Collaborative Multimedia Documents: Authoring and Presentation*

Kasım S. Candan[†] **B. Prabhakaran**[‡] **V.S. Subrahmanian**[§]

Abstract

Multimedia documents are composed of different data types such as video, audio, text and images. Authoring a multimedia document is a creative exercise. Unlike traditional computer supported collaborative work where documents are composed of static objects, multimedia documents have temporal, spatial and quality of service (QoS) requirements that must be supported by any collaborative multimedia platform. In this paper, we show that most requirements (including temporal, spatial, and QoS requirements) for collaborative multimedia systems can be expressed in terms of a highly-structured class of linear constraints called *difference constraints* that have been well-studied in the operations research literature. As a consequence, well known algorithms for solving difference constraints may be used as a starting point for creating multimedia documents. Based on our difference-constraint based characterization, we develop efficient, incremental algorithms for creating and modifying multimedia documents so as to satisfy the required temporal, spatial and QoS constraints. We further develop methods to identify inconsistent requirements, and show how such inconsistencies may be removed through constraint relaxation techniques.

1 Introduction

A multimedia document typically consists of a number of media objects that must be presented to a person "reading" (or "viewing") the document in a coherent, synchronized manner. For example, a multimedia document on conservation of crocodiles in the Everglades may consist of an introductory audio-video 3 minute presentation laying out the background of the Everglades, a 1 minute automatically scrolling text window describing various affiliated projects, and a 5 minute

*This research was supported by the Army Research Office under grant DAAH-04-95-10174, by the Air Force Office of Scientific Research under grant F49620-93-1-0065, by ARPA/Rome Labs contract Nr. F30602-93-C-0241 (Order Nr. A716), and by an NSF Young Investigator award IRI-93-57756. Proofs of all results are contained in the appendix.

†Department of Computer Science, University of Maryland, College Park, Maryland 20742. Email: candan@cs.umd.edu.

‡Department of Computer Science, University of Maryland, College Park, Maryland 20742. Email: prabha@cs.umd.edu.

§Department of Computer Science, Institute for Advanced Computer Studies & Institute for Systems Research, University of Maryland, College Park, Maryland 20742. Email: vs@cs.umd.edu.

video clip of the animals themselves, accompanied synchronously by a voice recording describing the animals seen in the video-clip.

In general, a multimedia document is composed of a set of media objects, along with an associated set of presentation requirements, These requirements could include temporal and spatial requirements that have to be satisfied during the presentation. Furthermore, in the case of distributed multimedia document presentations, the network has to guarantee a minimum Quality of Service (QoS) for retrieving the required media objects from the appropriate document servers. These requirements, temporal, spatial and QoS, have to be described for a multimedia document.

The primary aim of this paper is to develop a mathematical framework that supports the creation and incremental modification of multimedia documents. We show that spatial, temporal, and QoS constraints can all be uniformly described within a small class of the language of real valued linear constraints. This class of constraints are referred to in the Operations Research literature as *difference constraints*. While generalized linear constraints [9] have the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \quad \leq \quad b \tag{1}$$

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

$$x_1 - x_2 \quad \leq \quad b. \tag{2}$$

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

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

Due to the fact that difference constraints have a very tightly restricted syntactic form, it turns out that they are very easy to solve. As space, time, and QoS constraints can all be described within the framework of difference constraints, this means that a constraint solver for difference constraints may be used to handle space, time and QoS constraints within a single unified implementation. Using difference constraints, we will show how it is possible to determine if a given set of media objects can be scheduled in a way that satisfies the desired constraints – if no such schedule exists, then this means that the performance criteria demanded by the authors of the document are inconsistent. Our algorithms will check for such inconsistencies. We will further show how an inconsistent set of constraints may be *relaxed* so as to restore consistency. We develop three different notions of *relaxations* and show that one is NP-complete, while the others are solvable in polynomial-time (and hence are perhaps more suitable for real world use).

Finally, any framework that supports collaborative multimedia authoring must be *incremental* as media objects may be modified dynamically by the author (or authors) of the media document. For example, during the development of a multimedia document, one of the collaborators may "check out" a video-clip and edit it. The result of this edit operation may increase the length of the video clip, thus invalidating a previous solution of the constraints governing this multimedia presentation. The revised set of constraints must be re-computed. We will develop an incremental algorithm for this purpose.

This paper is part of a long term project on developing support for collaborative multimedia systems, jointly between University of Maryland and University of California, San Diego. In our first paper on this topic [2], we developed techniques whereby objects could be routed across a network (and possibly transformed along the way) in such a way that the person (i.e. author) requesting the object received it at the lowest possible cost and at the desired quality. However, no issues regarding the presentation or editing of objects was considered there, just routing and communication. In this effort, we assume that network servers can route the object to the author in the form and within the quality desired; instead, we concentrate on how such a set of objects must be presented within a multimedia presentation with space, time, and QoS constraints.

2 Collaborative Multimedia Systems (COMS): Desiderata

In any collaborative multimedia system, four fundamental questions need to be addressed:

(Problem 1) Who may access a given media-object ?

(Problem 2) If collaborator C is allowed to access a given media-object, then what operations may he perform on it ?

(Problem 3) If collaborator C is allowed to access a given media-object, how should that object be routed to him across the network so as to ensure that: a) he can perform the operations he needs to perform, b) the cost of sending the object to him is minimized, and c) the object has the desired quality ?

(Problem 4) How should the collaborative multimedia system (COMS, for short) modify the overall document structure in the face of changes made to a media-object by a collaborator who is given write-access to that object? How should the COMS present only relevant parts of a multimedia document when a reader is only interested in certain portions of the document?

Questions (1) and (2) above have been studied intensely in the (distributed) operating systems arena where access methods and protocols for sharing files and file systems are well developed. Question (3) above was recently studied in detail by Candan, Subrahmanian and Venkat Rangan [2]. *The primary aim of this paper is to study question (4) above.*

In order to have a solution to this problem, the properties and the structure of the COMS environment needs to be identified. Unlike text based collaborative environments, in a multimedia environment, the authors work on a common document which consists of a variety of objects. There are *temporal* objects, such as video clips, that must be presented over a period of time, and there are *static* objects that do not have any temporal dimension. Similarly, there are objects that are *visible*, i.e. that should be displayed on the screen and hence have *spatial attributes*, and there are objects that are *not visible* on the screen (e.g. sound) that have no spatial attributes. Each of these different forms of data has very different characteristics, and different I/O requirements. It is the responsibility of the designer of the collaborative multimedia system to guarantee that such different forms of data are appropriately handled.

Furthermore, the multimedia objects constituting a multimedia document may be distributed over a computer network. This distributed aspect of the multimedia document poses some problems to a collaborative multimedia system. First, each user (or multimedia author) on the network may have different viewing and editing capabilities. Second, the network may not be homogeneous, and this heterogeneity may cause differences in communication requirements. Third, it is entirely possible that the temporal constraints (likewise spatial and QoS) associated with objects are mutually inconsistent. It is the job of the collaborative multimedia system to identify these conflicts and suggest techniques to resolve these conflicts.

