the edges in the cycle is less than zero. For instance, the set of specifications in Example 2 corresponds to the graph in Fig. 6. The elimination of conflicting constraints requires that some constraints be weakened or eliminated. Three ways of eliminating constraints are presented below, all of which are based on a common (and simple) concept of constraint removal.

Definition 1 (Constraint removal). Suppose C is a set of difference constraints. A removal of C is any subset $C' \subseteq C$ such that C' is solvable.

The above definition allows any subset of C to be considered a removal. Thus, if c_1, c_2 are constraints in C , and if $(C - \{c_1\})$ is solvable, then it must necessarily be the case that $(C - \{c_1, c_2\})$ is solvable. However, the latter removal of C eliminates “more constraints” than strictly needed. Below, we present three alternative definitions of *optimal constraint removal*. The third definition assumes that each constraint has an associated *priority* – a number greater than or equal to 1. The higher the priority, the more important the constraint.

Definition 2 (Optimal constraint removal). Suppose C is a set of difference constraints.

- a card-optimal removal of C is any subset $C' \subseteq C$ such that C' is solvable, and there is no other removal C'' such that $\text{card}(C'') > \text{card}(C')$. Here, $\text{card}(C)$ is the number of constraints in C .
- a set-optimal removal of C is any subset $C' \subseteq C$ such that C' is solvable, and there is no other removal C'' such that $C'' \supset C'$.
- a priority-sum-optimal removal of C is a removal C' of C such that there is no other removal C'' which satisfies $(\sum_{c \in C''} \wp(c)) > (\sum_{c \in C'} \wp(c))$ where $\wp(c)$ denotes the priority of constraint c . We assume that $\wp(c) \geq 1$ for all constraints c .
- a priority-optimal removal of C is a removal C' of C such that if c is in $C - C'$, then there exists a set of conflicting constraints S in C where for all c_i in S , $\wp(c_i) \geq \wp(c)$.

Finding card-optimal and priority-sum-optimal removals of C is NP-complete, while finding a set-optimal removal is solvable in polynomial time. Similarly, finding priority-optimal removal is solvable in polynomial time.

Presentation schedule generation algorithms. Though most graph algorithms for computing shortest paths cannot handle negative edges in the graph, the well-known Bellman-Ford shortest path algorithm can deal with negative edges [7]. If there is no negative cycle, the algorithm produces the shortest paths and their weights. The shortest path along with the associated weights specify, in effect, the start times and durations of presentations of the objects composing the multimedia document. If there is such a cycle, the algorithm terminates indicating that there is no solution. The presence of a negative cycle indicates conflicting constraints. However, the Bellman-Ford algorithm cannot remove constraints so as to restore solvability. Also, local editing of a multimedia document necessitates the handling of incremental additions and deletions of constraints. The reason for this is that every time a document is modified, the set of associated constraints changes. Insertion of a new object into the presentation causes insertion of new constraints, while object deletion causes deletion of existing constraints. Object modification may cause both insertions and deletions of constraints. Hence, we have developed a set of algorithms (see Appendix) that will do the following.

- Take as input a set of temporal constraints and return as output a presentation schedule that satisfies as many constraints as possible.
- Handle incremental additions and deletions of constraints such that a maximal set of constraints is satisfied at all times.

All these algorithms work in polynomial time, and the incremental addition/deletion algorithms are easily seen to perform significantly better than re-solving the constraints from scratch.

5 Retrieval schedule generation

The retrieval schedule associated with a multimedia presentation specifies the time instants at which the client should make requests to the server(s) for delivering the objects that compose the presentation. As discussed earlier, the retrieval

schedule is constrained by system-dependent factors, such as the available throughput and available buffer resources, as well as by application-dependent factors, such as the time available for retrieval and the size of the objects. While deriving the retrieval schedule, we make the following assumptions.

- Multiple objects can be retrieved over the same network connection.
- The network provides a maximum throughput Th_{max} for each connection. Hence, this available throughput has to be shared by different objects in case their retrieval from the server has to be done in parallel. This throughput offered by the network service provider can vary with time, depending on the network load.
- The client provides a maximum buffer Buf_{max} for each connection for storing the retrieved objects before their presentation.
- The release of the buffer resources associated with the object presentation depends on the application, as well as on the media type to which the object belongs. The resources can be released, in the earliest case, once the object presentation is started (for media types such as still images).

Based on these assumptions, we now discuss the idea behind our approach for determining the retrieval schedule.

Single object retrieval. As the simplest case, let us consider the retrieval of a single object as shown in Fig. 7a. The object O has to be presented by the client at time $st(O)$. If the object is atomic, then the retrieval of the object has to be completed before $st(O)$. The client makes a request at time $req(O)$ to the server for the transfer of the object ($req(O)$ must be before $st(O)$). Here, $req(O)$ depends on the time required for transferring the object from the server to the client and on the round trip time required for sending the request to the server and receiving a response. Hence, for the case of single atomic object retrieval, $req(O)$ can be defined as:

$$req(O) = st(O) - \left\{ \frac{sz(O)}{Th_{max}} + \Delta t \right\}, \text{ and}$$

$$rec(O) = req(O) + \Delta t, \text{ where,}$$

Δt is the roundtrip propagation time for sending the request and receiving a response, and $rec(O)$ denotes the time at which the client receives the first byte of the request. Based on the above discussion, the system-dependent factors such as throughput and buffer requirements during a time interval for single atomic object retrieval are easy to capture:

$$\left. \begin{aligned} th(O, t) &= Th_{max} \\ buf(O, t) &= 0 && \{ \text{if } t < rec(O) \} \\ buf(O, t) &= Th_{max} \\ &\quad \times (t - rec(O)) && \{ \text{if } rec(O) \leq t \leq (st(O)) \} \end{aligned} \right\}$$

in interval $[req(O), st(O)]$

In the above equations, t varies between $req(O)$ and $st(O)$. When t is equal to $req(O)$, the buffer requirement is 0, whereas when t becomes equal to $st(O)$, the amount of buffer required for the object O rises to $sz(O)$.

In the case of O being a stream object, a chunk of frames (shown by the shaded portion in Fig. 7a) is to be retrieved

before the start of the presentation. The size of this chunk depends on the type of the media, the jitter requirement, and the display hardware at the client. The time $req(O)$ at which the client has to make a request to the server for transferring the object follows the same argument as in the case of an atomic object, except that the size of the object $sz(O)$ has to be replaced with the size of the chunk of frames to be retrieved $sz(O_{chunk})$. Hence,

$$req(O) = st(O) - \left\{ \frac{sz(O_{chunk})}{Th_{max}} + \Delta t \right\}.$$

However, the throughput and the buffer requirements are different from that for an atomic object:

$$\left. \begin{aligned} th(O, t) &= Th_{max} \\ buf(O, t) &= 0 && \{ t < rec(O) \} \\ buf(O, t) &= Th_{max} \times (t - rec(O)) && \{ rec(O) \leq t \leq (st(o)) \} \end{aligned} \right\}$$

in interval $[req(O), st(O)]$

$$\left. \begin{aligned} th(O, t) &= c(O) \\ buf(O, t) &= sz(O_{chunk}) \end{aligned} \right\} \text{ in interval } [st(O), et(O)]$$

Here again, t varies between $req(O)$ and $st(O)$.

Figure 7b shows the buffer requirements for a single object (atomic and stream) retrieval. Buffer requirements of an atomic object are basically in the time interval: $[req(O), st(O)]$, whereas the requirements of stream objects are distributed in the time interval: $[req(O), et(O)]$.