Finally, when a document is being collaboratively authored, the communication facilities and the QoS offered by the network may lead to multiple alternative presentations of the document. Furthermore, both the authors editing the document as well as users viewing the document, may decide that they are only interested in a partial view of the document (e.g. only the portions that are modified by them in the case of authors, or in the case of users, the specific portions of direct interest to them). Similarly, an individual viewing the document may wish to skip viewing certain parts of the document. Hence, the author(s) and viewer(s) may share the same document and at the same time might see different presentations of it. The precise form of each presentation is based on the following factors:

- User access rights
- User projection of the document
- Communication and QoS requirements
- Local capabilities
- Authors editing the document.

Figure 1 shows, in detail, the structure of such presentations. In particular, the document shown at the top of Figure 1 may be presented in two different ways to two different users, depending upon the interests of those users, their access rights, the communication and QoS requirements based on the locations of those users, and the capabilities and facilities available at their local host. The following example demonstrates this.

Example 2.1 Figure 2 shows a very simple multimedia document which consists of three multimedia objects: $o1$, $o2$ and $o3$. The multimedia author(s) state the following presentation constraints:

- $o2$ should appear on the screen only after the presentation of $o1$ is completed.
- $o3$ must start at the same time as $o2$.

Now consider three users $u1$, $u2$, and $u3$ with the following facts and requirements:

- It takes 3 seconds for $u1$ to get $o1$ across the network, and it takes 5 seconds for $u1$ to get $o2$ and $o3$.

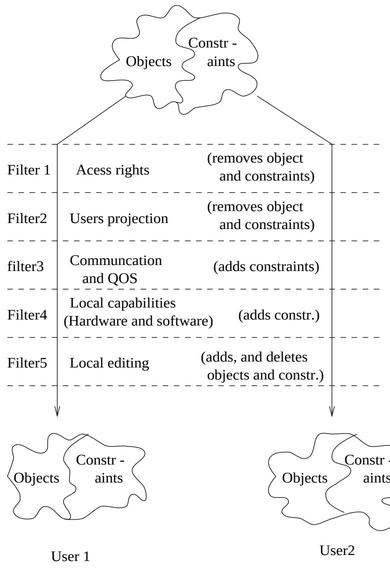

Figure 1: Multimedia system structure

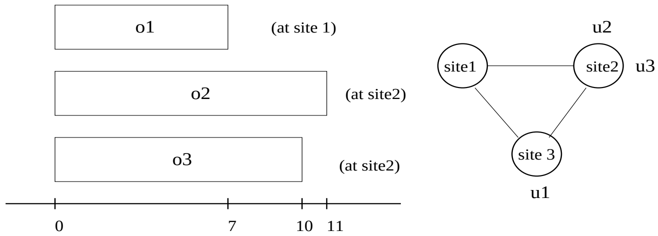

Figure 2: Duration of Objects in Example Multimedia Document

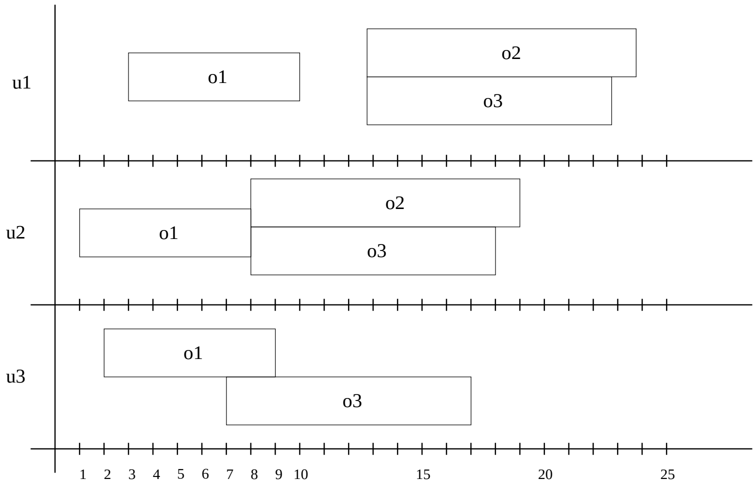

Figure 3: Different presentations of the same document.

- $u2$ has immediate access to objects $o2$ and $o3$, but it takes 1 second for him to get $o1$. Moreover, the user $u2$ wishes to start the presentation of object $o3$ within 8 seconds of the beginning of the presentation.
- $u3$ also has immediate access to the objects $o2$ and $o3$, and it takes 1 second to get $o1$. However, $u3$ does not want to see $o2$.

Figure 3 shows some possible realizations of this multimedia document for users $u1, u2$ and $u3$, respectively. It is easy to see that when object $o2$ is omitted from the presentation by $u3$, all related constraints are suppressed, thus leading to alternative renderings of the multimedia document. Hence, the display of $o3$ can overlap with the display of $o1$. □

2.1 Collaborative Authoring

In the distributed multimedia environment described above, there are two ways an author can edit a document. The first way is to edit the local view of the document. This is the *copy* of the presentation downloaded on the author's local site. In particular, this copy may differ from the "overall" document because the author's local presentation may differ from the overall document (cf. Example 2.1 above). The changes made on the local view will not be observed by the other readers of the document. Such local editing is useful in setting up one's own work space, and in adding tools and objects for local use. Note however that, such modifications on the local view of the document may be in conflict with the specifications of the original document. In some cases the user may want to adhere to the specifications of the original document, while in others, the user may wish to relax the specifications of the document in favor of his own modifications. Hence, there are two types of editing an author can perform on the local view of the document:

- **cautious edit:** make the change in the document unless the change conflicts with the document specifications,

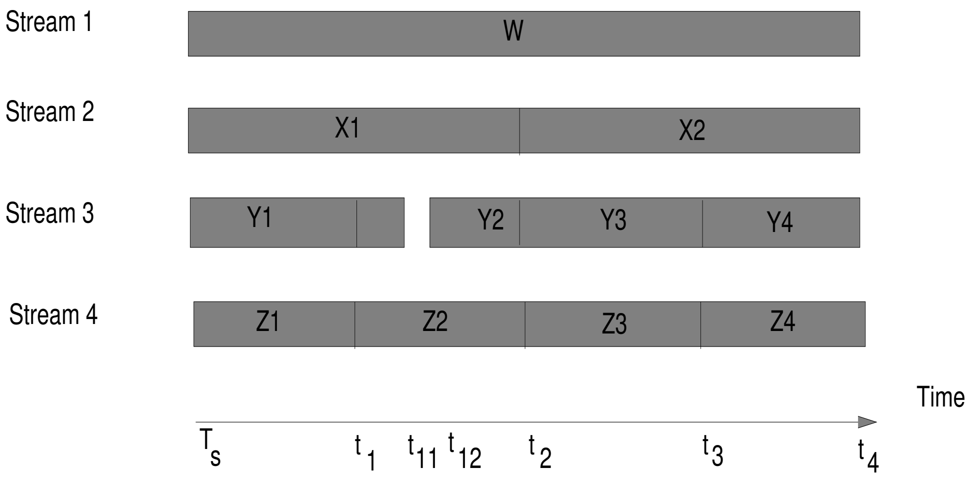

Figure 4: An Example Multimedia Document Structure

- **overwrite edit:** discard the specifications of the original document, if necessary, and perform the changes to the document.

The second way is to edit the original multimedia document itself. In order to edit the original document, the author must have the necessary access rights. Each author may have different access rights to different portions of the multimedia document. Furthermore, the authors may have priorities to resolve conflicts generated by the simultaneous editing of shared objects. The modifications performed on the original copy is immediately available to all users. Methods to study concurrent read-write accesses have been extensively studied in the operating systems community and database community[11], and hence we do not address these issues within the framework of this paper.

3 Multimedia Objects: Formal Definition

A multimedia document comprises of objects of different types such as video, audio, image and text. Each media object is presented in accordance with certain spatial and temporal constraints. The spatial constraints govern the presentation of media objects on the user's screen. The temporal constraints specify the time and duration of presentation of a media object, as well as the synchronization of presentation of with that of other media objects. Figure 4 shows a possible temporal structure of a multimedia document and Figure 5 shows an example spatial organization of the document. In this example, the document has four streams of information during the period $< T_s, t_4 >$:video, audio, image and text (T_s denotes the presentation start time). The spatial structure shown in Figure 5 maps each stream onto independent output devices (e.g. windows, speakers). However, in some instances, more than one stream may be mapped onto a window when object presentations are to be superimposed.

In many instances, the media objects composing the document can be distributed over a set

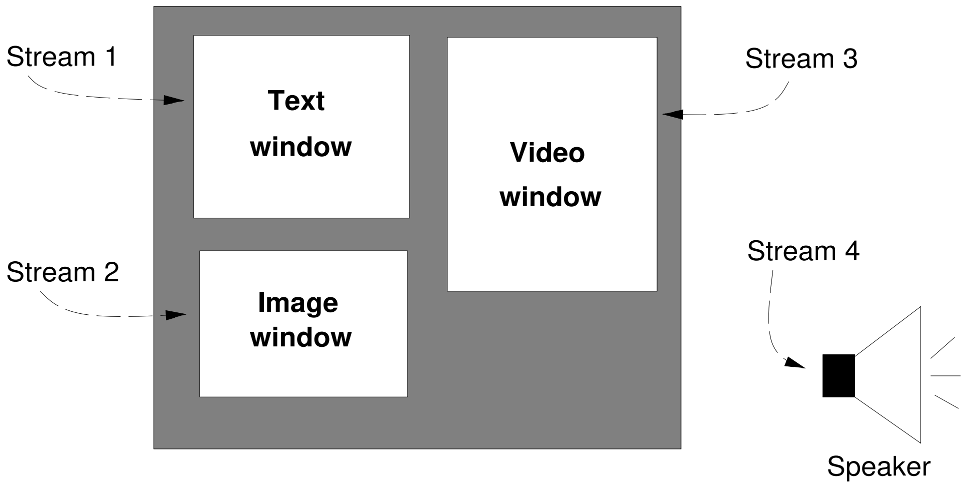

Figure 5: Spatial Structure of the Multimedia Document

of multimedia document servers. Hence, the media objects might have to be retrieved over a computer network during the presentation of a multimedia document. The necessity for retrieving the information imposes a Quality of Service (QoS) constraint that is to be satisfied by the network service provider. This QoS constraint depends on the structure of the multimedia document, the size of the objects composing the document and the associated temporal constraints.

A **multimedia object** O is a quadruple $O = < Name_O, Type_O, DispType_O, A_O >$ where:

1. $Name_O$: a string specifying the name of the object – we will assume without loss of generality that $Name_O$ and O are the same.
2. $Type_O$: each object must be declared as either a `static, quasi-static, or temporal` object. These are described in detail below.
3. $DispType_O$: each object must be declared to have a specific display type (e.g. `monitor, speaker`, etc.) specifying how the object is to be displayed. Note that display types can have sub-types as well. For instance, an object's display type may be `monitor:xv` specifying that the document is to be displayed on the monitor using `xv`. Similarly, `monitor:mpegplay` specifies that the object may be displayed on the monitor using the utility `mpegplay`.
4. A_O : This is a list of attributes of the object (e.g. size, quality/resolution, length, etc.) that may be of interest in an application.

Object type : Every multimedia object has one (and only one) associated type:

- **Static objects :** Static objects like *text* objects usually consist of one atomic component. There is no temporal dimension associated with such objects. A *gif* file is an example of a static object. Note that although there is no temporal aspect associated with a *gif* file, its presentation on the screen associates a temporal dimension to the object. In other words, static objects inherit the temporal dimension of the multimedia document to which they belong. The key aspect of static objects is that their display can not be divided into parts.

- **Quasi-static objects :** Some static objects consist of multiple pages (like postscript files). Each page can be considered as an atomic sub-object. However, the pages of a postscript document are linked with a strict *before-after* relationship. In fact, this relationship can be considered as a temporal dimension. However, the temporal dimension of a quasi-static object is stretchable, i.e. the display length of each sub-object may only be determined at run-time. This is because different readers may take different amounts of time scrolling through the pages of a postscript document. Variable-rate video objects may also be viewed as quasi-static objects. If O is any quasi-static object, we use the notation $len(O)$ to denote the number of atomic sub-objects that O has.

- **Temporal objects :** Some objects contain a predetermined number of atomic components, and a predetermined frequency for the display of these components. *Audio* objects and *fixed-rate video* objects are such objects.

Now that we have formally defined multimedia objects, we observe that a **multimedia document** D consists of:

1. A set, Obj_D, of objects,

2. For each object $O \in Obj_D$, a triple (T_O, S_O, Q_O) where T_O is a set of temporal constraints associated with object O, S_O is a set of spatial constraints associated with O, and Q_O is a set of QoS constraints associated with O.

3. A set of *Inter-Linking Constraints* governing the relationships between the presentations of different objects.

In the next few sections, we will study the precise structure of these constraints.

4 Specification of Temporal Constraints

Associated with each object O in a multimedia document D, we associate a set, T_O, of temporal constraints. As is customary in operations research[9], 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

- $ST_i(O)$: Denotes the start of the i^{th} component of object O (if O is a quasi-static or temporal object).
- $ET_i(O)$: Denotes the end of the i^{th} component of object O (if O is a quasi-static or temporal object).

Temporal constraints are defined for all objects, including static objects. There are four types of temporal 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. $\mathcal{T}(o) \in \{ST(o), ST_2(o), ET_2(o), \ldots, ST_{len(o)}(o), ET_{len(o)}(o), ET(o)\}$ and
2. $t \in \bigcup_j \{ST(o_j), ST_2(o_j), ET_2(o_j), \ldots, ST_{len(o)}(o_j), ET_{len(o)}(o_j), ET(o_j)\} \bigcup \{ST_p, ET_p\}$ and
3. ST_p and ET_p denote the start and end of the presentation respectively.

Recall that when O is a quasi-static object, $len(O)$ denotes the number of sub-objects of O.