Parallel, multiple object retrieval. In many cases, the presentations of multiple objects that are to be retrieved over the same network connection can overlap, as shown in Fig. 8. In this case, the available throughput and buffer resources have to be shared among the objects to be retrieved. For instance, the stream objects in Fig. 8a are initially assigned the following throughputs:

$$\begin{aligned} th(O1) &= Th_{max}/2, \\ th(O2) &= Th_{max}/2, \\ th(O3) &= Th_{max}/3, \\ th(O4) &= Th_{max}/3, \\ th(O5) &= Th_{max}/3, \\ th(O6) &= Th_{max}/2. \end{aligned}$$

Each object O is assigned a throughput of Th_{max}/n , where n is the maximum number of objects that simultaneously overlap during the presentation time of O (i.e., from $st(O)$ to $et(O)$). The corresponding buffer requirements, as well as $req(O)$, and $rec(O)$ can be calculated using these throughput values. However, the throughput values used for the above calculation are only estimates. These (heuristic and initial) estimates are made on the basis of the overlap of the presentation times of the objects. When the values for $req(O)$ are determined, one might find that the object retrieval time overlaps in a different manner from their presentation times. Figure 8b shows a possible overlap of the retrieval schedules of the objects in Fig. 8a. Hence, the summation of the throughput estimates in the retrieval schedules has to be checked to ensure that the maximum offered throughput Th_{max} (by the network service provider) is not exceeded. A similar discussion also applies to buffer estimates. Checking the throughput and buffer estimates can be

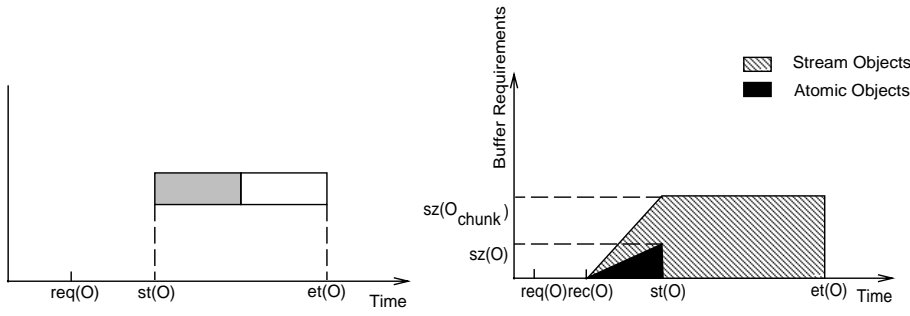
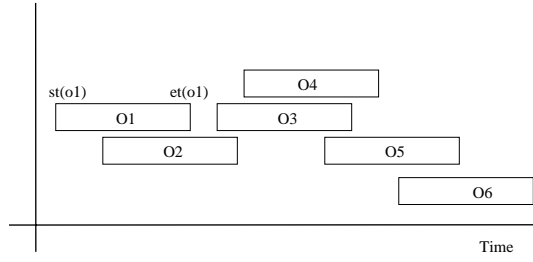


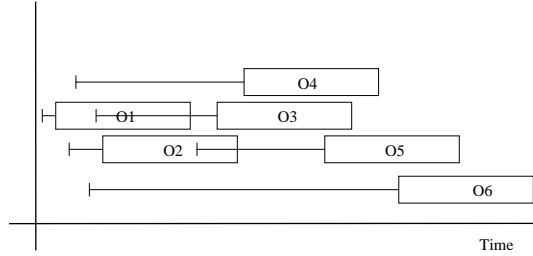
Fig. 7a,b. Single Object Retrieval

(a) Retrieval Schedule for a Single Object

(b) Buffer Requirements for a Single Object Presentation



a



b

Fig. 8a,b. Parallel, multiple object retrieval

done for each *time interval*. We can define a time interval (a, b) as the period between the occurrence of two successive events a, b . The events may be:

- request time of an object O ($req(O)$),
- presentation start time of an object O ($st(O)$),
- presentation end time of an object O ($et(O)$).

For each time interval, the constraints that must be obeyed by the schedules of all the objects sharing the same communication path are the following:

- throughput: $th(o_1) + \dots + th(o_n) \leq Th_{max}$;
- buffer: $buf(o_1, t) + \dots + buf(o_n, t) \leq Buf_{max}$

Example 5.1. Consider the two stream objects $o1$ and $o2$ that are scheduled as in Fig. 9.

The throughput requirement of the system can be calculated as follows.

- **interval (0-1).** No information is being transmitted on the connection line. Hence, the total throughput Th_{tot} is 0.
- **interval (1-2).** The initial fraction of the stream object $o1$ is being retrieved. Hence, assuming that $b_{init}(o) \leq Buf_{max}$, the total throughput requirement is

$$Th_{tot} = \frac{b_{init}(o1)}{(st(o1) - rec(o1))}$$

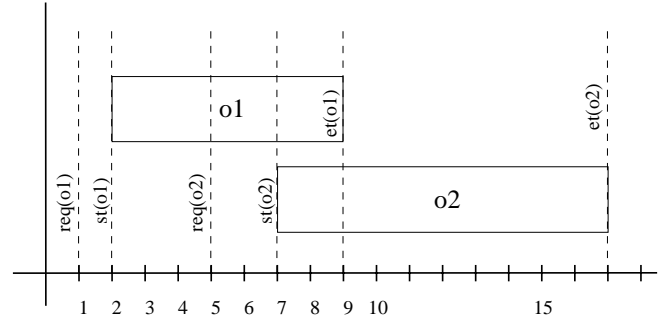


Fig. 9. Example 3

The buffer requirement, on the other hand, is

$$\begin{aligned} Buf_{tot}(t) &= buf(o1, t) = 0 \quad (\text{when } t < rec(o1)), \text{ and} \\ Buf_{tot}(t) &= buf(o1, t) \\ &= \left(\frac{b_{init}(o1)}{(st(o1) - rec(o1))} \right) \times (t - rec(o1)) \\ &\quad (\text{when } t \geq rec(o1)). \end{aligned}$$

- **interval (2-5).** The throughput on the communication path is equal to the consumption rate of $o1$. Hence,

$$\begin{aligned} Th_{tot} &= C_s(o1), \text{ and} \\ Buf_{tot} &= b_{init}(o1). \end{aligned}$$

- **interval (5-7).** Both $o1$ and $o2$ use the communication line: $o1$ receives the remaining portion of its stream information, and $o2$ receives its initial fraction.

$$Th_{tot} = \frac{b_{init}(o2)}{(st(o2) - rec(o2))} + C_s(o1).$$

If we assume that $\frac{b_{init}(o2)}{(st(o2) - rec(o2))}$ is non-zero, than $C_s(o1)$ must be strictly less than the available throughput.

Note that, during this interval, the following also hold:

$$\begin{aligned} buf(o1, t) &= b_{init}(o1), \\ buf(o2, t) &= 0 \quad (\text{when } t < rec(o2)), \text{ and} \\ buf(o2, t) &= \left(\frac{b_{init}(o2)}{(st(o2) - rec(o2))} \right) \times (t - rec(o2)) \\ &\quad (\text{when } t \geq rec(o2)). \end{aligned}$$

Hence, $buf(o1, t) + buf(o2, t)$ must be less than or equal to Buf_{max} . If this relation does not hold, then the schedule is not feasible. We will later show how to modify

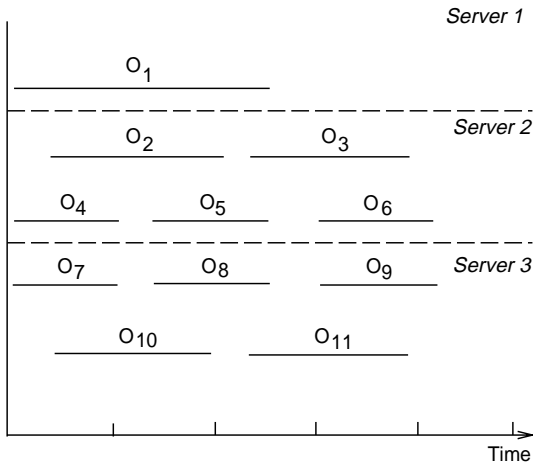


Fig. 10. Parallel, multiple object retrieval from multiple servers

non-feasible schedules to make them conform to the constraints imposed by the system. For now, we only state what needs to hold during the presentation.

- **interval (7-9).** Both $o1$ and $o2$ use the communication line to receive the remaining portions of their stream information:

$$Th_{tot} = C_s(o1) + C_s(o2)$$

and

$$b_{init}(o1) + b_{init}(o2) \leq Buf_{max}.$$

- **interval (9-17).** Only $o2$ is active and receiving the remaining portion of the stream information:

$$Th_{tot} = C_s(o2)$$

and

$$b_{init}(o2) \leq Buf_{max}.$$

Parallel, multiple object retrieval from multiple servers.

Figure 10 shows how objects composing a multimedia document presentation have to be retrieved from different servers. Here, separate network connections will be used for retrieving the objects from different servers. The throughput constraints for multiple object retrieval have to be satisfied for each network connection separately. However, the buffer constraint is the same, because the entire retrieval is handled by the client.

Interaction between throughput and buffer constraints.

Both throughput and buffer are resources provided by the system: the first is provided by the network service provider and the latter by the client system. When a certain throughput is offered by the network service provider, the required buffer resources become an estimate based on this offered throughput. The estimated buffer requirement, then, has to be checked to ensure that it can be provided by the client system. In order to achieve this, we go through a process of validating the retrieval schedule that was generated based on the above discussion.