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:

$$\begin{array}{c} ST(o_1) - ST(o_2) \leq 0 \\ ST(o_2) - ST(o_1) \leq 0 \\ ET(o_1) - ET(o_2) \leq 0 \\ ET(o_2) - ET(o_1) \leq 0 \end{array}$$

Note that using these constraints, not only we can specify Allen's 13 temporal relationships[1] between events (cf. Figure 6), but also specify more complex quantitative relationships that cannot be expressed in Allen's framework. For instance, in Allen's approach, it is possible to state that event A occurs before event B. However, it is not possible to say, for instance, that the completion of event A must precede the start of event B by at most 10 seconds and at least 5 seconds. In our framework, we can easily specify the temporal distance between two events.

Example 4.2 Let us reconsider the previous example, and suppose we also want the 5^{th} component of o_1 to be displayed at least 10 milliseconds after the 12^{th} component of o_2 is displayed. This requirement can be achieved by the addition of the following constraint to the above set:

	Multimedia Constraint	Specification
1	a before b	$ET(a) - ST(b) \leq \epsilon$
2	a equal b	$ST(b) - ST(a) \leq 0$ $ST(a) - ST(b) \leq 0$ $ET(b) - ET(a) \leq 0$ $ET(a) - ET(b) \leq 0$
3	a meets b	$ET(a) - ST(b) \leq 0$ $ST(b) - ET(a) \leq 0$
4	a overlaps b	$ST(a) - ST(b) \leq \epsilon$ $ST(b) - ET(a) \leq \epsilon$ $ET(a) - ET(b) \leq \epsilon$
5	a during b	$ST(b) - ST(a) \leq \epsilon$ $ET(a) - ET(b) \leq \epsilon$
6	a starts b	$ST(a) - ST(b) \leq 0$ $ST(b) - ST(a) \leq 0$ $ET(a) - ET(b) \leq \epsilon$
7	a finishes b	$ST(b) - ST(a) \leq \epsilon$ $ET(a) - ET(b) \leq 0$ $ET(b) - ET(a) \leq 0$

Figure 6: Allen's temporal relations (ϵ is a very small negative number)

	Multimedia Constraint	Specification
1	a should start when b starts	$ST(a) - ST(b) \leq 0$ $ST(b) - ST(a) \leq 0$
2	a should start 2 sec after b ends	$ET(b) - ST(a) \leq 2$ $ST(a) - ET(b) \leq -2$
3	a should start 2 sec before the end of the presentation	$ET_p - ST(a) \leq 2$ $ST(a) - ET_p \leq -2$
4	a should start within 3 seconds after the start of the 7^{th} frame of the object b	$ST(a) - ST_7(b) \leq 3$ $ST_7(b) - ST(a) \leq 0$
5	a should end within 2 seconds of the start of the $7th$ frame of the object b	$ST(a) - ST_7(b) \leq 2$ $ST_7(b) - ST(a) \leq 2$
6	a should be presented for 7 seconds	$ET(a) - ST(a) \leq 7$ $ST(a) - ET(a) \leq 0$
7	The second frame of a should start when the fifth frame of b ends	$ST_2(a) - ET_5(b) \leq 0$ $ET_7(b) - ST_2(a) \leq 0$

Figure 7: Some multimedia constraints and the corresponding specifications

$$ST_5(o_1) - ET_{12}(o_2) \geq 10$$

Figure 7 lists some examples of multimedia synchronization constraints that are difficult to express in Allen's framework, and shows how these may be easily represented in our approach.

In the 6^{th} row of figure 7, the second constraint, i.e. $ST(a) - ET(a) \leq 0$, obviously holds for each object in the multimedia document. Such constraints do not require explicit specification by the authors. A COMS system should automatically enforce the following constraints implicitly.

Implicit Temporal Constraints: For each multimedia object a, we have the constraint:

$$ST(a) - ET(a) \leq 0. \tag{3}$$

For each *temporal* or *quasi-static* multimedia object a, we have the following four constraints:

$$\begin{aligned} ET(a) - ET_{len(a)}(a) &\leq 0. && (4)\\ ET_{len(a)}(a) - ET(a) &\leq 0. && (5)\\ ST(a) - ST_1(a) &\leq 0. && (6)\\ ST_1(a) - ST(a) &\leq 0. && (7) \end{aligned}$$

These constraints merely specifying that the presentation of a quasi-static or temporal object begins (resp. ends) when its first (resp. last) atomic sub-object's display starts (resp. ends). In addition, for each *temporal* object a with fixed rate δt we have the following constraints:

$$\begin{aligned} ET(a) - ST(a) &\leq len(a) \times \delta t. && (8)\\ ST(a) - ET(a) &\leq -len(a) \times \delta t. && (9) \end{aligned}$$

In our framework, all the above constraints are enforced automatically without requiring an explicit specification by the user.

5 Specification of Spatial Constraints

Spatial constraints are defined for all objects in a multimedia presentation whose display types are set to *monitor*. In order to specify spatial constraints, we use the following *spatial variables*:

- $W(m)$ and $H(m)$ denote the width and the height, respectively, of the multimedia document.
- $X(m)$ and $Y(m)$ denote the coordinates of the lower left corner of the multimedia document on the screen.
- $x_r(o)$ and $x_l(o)$ denote the positions of the right and left borders of the multimedia object with respect to $X(m)$.

- $y_b(o)$ and $y_t(o)$ denote the positions of the bottom and top borders of the multimedia object with respect to $Y(m)$.

There are eight types of spatial constraints:

• $\mathcal{X}(o) - x \leq \delta x$	• $\mathcal{X}(o) - x \geq \delta x$
• $x - \mathcal{X}(o) \leq \delta x$	• $x - \mathcal{X}(o) \geq \delta x$
• $\mathcal{Y}(o) - y \leq \delta y$	• $\mathcal{Y}(o) - y \geq \delta y$
• $y - \mathcal{Y}(o) \leq \delta y$	• $y - \mathcal{Y}(o) \geq \delta y$

where:

1. $\mathcal{X}(o) \in \{x_r(o), x_l(o)\}$ and
2. $x \in \bigcup_j \{x_r(o_j), x_l(o_j)\} \bigcup \{W(m)\}$, and
3. $\mathcal{Y}(o) \in \{y_b(o), y_l(o)\}$ and
4. $y \in \bigcup_j \{y_b(o_j), y_l(o_j)\} \bigcup \{H(m)\}$.

The following example shows the use of the above constraints.

Example 5.1 Suppose a multimedia document contains two objects o_1 and o_2. Suppose we want the left border of o_1 to be 100 pixels from the left border of the multimedia document, and we want the left border of the o_2 be at most 10 pixels right to the right border of o_1. This arrangement can be described using the following constraints:

$$
\begin{gathered}
x_l(o_1) - 100 \leq 0 \\
100 - x_l(o_1) \leq 0 \\
x_r(o_1) - x_l(o_2) \leq 0 \\
x_l(o_2) - x_r(o_1) \leq 10
\end{gathered}
$$

□

Implicit Spatial Constraints: As in the case of temporal constraints, there are certain *implicit* spatial constraints that must always be honored by a COMS system. In particular, for all multimedia documents m and for all objects o the following constraints must be implicitly satisfied:

$$
\begin{aligned}
x_l(o) - x_r(o) &\leq 0 && (10) \\
y_b(o) - y_t(o) &\leq 0 && (11) \\
x_r(o) - X(m) &\leq W(m) && (12) \\
y_t(o) - Y(m) &\leq H(m) && (13)
\end{aligned}
$$

In our framework, these constraints will be automatically satisfied.

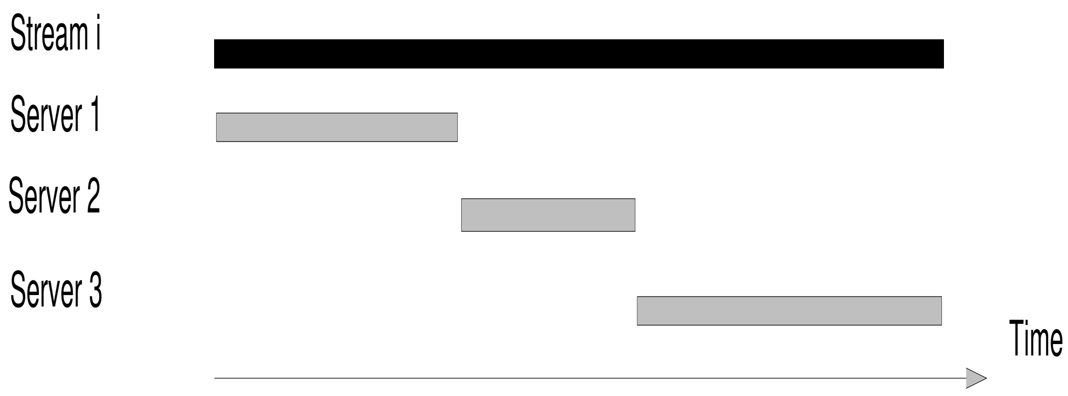

Figure 8: Network Sessions to Different Servers

6 Specification of QoS Constraints

The QoS required for an application is specified by a set of parameters such as throughput, delay, delay jitter and packet (or cell) loss probabilities. The QoS required for viewing (or editing) a multimedia document depends on the size of the various media objects and the time available for retrieving the objects i.e., their temporal constraints. Methodologies have been proposed in [17, 19] to identify the QoS requirements of a multimedia document. For a typical stream i in a multimedia document, the objects may be stored in different servers. In this case, the QoS requirements of the stream i has to be mapped onto the network requirements for connections to the different servers. Figure 8 shows an example where objects for a stream i are to be retrieved from three different servers.

Throughput Constraints Specification: In a distributed multimedia document presentation, the desired objects are retrieved from servers and then presented to the user. In such stored presentations, reliable existing network protocols may be employed for transferring the objects. Hence, the specification of throughputs is sufficient for describing the QoS requirements of a multimedia document system. In this section, we show how throughput requirements may be specified as constraints. It should be noted that the same methodology can be adopted for describing other QoS parameters such as delay, delay jitter and packet (or cell) loss probabilities.

Let us consider the throughput requirements for presenting the multimedia document. [17, 18, 19] presents methodologies to identify the throughput requirements for an orchestrated presentation. A similar methodology can be used to derive the throughput requirement of a multimedia document presentation. In order to describe the throughput constraints, we utilize the following *throughput variables*:

- $TH_o(t_i)$: Denotes the throughput required for retrieving the multimedia object o, at time t_i
- TH_{avg} : Denotes the average throughput required for a multimedia stream.

Here, the time instant t_i may be either:

- t_s : Denotes the start time of *retrieval* of the object. The retrieval time precedes the *presentation time* $ST(O)$, since the object is needed at the client side prior to its presentation, or

- t_e : Denotes the end time of *retrieval* of the object.

Using this definition, the throughput constraint for a multimedia document presentation may be defined as follows:

$$\boxed{\begin{array}{l} TH_o(t_s) - TH_{avg} \geq \delta h \\ TH_o(t_e) - TH_{avg} \geq \delta h \end{array}}$$

The above constraints describe the variations in the instantaneous throughput requirements for retrieving an object, from the average throughput required for the multimedia stream. These two constraints help in specifying the throughput requirements for retrieving a multimedia object o. By concatenating the throughput requirements of individual objects, we can identify the throughput requirements of a stream composing the multimedia document. The solution of these constraints gives the instantaneous throughput that must be guaranteed by the network service provider. This solution can be used for QoS negotiation with the network service provider.

Delay Constraints We now show how delay requirements may be modeled within our framework. Since the objects that are used in the multimedia document are not replicated at each node on the network, they need to be sent to the nodes when a user wants to use them. However, the communication between the nodes of the network imposes some delay, and this delay must be taken into account while scheduling the presentation of these objects. The following constraints represent the delay criteria:

$$\boxed{\begin{array}{l} T_{startdelivery}(o) - T_{enddelivery}(o) \leq -\delta. \\ T_{enddelivery}(o) - ST(o) \leq 0. \end{array}}$$

Suppose object o is needed by the node n and suppose the shortest delay that can be guaranteed by the network for the delivery of the object o to the node n is δ. This fact associated with node n is represented by the first constraint above. The second constraint above represents the fact that object o can not be presented before it is delivered.

These two constraints together specify the relationship between the start of the presentation of the object o and the start of the delivery of the object o.

Delay Jitter Constraints: Delay jitter constraints are specified as the maximum tolerable variations in the delay suffered by an object or a network packet [4, 5]. In order to specify the delay jitter constraints, we need to introduce the following variables:

- $D(o)$: Denotes the average delay suffered by an object on its transfer over the computer network.
- $D_{max}(o)$: Denotes the maximum delay that is permissible for the object transfer.

The delay jitter constraint may now be specified as follows:

$$D_{max}(o) - D(o) \leq \delta$$

which is easily seen to be a difference constraint.

Cell Loss Probability Constraints: The cell loss probability constraints describe the maximum percentage of cells (or the network packets) that can be *lost* during an object transfer [4, 5]. In order to specify the cell loss probability constraints, we need to introduce the following variables :

- $C(o)$: Denotes the number of network cells that an object o is composed of.
- $L(o)$: Denotes the number of network cells that were transferred to the client.

The cell loss probability constraint may now be specified as :

$$C(o) - L(o) \leq \delta \times 100$$

where δ is a *constant* specifying the maximum tolerable percentage of cell loss. It is easy to see that the above constraint is also a difference constraint.

Prior to completing this section, we observe that a vast number of important specialized QoS methodologies have been reported in the literature. Though such QoS constraints can be easily represented in our framework, they are not necessarily representable as difference constraints. Such complicated constraints, however, can be processed with suitable constraint solvers, and the result can be piped into our framework. The communication and cooperation between different constraint solvers is a difficult problem, and we do not address it in this paper.

7 Solving The Difference Constraints

By now, the reader would have noticed that *each and every temporal, spatial and QoS constraint associated with a multimedia document is a difference constraint* (cf. Sections 4, 5 and 6). Consequently, we may implement a single algorithm for solving difference constraints and call it with different inputs (corresponding, respectively, to the temporal, spatial, and QoS constraints associated with a multimedia document).