6 Retrieval schedule validation

As discussed above, the generated retrieval schedule has to be checked to see whether it satisfies system constraints such as throughput and buffer. This validity is checked for every time interval in the generated retrieval schedule. It should be noted here that the retrieval schedule generation and validation process is done for every time segment in the entire multimedia presentation, as discussed in Sect. 3.1. In case modifications to the presentation or the retrieval schedules are necessary, it is easier to start from the end of the segment and work backwards. Working backwards from the segment end time helps in avoiding redoing already validated schedules. The following algorithm shows how the presentation and retrieval schedules for a given segment s is validated.

Input. A segment s , the tentative presentation and retrieval schedules for s , the throughput and buffer availability constraints that the system must obey.

Output. Validity or otherwise of the presentation and retrieval schedules for the segment s .

The algorithm starts from the last interval of the segment, and it proceeds towards the earlier intervals. This enables the system to first fix the *starting times* and then the *retrieval times*. As a result, the system tries to change the presentation schedule only if it cannot change the retrieval schedule. We chose this order because, as mentioned earlier, sticking to the presentation schedule is more desirable than sticking to the retrieval schedule.

Algorithm.

1. Sort the events (the vertical dashed-lines in the figure), and identify the number of intervals (num_{int}).
2. Set the borders of the intervals as **unmarked**. (When a border is **marked**, the event that corresponds to the border cannot be changed).
3. $Satisfied = True$;
4. $\forall FirstInterval \leq i \leq LastInterval (FailNo[i] = 0)$;
5. $ThisInterval = LastInterval$;
6. **while** $Satisfied$ and $(ThisInterval \geq FirstInterval)$ **do**
 - a) Check if the interval $ThisInterval$ is *valid*
 - b) **while** $ThisInterval$ is not *valid* **do**
 - i. $FailNo[ThisInterval] = FailNo[ThisInterval] + 1$;
 - ii. **if** $FailNo[ThisInterval] > MaxFails$ **then** return empty schedule
 - iii. Create k alternative feedbacks, each modifying either the presentation schedule or the retrieval schedule (here, we do not discuss how the feedback is generated). Note that the values that are already **marked** must be kept constant in feedbacks.
 - iv. Send all feedbacks to the *schedule modifier*.
 - v. **while** *schedule modifier* returns no schedule and $(ThisInterval < LastInterval)$ **do**
 - A. $ThisInterval = ThisInterval + 1$;
 - B. $FailNo[ThisInterval] = FailNo[ThisInterval] + 1$;
 - C. **if** $FailNo[ThisInterval] > MaxFails$ **then** return empty schedule

- D. Set the borders of the interval *ThisInterval* as **not-marked**
- E. Create k alternative feedbacks. Note that the values that are already **marked** must be kept constant in feedbacks.
- F. Send feedbacks to the *schedule modifier*.
- vi. Check if the interval *ThisInterval* is *valid*
- c) **if** interval *ThisInterval* is *valid* **then**
- i. Set the borders of the interval as **marked**
 - ii. $ThisInterval = ThisInterval - 1$
- d) **else**
- i. $Satisfied = False$
7. **if** $Satisfied$ **then** return the schedules
8. **else** return empty schedule

Example 6.1. Consider the two stream objects $o1$ and $o2$ that are scheduled as in Fig. 9.

Assume that the segment in Example 5.1 is fed into the schedule validator. Let us also assume that the throughput and the buffer constraints of the system are also as specified in Sect. 5.

The algorithm will start from the last interval, i.e., (9–17). It will check the throughput and the buffer constraints as specified in Example 5.1. Let us assume that this interval does not violate any constraints. The algorithm then marks the variables $et(o2)$ and $et(o1)$, i.e., it declares that the values of these variables should not change with subsequent operations.

The algorithm will then try to validate interval (7–9). Let us assume that the throughput required for this operation is more than the available throughput. One solution to this problem is to reduce the stream throughput of the object $o2$ and to move its request time to an earlier point in time (the details of this operation are described in greater detail in Sect. 7). Let us assume that the system decides to apply this solution, and it moves the $req(o2)$ from 5 to 4. Hence, as a result, the interval (5–7) changes to interval (4–7), and similarly the interval (2–5) changes to interval (2–4). At the end of this step, the variable $st(o2)$, which denotes the start of the interval (7–9), is marked by the system.

In subsequent iterations, the intervals (4–7), (2–4), (1–2), and (0–1) are going to be validated in a similar fashion.

In the next section, we show how the feedback is generated, and how it is used by the system.

7 Feedback generation and schedule modification

As discussed above, the schedule validator checks for the satisfiability of the two system constraints: throughput and buffer. If the throughput or buffer required by the schedules exceeds the appropriate upper bound, then the schedule validator declares the generated retrieval schedule invalid. In such cases, the schedules (retrieval, presentation or both) have to be modified so that the system constraints are satisfied. If schedule modifications are not possible, then the quality of the presentation can be reduced.

The schedule validator generates feedback for modifying the schedules, depending on how the system constraints were violated. In this section, we discuss how the schedule validator generates appropriate feedbacks.

7.1 Throughput violation

In Sect. 5, we showed that the total throughput needed for retrieval of multimedia objects in a time interval can be expressed as

$$Th_{tot} = \underbrace{\frac{c_1}{(st(o_1) - rec(o_1))} + \dots + \frac{c_n}{(st(o_n) - rec(o_n))}}_{non-stream} + \underbrace{d_1 + \dots + d_m}_{stream},$$

where c_1 through c_n are constants denoting the sizes of the non-stream information ($sz(o_i)$ for atomic object o_i and the initial buffer requirement $b_{init}(o_j)$ for stream object o_j), and d_1 through d_m are constants denoting the throughput requirements of the objects with constant consumption rate (stream objects).

If the total required throughput (Th_{tot}) exceeds the available throughput (Th_{max}), then we need to reduce the throughput requirement by

$$\delta_{thru} = Th_{tot} - Th_{max} = \underbrace{\delta_{thru}^1 + \dots + \delta_{thru}^n}_{non-stream} + \underbrace{\theta_{thru}^1 + \dots + \theta_{thru}^m}_{stream}.$$

Here, the δ s come from the atomic components, and the θ s come from stream components of the above equation. The amount of reduction required (δ_{thru}) is distributed on δ s and θ s using their priorities.

7.1.1 Handling non-stream retrievals

For reducing the throughput utilized by the non-stream information, i.e., the throughput of the form $\frac{c}{(st(o) - rec(o))}$, we need either to increase the time of retrieval ($st(o) - rec(o)$) or decrease the size of the object c . In other words, we have the following options:

- modify the retrieval schedule by changing $rec(o)$;
- modify the presentation schedule by changing $st(o)$;
- modify the quality of the presentation by reducing the size c .

Changing the retrieval schedule or the presentation schedule involves modification of the value of $rec(o)$ (time at which the object has to arrive at the client side) and $st(o)$ (the presentation start time). The modification of the retrieval schedule is the most desirable option since it does not involve any change in the presentation schedule or the quality of presentation.

The desired reduction in throughput for non-stream information of size c_j can be expressed as

$$\frac{c_j}{(st(o_j) - rec(o_j))} - \frac{c_j}{(st'(o_j) - rec'(o_j))} \geq \delta_{thru}^j,$$

where st and rec denote the current values of the presentation start time and the receive time, st' and rec' denote the corresponding new values, and δ_{thru}^j is a positive real number. The above equation can also be rewritten as

$$st'(o_j) - rec'(o_j) \geq \frac{c_j}{-\delta_{thru}^j + \frac{c_j}{(st(o_j) - rec(o_j))}}.$$

Modifying the retrieval schedule. The retrieval schedule $rec(o)$ can be modified by keeping the presentation start time unchanged (i.e., $st'(o) = st(o)$). Hence, the new value for $rec'(o)$ is

$$rec'(o_j) \leq st(o) - \frac{c_j}{-\delta_{thru}^j + \frac{c_j}{(st(o_j) - rec(o_j))}}.$$

If this change in the retrieval schedule is not acceptable (for example, if it leads to a longer wait time before the presentation can be started), then the presentation schedule $st(o)$ can be modified as follows.

Modifying the presentation schedule. To modify the presentation start time of an object, we need to keep the retrieval schedule $rec(o)$ unchanged (i.e., $rec'(o) = rec(o)$). When we substitute $rec'(o_j) = rec(o_j)$ in the equation, we get

$$st'(o_j) \geq rec(o_j) + \frac{c_j}{-\delta_{thru}^j + \frac{c_j}{(st(o_j) - rec(o_j))}}.$$

The range for the new presentation start time of the object o_j can be fed into a presentation schedule modifier (discussed in Sect. 7.3) in order to find another feasible value for $st(o_j)$. The modifier generates a new presentation schedule with the new presentation start time for object o_j , such that the values of the already validated presentation schedule variables are kept unchanged. Sect. 7.3 discusses this issue in more detail.

In both the above cases (modifying the presentation schedule and modifying the retrieval schedule), there will be a change in the intervals of the segment. Since the validation process is carried out backwards starting from the segment end time, and since the values of the already validated intervals are kept unchanged, there is no need for backtracking in the validation process. The interval changes only affect the non-validated parts of the segment.

Modifying the quality of presentation. If the above modifications of retrieval and presentation schedules are not feasible, then the quality of the presentation may be modified as a last resort. Reduction in the quality of an object implies a reduction in the size of the information to be retrieved. A smaller sized object can than be retrieved with a smaller throughput. As described in Sect. 2.2, this operation can be performed by using a look-up table to see what the smallest feasible reduction in the object quality is.

7.1.2 Handling stream retrievals

The stream retrieval part deals with the throughput requirements of the stream objects *after* the object presentation has started. For reducing the throughput required for stream object retrieval, we have the following options.

- Buffering a larger initial chunk of the object *before* the presentation,
- Reducing the quality of the presentation by reducing the size of the objects to be retrieved.

Increased initial buffering. Let us assume that, for the stream object o_j , the required reduction in throughput is θ_{thru}^j . This reduction is for the time interval $\langle st(o_j), et(o_j) \rangle$, corresponding to the start and end times of presentation of the objects o_j . Hence, the required increase in the size of the initial buffer can be calculated as

$$buf_{inc}^j = duration_j \times \theta_{thru}^j,$$

$$\text{where } duration_j = et(o_j) - st(o_j).$$

Increasing the initial buffer size, however, may cause buffer violations at the already validated intervals. Hence, an increase can be allowed only if there is enough available buffer space to accommodate the suggested increase: buf_{inc}^j must be limited by the minimum buffer size available during the already validated portion of o_j 's presentation (Fig. 11).

Note that an increase in the size of the initial buffer requires that the retrieval schedule for the initial chunk is suitably modified. The change in $rec(o)$ can be calculated as follows:

$$\delta_{rec(o_j)} = buf_{inc}^j \times th(o_j),$$

where $th(o_j)$ denotes the throughput assigned to object o_j for the initial chunk retrieval.

7.2 Buffer violation

The other system resource that may be inadequate is buffer space. In Sect. 5, we showed that the total buffer requirement at a time instant t is

$$\begin{aligned} \text{Buf}_{tot} &= \left. \begin{aligned} &\frac{c_1}{(st(o_1) - rec(o_1))} \times (t - rec(o_1)) + \dots \\ &\dots + \frac{c_n}{(st(o_n) - rec(o_n))} \times (t - rec(o_n)) \end{aligned} \right\} \text{non-stream} \\ &\quad + \underbrace{e_1 + \dots + e_m}_{\text{stream}}, \end{aligned}$$

where c_1 through c_n are constants denoting the sizes of the non-stream information being retrieved, and e_1 through e_m are constants denoting the buffer requirements of the stream objects.

If we consider an interval of the form $\langle t_{start}, t_{end} \rangle$, the total buffer requirement at the end of the time interval is

$$\begin{aligned} \text{Buf}_{tot} &= \left. \begin{aligned} &\frac{c_1}{(st(o_1) - rec(o_1))} \times (t_{end} - rec(o_1)) + \dots \\ &\dots + \frac{c_n}{(st(o_n) - rec(o_n))} \times (t_{end} - rec(o_n)) \end{aligned} \right\} \text{non-stream} \\ &\quad + \underbrace{e_1 + \dots + e_m}_{\text{stream}}. \end{aligned}$$

Note that the maximum buffer requirement within an interval occurs at the end of the interval. Hence, the above equation also gives the maximum buffer requirement within the interval $\langle t_{start}, t_{end} \rangle$. If Buf_{tot} calculated as above exceeds the available buffer space (Buf_{max}) of the system, then we must reduce the buffer usage by

$$\delta_{buf} = \text{Buf}_{tot} - \text{Buf}_{max}.$$

Note that δ_{buf} can also be written as

$$\delta_{buf} = \delta_{buf}^1 + \dots + \delta_{buf}^n + \theta_{buf}^1 + \dots + \theta_{buf}^m.$$

As in the discussion for throughput violation, the above equation comprises two components:

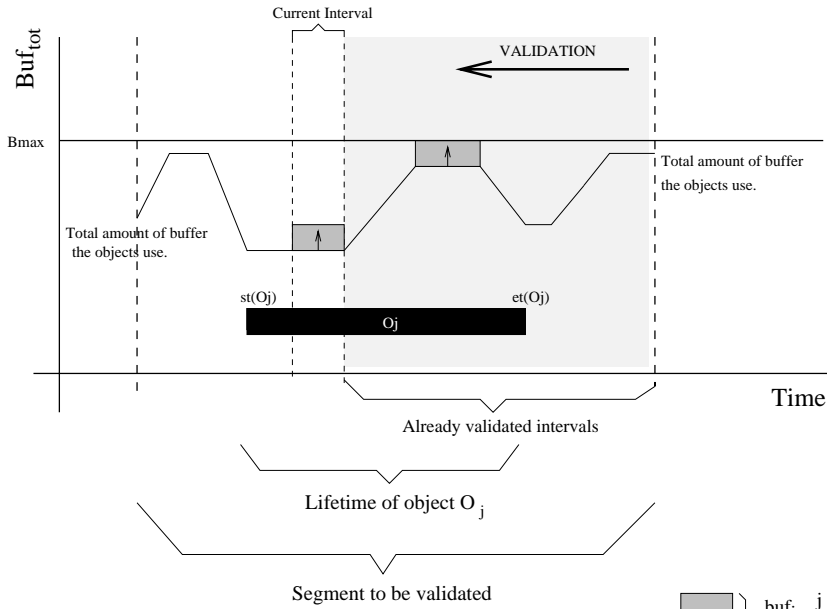


Fig. 11. The increase in the buffer size is limited with the minimum available buffer

- (non-stream) buffer requirements for atomic objects and initial chunk retrieval of stream objects,
- (stream) buffer requirements during the presentation of stream objects.

7.2.1 Handling non-stream retrievals

To reduce the buffer requirements during the retrieval of the atomic objects and the initial chunk retrieval of stream objects (i.e., the component $\frac{c}{(st(o) - rec(o))} \times (t - rec(o))$, we need to either increase the retrieval time ($st(o) - rec(o)$) or reduce the size of the object c . This, in effect, results in one or more of the following.

- Modification of the retrieval schedule ($rec(o)$).
- Modification of the presentation schedule ($st(o)$).
- Modification of the presentation quality (c).

Changing the retrieval or presentation schedule involves modification of the value of $rec(o)$ and $st(o)$, i.e., the time at which the object has to arrive at the client side and the presentation start time. The modification of the retrieval schedule is most desirable, since it does not involve any change in the presentation schedule or the quality of presentation. The desired reduction in the buffer requirement for an object o_j can be expressed as

$$\frac{c_j}{(st(o_j) - rec(o_j))} \times (t_{end} - rec(o_j)) - \frac{c_j}{(st'(o_j) - rec'(o_j))} \times (t_{end} - rec'(o_j)) \geq \delta_{buf}^j,$$

where st and rec denote the current values of these variables, st' and rec' denote the values that we are searching for, and δ_{buf}^j is a positive real number. The above equation can be rewritten as

$$\frac{c_j}{(st'(o_j) - rec'(o_j))} \times (t_{end} - rec'(o_j)) \leq -\delta_{buf}^j + \frac{c_j}{(st(o_j) - rec(o_j))} \times (t_{end} - rec(o_j)).$$

Modifying the retrieval schedule. This involves a change in the value of $rec(o)$. To change $rec(o)$, we should keep the presentation start time unchanged (i.e., $st'(o) = st(o)$). Substituting $st'(o_j) = st(o_j)$ in the above equation, we get

$$\frac{c_j}{(st(o_j) - rec'(o_j))} \times (t_{end} - rec'(o_j)) \leq -\delta_{buf}^j + \frac{c_j}{(st(o_j) - rec(o_j))} \times (t_{end} - rec(o_j))$$

$$(\gamma - c_j) \times rec'(o_j) \leq (\gamma \times st(o_j)) - (c_j \times t_{end})$$

$$\begin{cases} rec'(o_j) \leq \frac{(\gamma \times st(o_j)) - (c_j \times t_{end})}{(\gamma - c_j)} & \text{for } (\gamma - c_j) > 0 \\ rec'(o_j) \geq \frac{(\gamma \times st(o_j)) - (c_j \times t_{end})}{(\gamma - c_j)} & \text{for } (\gamma - c_j) < 0 \end{cases}$$

$$\begin{cases} rec'(o_j) - rec(o_j) \leq \frac{(\gamma \times st(o_j)) - (c_j \times t_{end})}{(\gamma - c_j)} - rec(o_j) & \text{for } (\gamma - c_j) > 0 \\ rec'(o_j) - rec(o_j) \geq \frac{(\gamma \times st(o_j)) - (c_j \times t_{end})}{(\gamma - c_j)} - rec(o_j) & \text{for } (\gamma - c_j) < 0 \end{cases}$$

The above set of equations gives a range ($rec(o_j) - rec'(o_j)$) in which the retrieval schedule can be suitably modified.