Before proceeding any further, we need to formally define a solution to a set of constraints. Though this definition is quite "obvious" it is needed for the proofs of the main results.

Definition 7.1 (Solution to a set of Constraints) Suppose C is a set of constraints C, and $v_1, \ldots, v_m$ are all the variables in C. A solution for C is a set $\sigma = \{v_1 = \gamma_1, \ldots, v_m = \gamma_m\}$ such that if we replace all occurrences of v_i in C with γ_1 for all $i = 1, \ldots, m$, then all the constraints in C evaluate to true. □

Handling Unsolvable Sets of Constraints: When a multimedia document is created by many different collaborators, each of whom may associate some presentation constraints with one or more objects, it is very likely that conflicts may arise. For instance, author A may have assumed that a given media object o_1 is of duration 5, but author B might edit o_1 and extend its duration to 8, thus leading to a violation of temporal constraints. In such cases, a COMS system must not only detect violation of the constraints, but it must also suggest ways of relaxing it. Consequently, we require a *conflict handler* to resolve the conflicts that might exist in (or that might arise during the course of refinement of) a specification of a multimedia document. Such a conflict handler should have the following properties.

- **Minimal discard:** The conflict handler should chose a minimal set of specifications to remove.

  Example 7.1 Assume the following set of specifications:
  $(1a)\, a - b \leq 0$
  $(2a)\, b - a \leq -1$
  $(3a)\, c - a \leq 0$
  $(4a)\, b - c \leq -1$

  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. □

- **Specification reuse:** The removed specifications must be retained by the COMS for possible future use, unless specified otherwise. For example, a COMS may have temporarily discarded constraint C. However, if the multimedia document is subsequently edited by the authors, then some of the conflicting constraints that caused C to be removed may themselves have been suppressed or modified, thus (possibly) allowing C to be consistent with the new set of constraints. The COMS should be capable of finding and reinserting the specifications which become realizable after such changes in the multimedia document.

  Example 7.2 Assume again the above set of specifications where (1a) is removed for keeping the set conflict free[1]:
  $*(1a)\, a - b \leq 0$
  $(2a)\, b - a \leq -1$

[1]the mark * denotes removal from the set of specifications.

$(3a)\, c - a \leq 0$
$(4a)\, b - c \leq -1$

Now assume that, the multimedia author deleted the specifications (2a) and (3a). The constraint (1a) is now satisfiable along with the remaining constraint (4a). Hence, (1a) should be reinserted to the specification list:

$(1a)\, a - b \leq 0$
$(4a)\, b - c \leq -1$ □

The specification reuse strategy is useful especially when there are conflicts between the document/system specifications and the user specifications, as well as when the user chooses to eliminate conflicts by deleting user specifications. The reason is that it is usually not desirable to omit conflicting user specifications, because they reflect how the user wants to view the document. Instead, the conflict handler should mark them as "currently" unsatisfiable, and retain them for possible reuse. If in the future, the document/system specifications change, then some of the user specifications may become satisfiable.

In the rest of this section, we will first (Section 7.1) define a data structure to store a set of difference constraints. As temporal, spatial and QoS specifications can all be captured by difference constraints, this data structure is adequate for reasoning about all these different forms of data. Later (Section 7.2), we will develop algorithms that check for solvability of these constraints, that automatically find ways of discarding minimal sets of constraints when constraints are unsolvable (i.e. when there is a conflict), and that automatically re-solve the constraint set when a new constraint is added/deleted (e.g. when a user edits a document).

In this paper, we will show that difference constraints associated with temporal, spatial and QoS constraints on presentation of a document, may naturally be represented as a weighted, directed graph in such a way that solutions of the constraints (which correspond to how the document must be presented in space, time, and quality) correspond to shortest paths in the graph.

7.1 Data Structure For Difference Constraints

Suppose D is any document and T_D, S_D, Q_D are the sets of temporal, spatial, and QoS constraints, respectively, that are associated with D. With each of these sets 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 (e.g.T_D, S_D, Q_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 edge is δt. Furthermore, for each node v_i, there is an edge from v_i to v_s with weight 0 and an edge from v_e to v_i with weight 0.

Thus, given any document D, we have one graph each associated with its temporal, spatial and QoS constraints. Suppose $\mathbf{C}$ is any cycle in graph G. $\mathbf{C}$ is said to be a *negative cycle* iff the sum of the weights of the edges in $\mathbf{C}$ is a negative number. The following result is an immediate consequence of the well known result ([3]) stating that a set of difference constraints is solvable iff the graph associated with it is free of negative cycles.

Theorem 7.1 Suppose D is any multimedia document and G_t, G_s, G_q denote the graphs associated with D and the constraint sets T_D, T_S, T_Q respectively. Then: T_D (resp. T_S, T_Q) is solvable iff G_t (resp. G_s, G_q) contains no negative cycle. □

Consequently, our framework for synchronized document authoring in COMS allows us to check for coherence/consistency of a multimedia document by merely checking whether a graph has a negative cycle. Optimal algorithms for this purpose were developed by Bellman and Ford [3].

We may use a *single unified data structure* for difference constraints to handle temporal, spatial and QoS constraints. This data structure is shown in Figure 9. We now describe the basic intuition underlying this data structure:

1. First, we have an array of object identifiers associated with objects occurring in the multimedia document.
2. Each entry (associated, say with object o) in the above array *points* to a node having three fields:
   (a) the first and second fields, S and E, are pointers that point to nodes $N1$ that refer to the "start" and "end" of presentation of the object.
   (b) The third field is a NIL pointer if object o is a static object. Otherwise, the third field points to a list of nodes of the form: $(s_i, e_i, Next)$ where each of s_i, e_i are similar to S and E above and refer to the start and end of the i'th component of the object in question and $Next$ points to the next element (if any) in the list.
   (c) Nodes of the form $N1$ above have a record structure containing five fields: *nodeid*, *count*, *value*, *inarcs*, *outarcs*. The *nodeid* field specifies a node in the graph associated with the multimedia document and a set of difference constraints (spatial, temporal or QoS). The *value* field specifies a time-instant at which the object (or a part of it) is displayed. The *count* field specifies how many sub-objects of the object must be displayed, starting from the aforementioned time instant. This field is always 1 if the object in question is static.
   (d) *inarcs* and *outarcs* specify the incoming edges to the node and the outgoing edges (w.r.t. the graph associated with the multimedia document and the associated difference constraints) from the node respectively. These arcs are defined as follows: at any given point in time t, the algorithm that we will define in Section 7.2, will associate with each node with node-id N, a set of edges, *inarcs* and (another set of edges *outarcs*) specifying the edges incident on (outgoing from) this node w.r.t. the edge relation in the graph associated with the multimedia document. Each such arc falls into one of three categories:

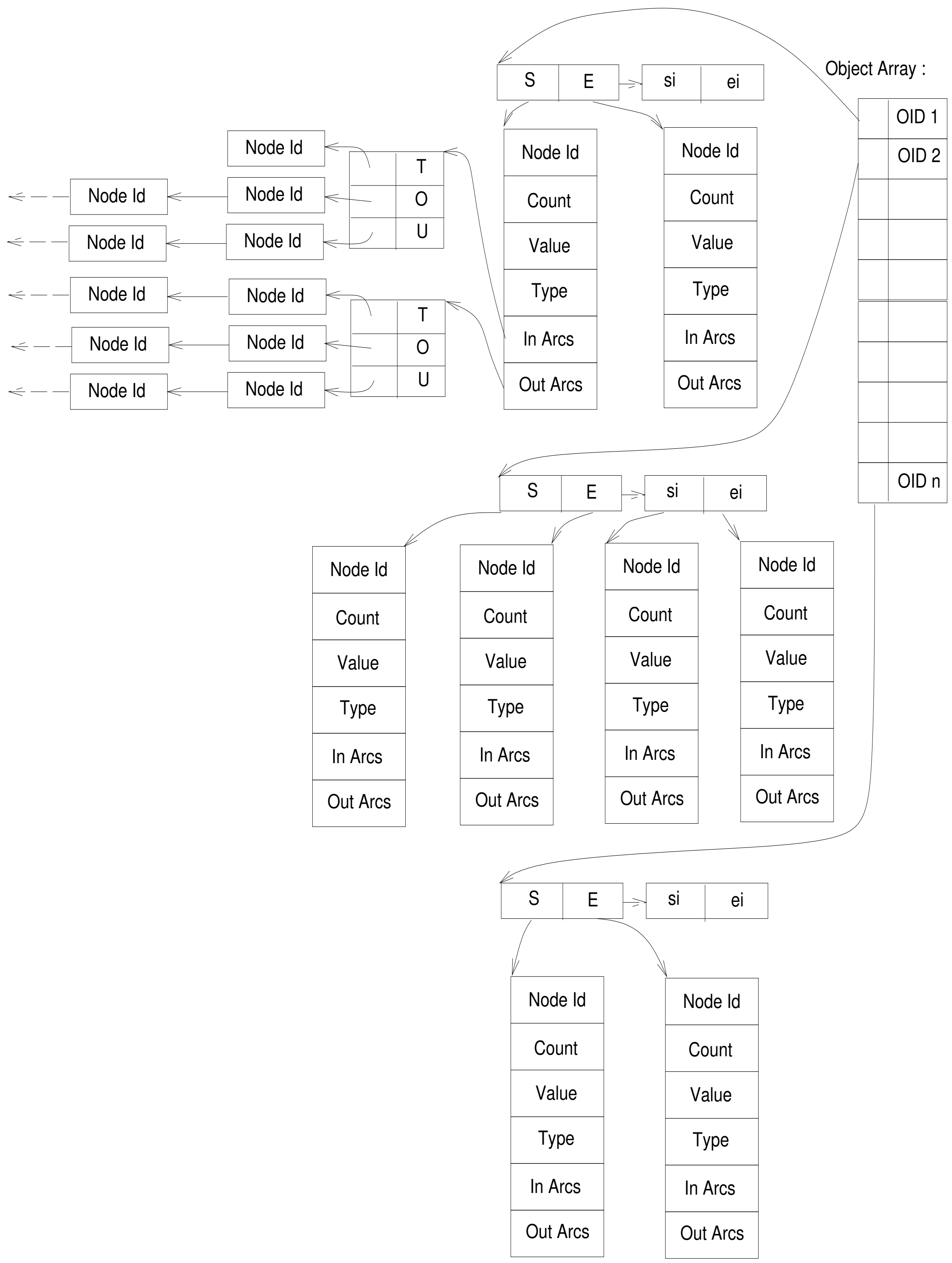

Figure 9: Difference Constraint Specification Data Structure

i. Node N has a list called T (strictly speaking we should write T_N, but will not do so when N is clear from context) that contains at most one arc from *inarcs* – intuitively, this arc represents the last segment of the "cheapest known path" from the start node to N found until now.

ii. Node N has a list called O – this is a list of "other" nodes in *inarcs* that do not occur in the shortest path known thus far; however if an update occurs (e.g. an author of the document modifies the document, thus increasing an edge weight/cost), then a node from the list O may be moved into T.

iii. Node N also has a list called I containing "inconsistent" edges. As stated in theorem 7.1, the existence of a negative cycle means that the constraints are unsolvable. To restore solvability, negative cycles must be "broken" by discarding a minimal number of constraints (policies that perform such "discards" will be discussed in detail in Section 7.2. For now, it suffices to know that all edges in *inarcs* that are associated with constraints[2] discarded in this way, are placed in I. The reason for their retention is the principle of Specification Reuse articulated earlier; later updates may very well make these discarded constraints satisfiable.

Example 7.3 (Difference Constraints Data Structure) Figure 10 shows a portion of the temporal constraint data structure for the objects $X1, X2, Y1, Y2, Y3$ and $Y4$ composing the multimedia document shown in Figure 4 earlier on in the paper. Each object has a *start* and an *end* node. The *value* associated with each of the nodes denotes the actual time of start or the end of presentation of the object. The *count* field specifies the number of sub-atomic objects composing the object.

For example, consider the object $X1$: the node id of the constraint variable $ST(X1)$ is 1, while the node id of the constraint variable $ET(X1)$ is 2. The count fields are 1 as $X1$ is an unbreakable image (hence, non quasi-static).

On the other hand, the object $Y2$ is a quasi-static video object (variable-rate video). This object has a total of 8 blocks of video – the first chunk shows 5 blocks, while the second consists of 3. The first chunk is shown starting at time $t1$, while the second chunk is shown starting at time $t12$. The display of the first chunk ends at time $t11$ while the display of the second ends at time $t2$.

7.2 Efficient Algorithms for Solving Difference Constraints

In this section, we are interested in the following problem: given a set of difference constraints (temporal, spatial, or QoS) associated with a multimedia document, attempt to solve this set of difference constraints. If no solution exists, then find a way of relaxing these constraints *minimally* so as to make the original unsolvable set of constraints solvable.

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 [3] in the manner described earlier on. Note that

[2] Recall that by the construction of the graph associated with a multimedia document, there is a one-one correspondence between those edges not involving v_s, v_e in the graph and constraints.