Modifying the presentation schedule. This involves a change in the value of $st(o)$. To do this, we should keep the object retrieval time unchanged (i.e., $rec(o_j) = rec'(o_j)$). Substituting $rec(o_j) = rec'(o_j)$ in the above equation, we get

$$\frac{c_j}{(st'(o_j) - rec(o_j))} \times (t_{end} - rec(o_j)) \leq -\delta_{buf}^j - \frac{c_j}{(st(o_j) - rec(o_j))} \times (t_{end} - rec(o_j))$$

$$\begin{cases} st'(o_j) \geq \frac{c_i \times (t_{end} - rec(o_j))}{\gamma} + rec(o_j) & \text{for } (\gamma > 0) \\ st'(o_j) \leq \frac{c_i \times (t_{end} - rec(o_j))}{\gamma} + rec(o_j) & \text{for } (\gamma < 0) \end{cases}$$

$$\begin{cases} st'(o_j) - st(o_j) \geq \frac{c_i \times (t_{end} - rec(o_j))}{\gamma} + rec(o_j) \\ \text{for } (\gamma > 0) - st(o_j) \\ st'(o_j) - st(o_j) \leq \frac{c_i \times (t_{end} - rec(o_j))}{\gamma} + rec(o_j) \\ \text{for } (\gamma < 0) - st(o_j) \end{cases}$$

The above set of equations gives us a range ($st'(o_j) - st(o_j)$) in which the presentation start time can be suitably modified. This range has to be given to the presentation schedule modifier (discussed in Sect. 7.3) to generate a new schedule with the presentation start time for the object o_j in the suggested range. While generating this new presentation schedule, the schedule modifier keeps the values of the other presentation variables (start and end times of presentations of other objects) constant.

Modifying the presentation quality. This involves reducing the size of the objects to be retrieved. The procedure is similar to the one discussed for handling throughput violation in Sect. 7.1.

7.2.2 Handling stream retrievals

The reduction in the stream components of the buffer usage can only be made by reducing the size of the objects: a reduction in the object size would reduce the amount of buffers needed for its storage. Reduction in presentation quality can be made as a last resort. The procedure is the same as the one discussed for handling throughput violation in Sect. 7.1.

7.3 Presentation schedule modifier

The schedule modifier module takes as input the range of values for the presentation start times of the objects, as suggested by the feedback generator. The schedule modifier then generates a new schedule in which only the start times of the objects suggested by the feedback generator are modified and other presentation variables (i.e., the start and end presentation times of other objects) are left unchanged.

Definition 7.1 (Presentation schedule modifier). Let G be a weighted, directed constraint graph which represents the temporal specifications of the multimedia document. Let v_i denote the temporal variables of the document, and let ϕ denote a schedule for the document, i.e., a mapping from the variables into reals.

Let δ be a mapping from the vertex variables into $(R \times R) \cup \perp$, where $\delta(v_i)$ specifies the range of the change required in the value of v_i . If $\delta(v_i) = \perp$, then the value of v_i can be changed freely. Note that $\delta(st_p)$ must always be $\langle 0, 0 \rangle$.

The *presentation schedule modifier* takes G , ϕ (the initial solution) and δ (the feedback) as inputs, and it returns a new solution ϕ_{new} . If, however, there is no solution satisfying the feedback, then the algorithm returns false.

In order to see how the schedule modifier works, observe that, if $\delta(v_i) = \langle a, b \rangle$, then

$$\begin{aligned} \phi(v_i) + a &\leq \phi_{new}(v_i), \text{ and} \\ \phi_{new}(v_i) &\leq \phi(v_i) + b. \end{aligned}$$

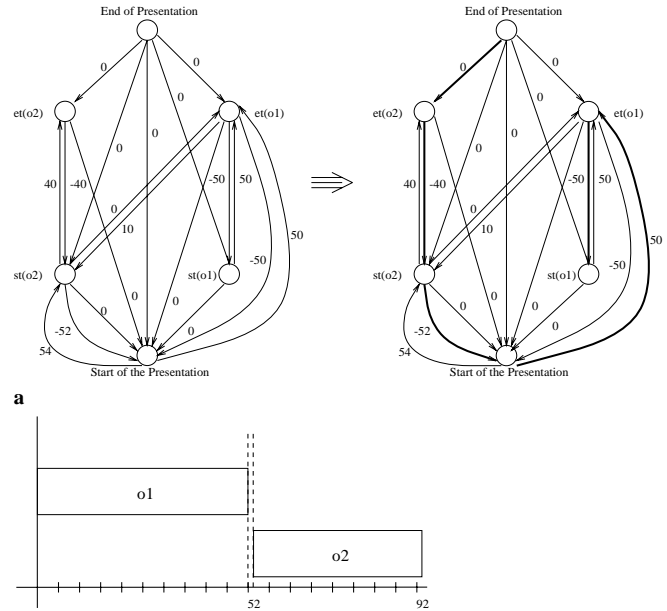


Fig. 12a. The modified constraint graph and **b** the corresponding solution

Since the values of the temporal variables are with respect to the start of the presentation, i.e., $\phi_{new}(v_i) = v_i - st_p$, we can rewrite the above inequalities as

$$\begin{aligned} st_p - v_i &\leq -(\phi(v_i) + a), \text{ and} \\ v_i - st_p &\leq \phi(v_i) + b. \end{aligned}$$

Hence, for each variable v_i such that $\delta(v_i) \neq \perp$, we need to introduce two new constraints to the document. These insertions can be represented as the additions of two new edges to the graph G . Algorithm A9 (given in the Appendix) handles incremental additions of constraints and provides a new solution.

Example 7.1. Let us consider the multimedia document described in Example 4.1. Let us assume that the presentation schedule found in that example does not lead into a suitable retrieval schedule, and the validator asks the schedule modifier to postpone the start of the object o_2 by 2–4 s while keeping the end of the object o_1 as it is (we show in Sect. 7) how the feedbacks are generated).

The inputs to the schedule modifier, that is ϕ and δ , are as follows:

$$\begin{aligned} \phi(st_p) &= 0 & \delta(st_p) &= \langle 0, 0 \rangle \\ \phi(st(o_1)) &= 0 & \delta(st(o_1)) &= \perp \\ \phi(et(o_1)) &= 50 & \delta(et(o_1)) &= \langle 0, 0 \rangle \\ \phi(st(o_2)) &= 50 & \delta(st(o_2)) &= \langle 2, 4 \rangle \\ \phi(et(o_2)) &= 90 & \delta(et(o_2)) &= \perp \\ \phi(et_p) &= 90 & \delta(et_p) &= \perp \end{aligned}$$

Hence, the corresponding new constraints are

$$\begin{aligned} st_p - st(o_1) &\leq -50, \text{ and} \\ st(o_1) - st_p &\leq 50. \\ st_p - st(o_2) &\leq -52, \text{ and} \\ st(o_2) - st_p &\leq 54. \end{aligned}$$

The corresponding new graph can be seen in Fig. 12a. At the end, the vertices of the graph and the ϕ_{new} have the following values:

$$\begin{array}{ll}
et_p = 0 & \phi_{new}(st_p) = 92 \\
st_p = -92 & \phi_{new}(et_p) = 0 \\
et(o_2) = 0 & \phi_{new}(et(o_2)) = 92 \\
st(o_2) = -40 & \phi_{new}(st(o_2)) = 52 \\
et(o_1) = -42 & \phi_{new}(et(o_1)) = 50 \\
st(o_1) = -92 & \phi_{new}(st(o_1)) = 0
\end{array}$$

Figure 12b shows the corresponding schedule.

8 The CHIMP project

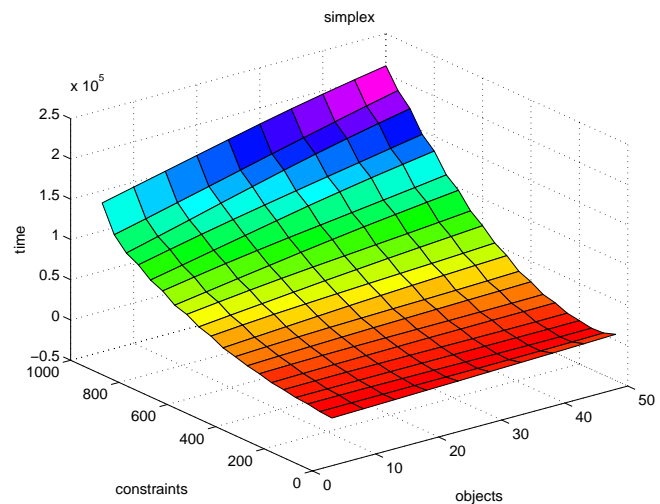
This paper is part of the Collaborative Heterogeneous Interactive Multimedia Platform (CHIMP) project, which has the goal of studying the technical aspects of collaborative multimedia document authoring and presentation, as well as building a system based on these results. Suppose we consider a team of individuals jointly *authoring* a multimedia document. In order to successfully author such a document, the authors must

- identify the objects (e.g., audio objects, video objects, text objects, etc.) that will be part of the authored multimedia document. This is studied in detail by Marcus and Subrahmanian [22, 23] who showed that a fragment of Datalog queries may be used to identify objects of interest,
- specify how these objects should be presented to an end-user wishing to view the final multimedia document. This specification includes, amongst other things, the temporal constraints used to generate the presentation. In contrast to previous work [2, 18, 23] that describes how arbitrary constraints may be used to specify presentations, CHIMP benefits from the use of a small class of constraints called difference constraints that are adequate for specifying very flexible presentations.

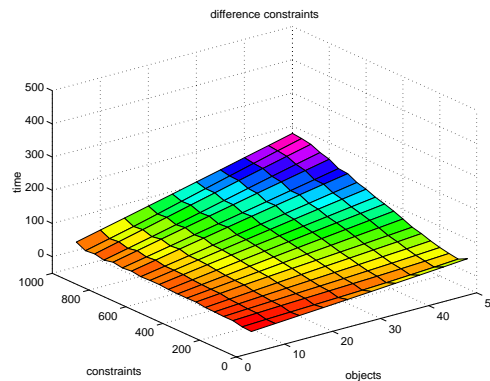
In addition to the presentation constraints, when a set of objects are scattered across the network, we need to generate a *retrieval schedule* that specifies how the CHIMP retrieval engine will retrieve the desired objects from other locations by interacting with remote servers. This too, constitutes parts of the CHIMP project, and includes the study of resource reservation algorithms, as well as servers that can be used for networked delivery of multimedia objects [3, 4], keeping in mind the available resources (which include bandwidth resources, buffer resources, available viewing formats at different nodes, etc.). CHIMP is currently implemented on the SUN/Unix platform, and includes some but not all the services listed above.

9 Experiments

In this section, we describe the results of the experiments we performed to measure the effectiveness of our temporal scheduling approach compared with the existing approaches. Buchanan and Zellweger [2], use the *simplex* algorithm to create presentation schedules given a set of objects and a set of presentation specifications. However, the *simplex* algorithm has major disadvantages. First of all it is very costly (exponential time complexity); second, it is not optimized



a



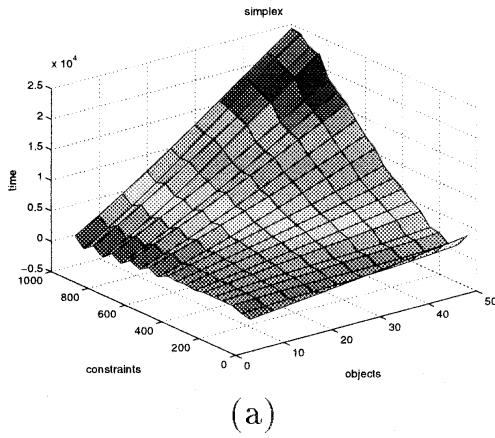
b

Fig. 13a,b. Comparison of successful running times (in milliseconds): **a** Time necessary for simplex to find a solution, **b** time necessary for shortest path based algorithm to find a solution. NOTE: The scales of the two graphs are different

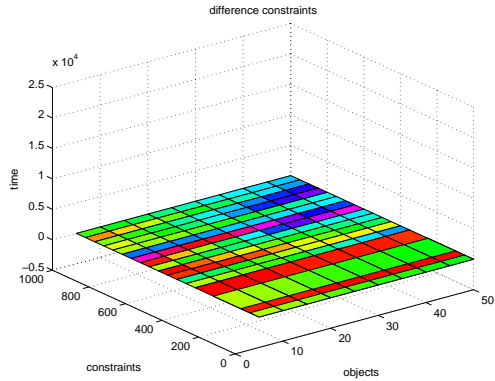
to handle difference constraints; third, it cannot handle inconsistencies that may occur in the input set of presentation constraints.

In the first experiment, we varied the number of objects in the presentation from 5 to 45 and we varied the number of presentation specifications (constraints) from 10 to 1000 (the constraints specify the length of the objects as well as their relative temporal positioning). The presentation specifications are generated randomly to observe the general behavior of the algorithms; however, we made sure that the randomly generated specifications are consistent. Figure 13 shows the performance comparison of our approach with the *simplex* method. Note that, while our approach finds a solution within hundreds of milliseconds, simplex may spend up to 3–4 min to find a solution as the number of objects and constraints increase. This difference is due to the fact that our approach takes advantage of the restricted syntax of the constraints, while simplex does not.

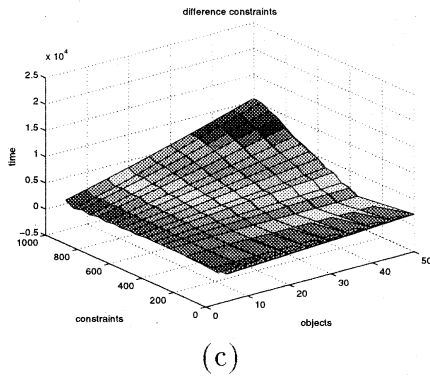
In the second experiment, we observed how suitable the two different approaches are for the environments where inconsistencies are very likely (such as with multi-authored presentation specifications, or scheduling with presentation constraints plus network constraints). We again varied the



a



b



c

Fig. 14a–c. Comparison of running times (in ms) with inconsistent input sets. **a** Time necessary for simplex to identify inconsistencies, **b** time necessary for our algorithm to identify inconsistencies, **c** time necessary for our algorithm to both identify and remove inconsistencies and provide a solution

number of objects in the presentation from 5 to 45 and we varied the number of presentation specifications from 10 to 1000. However, this time we let the specifications have random numbers of inconsistencies.

Figure 14 shows a performance comparison of our algorithm with the *simplex* method when inconsistencies are allowed. Figure 14a shows how the *simplex* algorithm worked in our simulation. Note that the time required for the *simplex* algorithm to determine that a set of speci-

cations is inconsistent increases dramatically with an increase in the number of objects and constraints. However, as Fig. 14b shows, our approach can identify inconsistencies very quickly. Furthermore, it can remove the inconsistencies and provide a solution in much less time (Fig. 14c) than that taken by *simplex* just to identify the inconsistency. *Simplex*, of course, would not remove the inconsistencies.

These results show that our approach works significantly better than the *simplex* method, both when there are no inconsistencies and when there are inconsistencies. Note that, in the later case, *simplex* cannot give a solution, whereas our algorithm can find a consistent subset of the input constraints.

10 Conclusion

A distributed multimedia document involves retrieval of objects from server(s) and their presentation at the client systems. The presentation of the multimedia objects have to be carried out in accordance with the specified temporal relationships among the objects composing the presentation. Flexibility in the specification of the temporal relationships helps in deriving a set of possible presentation schedules, with each schedule representing one possible *view* of the document. The retrieval of multimedia objects from the server(s) is influenced by factors such as

- presentation schedule of the multimedia objects,
- maximum throughput offered by the network service provider,
- maximum buffer resources available on the client system.

In the previous approaches [17, 19, 29, 30], the multimedia presentation schedule is fixed before the generation of the retrieval schedule. Based on an assumed network throughput availability, the retrieval schedules are derived to generate a retrieval schedule. The generated schedules do not handle variations in the offered system resources such as network throughput and available buffer resources.

In this paper, we have developed techniques for deriving flexible object presentation and retrieval schedules for a distributed multimedia document presentation. The main advantage in the proposed methodology is that, instead of choosing a presentation schedule and trying to find a matching retrieval schedule, the proposed algorithms modify both the presentation and retrieval schedules in such a way that system resources (network throughput and buffer resources in the client system) are used in a very efficient manner.

References

1. Allen JF (1984) Towards a General Theory of Time and Action. *Artif Intell* 23: 123–154
2. Buchanan MC, Zellweger PT (1993) Automatic Temporal Layout Mechanisms. In: *ACM Multimedia 93*, pp 341–350, Anaheim, California
3. Selçuk Candan K, Subrahmanian VS, Venkat Rangan P (1995) Collaborative Multimedia Systems: Synthesis of Media Objects. Technical Report. CS-TR-3595, UMIACS-TR-96-8, University of Maryland, College Park, Computer Science Technical Report Series

4. Selçuk Candan K, Subrahmanian VS, Venkat Rangan P (1996) Towards a Theory of Collaborative Multimedia. In: IEEE International Conference on Multimedia Computing and Systems, Hiroshima, Japan, pp 278–282
5. Selçuk Candan K, Prabhakaran B, Subrahmanian VS (1996) CHIMP: A Framework for Supporting Distributed Multimedia Document Authoring and Presentation. In: Fourth ACM International Multimedia Conference, Boston, Mass., pp 329–340
6. Selçuk Candan K, Prabhakaran B, Subrahmanian VS (1996) Collaborative Multimedia Documents: Authoring and Presentation. Technical Report: CS-TR-3596, UMIACS-TR-96-9, University of Maryland, College Park, Computer Science Technical Report Series
7. Cormen TH, Leiserson CE, Rivest RL (1990) Introduction to Algorithms. McGraw-Hill
8. Ferrari D (1990) Client Requirements For Real-Time Communication Services. IEEE Commun Mag 28 (11): 65–72
9. Ferrari D, Ramaekers J, Ventre G (1992) Client-Network Interactions in Quality of Service Communication Environments. In: Proc. of High-Performance Networking, 4th IFIP Conf. on High Performance Networking, Liege, Belgium, 14–18 December, pp E1.1–E1.14
10. Gajewska H (1994) Argo: A System for Distributed Collaboration. IN: ACM Multimedia 94, San Francisco, Calif, pp 433–440
11. Ginsberg A, Ahuja S (1995) Automating envisionment of virtual meeting room histories. In: ACM Multimedia 95, San Francisco, Calif, pp 65–76
12. Gong F (1994) Multipoint Audio and Video Control for Packet-Based Multimedia Conferencing. In: ACM Multimedia 94, San Francisco, Calif, pp 425–432
13. Hillier F, Lieberman G (1974) Operations Research. Holden-Day
14. Imai T, Yamaguchi K, Muranaga T (1994) Hypermedia Conversation Recording to Preserve Informal Artifacts in Realtime. In: ACM Multimedia 94, San Francisco, Calif, pp 417–424
15. Korth H, Silberschatz A (1986) Database System Concepts. McGraw-Hill
16. Kim MY, Song J (1995) Multimedia Documents with Elastic Time. In: ACM Multimedia 95, San Francisco, Calif, pp 143–154
17. Li L, Karmouch A, Georganas ND (1994) Multimedia Teleorchestra With Independent Sources: Part 1 and Part 2. J Multimedia Syst 1 (4) 143–165
18. Little TDC, Ghafoor A (1990) Synchronization and Storage Models for Multimedia Objects, IEEE J Sel Areas Commun 8 (3): 413–427
19. Little TDC, Ghafoor A (1991) Multimedia Synchronization Protocols for Broadband Integrated services. IEEE J Sel Areas Commun 9 (9): 1368–1382
20. Little TDC, Ghafoor A (1993) Interval-Based Conceptual Models for Time-Dependent Multimedia Data. IEEE Trans Knowl Data Eng 5 (4): 551–563
21. Manohar NR, Prakash A (1995) Dealing With Synchronization and Timing Variability in the Playback of Interactive Session Recordings. In: ACM Multimedia 95, San Francisco, Calif, pp 45–56
22. Marcus S, Subrahmanian VS (1995) Towards a Theory of Multimedia Database Systems. In: Subrahmanian VS, Jajodia S (eds) Multimedia Database Systems: Issues and Research Directions. Springer, Berlin Heidelberg New York, pp 1–35
23. Marcus S, Subrahmanian VS (1996) Foundations of Multimedia Database Systems. ACM 43 (3): 474–523
24. Özsoyoglu G, Hakkoymaz V, Kraft JD (1996) Automating the Assembly of Presentations from Multimedia Databases. Twelfth International Conference on Data Engineering, New Orleans, February 1996, pp 593–601
25. Perez-Luque MJ, Little TDC (1996) A Temporal Reference Framework for Multimedia Synchronization. IEEE Journal on Selected Areas in Communications (Special Issue: Synchronization Issues in Multimedia Communication), Vol. 14, No. 1, pp 36–51
26. Prabhakaran B, Raghavan SV (1994) Synchronization Models For Multimedia Presentation With User Participation. J Multimedia Syst 2(2): 53–62
27. Prabhakaran B (1996) Multimedia Synchronization. In: Multimedia Systems and Techniques. Kluwer, Dorecht pp 177–214
28. Qazi NU, Woo M, Ghafoor A (1993) A Synchronization and Communication Model for Distributed Multimedia Objects. In: Proceedings of the First ACM Conference on MultiMedia Systems, Anaheim, Calif, pp 147–155
29. Raghavan SV, Prabhakaran B, Tripathi SK (1996) Synchronization Representation and Traffic Source Modeling in Orchestrated Presentation. Special issue on Multimedia Synchronization: IEEE J Sel Areas Commun Vol. 14, No (2)
30. Raghavan SV, Prabhakaran B, Tripathi SK (1994) Quality of Service Considerations For Distributed, Orchestrated Multimedia Presentation. Technical Report: CS-TR-3167, UMIACS-TR-93-113, University of Maryland, College Park, Computer Science Technical Report Series
31. Raghavan SV, Prabhakaran B, Tripathi SK (1996) Handling QoS Negotiations in Orchestrated Multimedia Presentations. J High-Speed Networking 5(3): 277–292
32. Rajan S, Rangan PV, Vin HM (1995) A Formal Basis for Structured Multimedia Collaborations. In: IEEE Int. Conf. on Multimedia Computing and Systems, Washington DC, pp 194–201
33. Steinmetz R (1990) Synchronization Properties in Multimedia Systems. IEEE J Sel Areas Commun 8(3): 401–412
34. Stotts PD, Furuta R (1990) Temporal Hyperprogramming. J Visual Languages Comput 7: 237–253
35. Thimm H, Klas W (1996) δ -Sets for Optimal Reactive Adaptive Play-out Management in Distributed Multimedia Database Systems. In: 12th International Conference on Data Engineering, New Orleans, pp 584–592
36. Beek P van (1990) Exact and Approximate Reasoning about Qualitative Temporal Relations. PhD. Thesis, University of Waterloo, Ont., Canada
37. Vilain M, Kautz H (1986) Constraint Propagation Algorithms for Temporal Reasoning. In: Proceedings of AAAI-86, Artificial Intelligence, Philadelphia, Pennsylvania, pp 377–382
38. Wittenburg TM, Little TDC (1994) An Adaptive Document Management System for Shared Multimedia Data. In: IEEE Intl. Conf. on Multimedia Computing and Systems, pp 245–254
39. Wolf KH, Froitzheim K, Schulthess P (1995) Multimedia Application Sharing in a Heterogeneous Environment. In: ACM Multimedia 95, San Francisco, Calif, pp 57–64
40. Wray S, Glauert T, Hopper A (1994) The Medusa Applications Environment. In: IEEE Int. Conf. on Multimedia Computing and Systems, pp 265–274

Appendix

Algorithms for solving difference constraints

A1 : INITIALIZE_SINGLE_SOURCE (G, v_s)

1. **for** each vertex $v \in V[G]$ **do**
 - (a) $d[v] \leftarrow \infty$
 - (b) $\pi[v] \leftarrow NIL$
2. $d[s] \leftarrow 0$
3. $all_cycles = \emptyset$

A2 : RELAX (u, v, w)

1. **if** $d[v] > d[u] + w(u, v)$ **then**
 - (a) $d[v] \leftarrow d[u] + w(u, v)$
 - (b) $\pi[v] \leftarrow u$

A3 : RELAX_and_MARK_CYCLE (u, v, w)

1. $relaxed = 0$
2. **if** $d[v] > d[u] + w(u, v)$ **then**
 - (a) **if** NOT_CYCLE (u, v) **then**
 - i. $d[v] \leftarrow d[u] + w(u, v)$
 - ii. $\pi[v] \leftarrow u$
 - iii. $relaxed = 1$
3. **return**(relaxed)

A4 : NOT_CYCLE (u, v)

1. $cycle = PATH(u, v)$
2. **if** $cycle \neq \perp$ **then**
 - (a) $cycle = cycle \rightarrow v$

(b) `all_cycles = all_cycles \cup {cycle}`

(c) `return(0)`

3. **else**

(a) `return(1)`

A5 : PATH (u, v)

1. **if** $\pi[u] = \perp$ **then**

(a) `return(\perp)`

2. **if** $\pi[u] = v$ **then**

(a) `return($v \rightarrow u$)`

3. **else**

(a) `temp_path = PATH($\pi[u], v$)`

(b) **if** `temp_path = \perp` **then**

i. `return(\perp)`

(c) **else**

i. `return(temp_path $\rightarrow u$)`

A6 : REMOVE_CYCLES (G, cycles)

1. Let `cycles` be (`auth_cycles \cup sys_cycles`)

2. `<deleted_cons, marked_cons> = CONSULT_AUTHORS(auth_cycles)`

3. delete the constraints in `deleted_cons` from the graph

4. mark the constraints in `marked_cons` as unsatisfiable

5. let E be the set of edges involved in any of the cycles in `sys_cycles`, and let $pri(e)$ be the priority of the edge e

6. sort E with respect to the priorities in ascending order

7. $e_m = 1$

8. **while** `sys_cycles $\neq \emptyset$` **do**

(a) remove all negative cycles containing e_m from `sys_cycles`

(b) mark e_m as unsatisfiable

(c) $e_m = e_m + 1$

A7 : SOLVE_WITHOUT_CYCLE_CHECK (G, w, s)

1. INITIALIZE_SINGLE_SOURCE (G, s)

2. **for** $i = 1$ **to** $|V[G]|$ **do**

(a) **for** each edge $(u, v) \in E[G]$ **do**

i. **do** RELAX(u, v, w)

A8 : SOLVE_AND_MARK_CYCLE (G, w, s)

1. INITIALIZE_SINGLE_SOURCE (G, s)

2. **for** $i = 1$ **to** $|V[G]|$ **do**

(a) **for** each edge $(u, v) \in E[G]$ **do**

i. RELAX_AND_MARK_CYCLE (u, v, w)

3. **if** `all_cycles $\neq \emptyset$` **then**

(a) `temp_cycles = copy(all_cycles)`;

(b) $G' = \text{REMOVE_CYCLES}(G, \text{all_cycles})$

(c) `all_cycles = \emptyset`

(d) SOLVE_AND_MARK_CYCLE(G', w, s)

(e) `all_cycles = temp_cycles \cup all_cycles`

A9 : INSERT_CONSTRAINT (G, w, s, G', T, e)

1. let e be from u to v with weight w

2. `relaxed = false; cyclefound = false;`

3. **if** $d[v] \leq d[u] + w(e)$ **then**

(a) `insertion = nontree`

(b) `exit`

4. let aps be the all_pairs_shortestpath matrix of G'

5. `old_graph = copy_graph(G')`

6. **while** (`relaxed = false`) **do**

(a) $d' = \text{copy_array}(d)$

(b) $d'[v] = d'[u] + w(e)$

(c) $\text{cyc} = \perp$; `edgestoinset = e`

(d) `oldedge = \perp`

(e) **while** (`(edgestoinset $\neq \perp$) and (cyclefound = false)`) **do**

i. `newedge = head(edgestoinset)`

ii. `edgestoinset = tail(edgestoinset)`

iii. **if** (`oldedge $\neq \perp$`) **then**

A. **while** (`source(newedge) \neq destination(head(cyc))`) **do**

– `cyc = tail(cyc)`

B. `cyc = newedge \rightarrow cyc`

C. **if** (`destination(newedge) = u`) **then**

– `cyclefound = true`

D. **else for** each unmarked and undeleted edge, `tempedge`, such that `source(tempedge) = destination(newedge)` **do**

– **if** ($d'[destination(tempedge)] \leq$

$d'[source(tempedge)] + w(tempedge)$) and

(`tempedge = aps(source(tempedge), dest(tempedge))`) **then**

if (`destination(tempedge) $\neq u$`)

$d'[destination(tempedge)] =$

$d'[source(tempedge)] + w(tempedge)$)

`edgestoinset = tempedge \rightarrow edgestoinset`

(f) **if** (`cyclefound = true`) **then**

i. `all_cycles = temp_cycles \cup all_cycles`

ii. let, `minedge`, be the edge with the minimum priority on the cycle

iii. **if** (`minedge = e`) **then**

A. restore G' from `old_graph`

B. `insertion = marked`

C. `exit`

iv. **else if** (`minedge is a non-tree edge`) **then**

A. `minedge is marked` (i.e. removed from G')

B. let aps be the all_pairs_shortestpath matrix of G'

v. **else**

A. `minedge is marked` (i.e. removed from G')

B. SOLVE_WITHOUT_CYCLE_CHECK(G', w, s)

C. let aps be the all_pairs_shortestpath matrix of G'

D. **if** $d[v] \leq d[u] + w(e)$ **then**

`insertion = nontree; relaxed = true`

(g) **else**

i. `relaxed = true`

ii. $d = \text{copy_array}(d')$

iii. modifies the labels of the edges to reflect changes

A10 : DELETE_CONSTRAINT₁ (e)

1. $G' = (V, E - \{e\})$

2. unmark all edges

3. SOLVE_AND_MARK_CYCLE (G', w, s)

A11 : DELETE_CONSTRAINT₂ (e)

1. $G' = (V, E - \{e\})$ —all the marked edges

2. SOLVE_WITHOUT_CYCLE_CHECK (G', w, s)

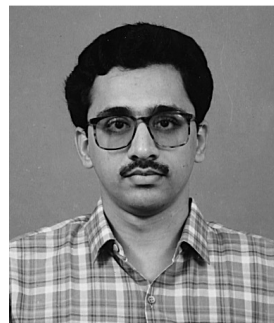
3. $E' = \text{SORT}(\text{all the marked edges}, p)$

4. **while** $E' \neq \perp$ **do**

(a) $e' = \text{head}(E')$

(b) $E' = \text{tail}(E')$

(c) INSERT_CONSTRAINT (e')



B. PRABHAKARAN received his PhD in Computer Science and Engineering from the Indian Institute of Technology, Madras, India. He is currently a faculty member in the Department of Information Systems and Computer Science at the National University of Singapore. He has been working in the area of network protocols and development of distributed multimedia applications for the past few years. His research interest includes high-speed networking issues for distributed multimedia presentations. He has published many research papers in

this field. He has also authored a book on Multimedia Database Management Systems. He is a member of the Editorial Board for the Journal of Multimedia Tools and Applications, Kluwer Academic Publishers.



KASIM SELÇUK CANDAN received his PhD in Computer Science from the University of Maryland at College Park in 1997 and currently is an Assistant Professor in the Department of Computer Science and Engineering at the Arizona State University. He is currently working on the CHIMP (Collaborative Heterogeneous Interactive Multimedia Platform) project. Although his main research interest is in multimedia document authoring/presentation/retrieval, he has also worked extensively on related topics, such as video indexing, image and video retrieval, video server scheduling, and heterogeneous information integration.

He has published various articles in respected journals and conferences in related areas. He received his B.S. degree in computer science from Bilkent University in Turkey in 1993. His research interests include multimedia databases and multimedia collaboration.



V.S. SUBRAHMANIAN received his PhD in Computer Science from Syracuse University in 1989 and currently is an Associate Professor in the Computer Science Department at the University of Maryland, College Park. He received the prestigious NSF Young Investigator Award in 1993. He has worked extensively in the fields of nonmonotonic reasoning, reasoning with inconsistency and uncertainty, databases, and in integrating multiple forms of data, software and reasoning paradigms. He has published over 65 papers in prestigious journals and conferences. He has also given invited talks and served on invited panels

at several leading conferences.