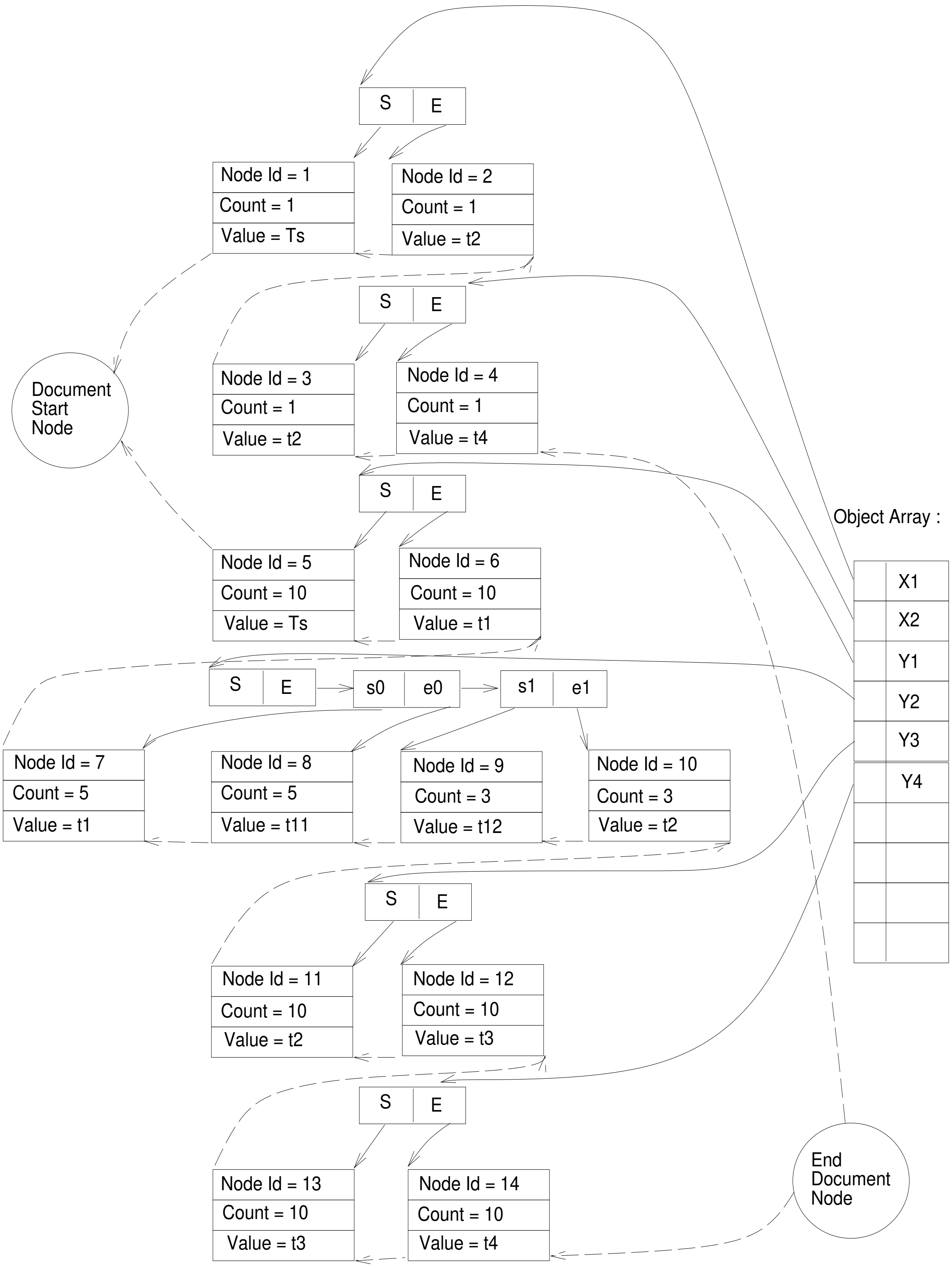

Figure 10: Example Temporal Constraint Data Structure

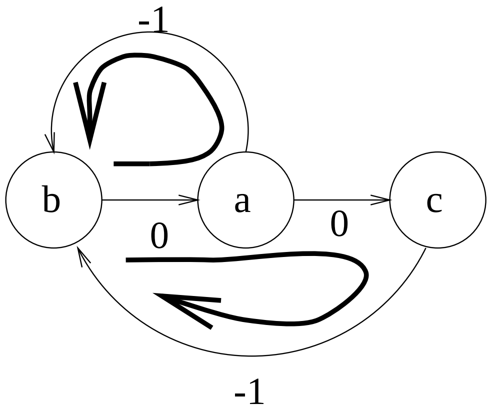

Figure 11: A constraint graph with negative cycles

constraint graphs can contain edges with negative weights. As proved in Theorem 7.1, conflicting constraints are captured by the existence of negative cycles in the constraint graph. For instance, the set of specifications in Example 7.1 corresponds to the graph in figure 11. Note that, the negative cycles in the graph correspond to the set of conflicts given in example 7.1.

Though most graph algorithms for computing shortest paths cannot handle negative cycles, the well known Bellman-Ford shortest path algorithm can deal with negative cycles in the graph [3]. This algorithm also detects the presence of a negative cycle that is reachable from the start of multimedia document presentation. If there is no cycle, the algorithm produces the shortest paths and their weights. The shortest path along with the associated weights in effect specify the time instances and durations of presentations of the objects composing the multimedia document. If there is such a cycle, the algorithm terminates indicating that there does not exist a solution. The presence of a negative cycle indicates conflicting constraints.

However, the Bellman Ford algorithm cannot relax constraints so as to restore solvability, nor can it even make suggestions to the authors of the multimedia document on how these constraints may be relaxed. *The primary aim of this section is to present an algorithm that will take as input, a set of (temporal/spatial/QoS) constraints, and return as output, a schedule that satisfies as many of the constraints as possible.* Before proceeding any further, we first define constraint relaxation.

Definition 7.2 (Constraint Relaxation) Suppose C is a set of difference constraints. A *relaxation* of C is any subset $C' \subseteq C$ such that C' is solvable. $\square$

The above definition allows any subset of C to be considered a relaxation. 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 relaxation of C eliminates "more constraints" than strictly needed. Below, we present three alternative definitions of *optimal constraint relaxations.* 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 7.3 (Optimal Constraint Relaxation) Suppose C is a set of difference constraints.

1. a *card-optimal relaxation* of C is any subset $C' \subseteq C$ such that C' is solvable, and there is no other relaxation C'' such that $card(C'') > card(C')$. Here $card(C)$ is the number of constraints in C.

2. a *pre-set-optimal relaxation* of C is any subset $C' \subseteq C$ such that C' is solvable, and there is no other relaxation C'' such that $C'' \supset C'$. A *set-optimal relaxation* of C is a *pre-set-optimal relaxation* C', and there is no other *pre-set-optimal relaxation* C" such that $card(C'') > card(C')$.

3. a *priority-optimal relaxation* of C is a *set-optimal relaxation* C' of C such there is no other *set-optimal relaxation* 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.

$\square$

As we will show below, finding card-optimal relaxations of C is an NP-complete problem , while finding a set-optimal relaxation is solvable in polynomial time. Similarly, finding Priority Optimal Relaxations is solvable in polynomial time.

Theorem 7.2 Card-optimal removal of negative cycles is NP-hard.

We now present an algorithm for finding solutions of both a priority-optimal relaxation of C and a set-optimal relaxation of a set C of constraints. In other words, the problem we address is the following:

Priority (resp. Set) Optimal Relaxable Constraint Problem:

- INPUT: A set C of difference constraints (temporal, spatial or QoS) associated with a multimedia document.
- OUTPUT: A solution σ to a priority-optimal (resp. set-optimal)relaxation of C.

7.3 Algorithms for Solving Optimal Relaxable Constraint Problem

Let $G = (V, E)$ be a weighted, directed graph associated with a set of difference constraints. Let the end document node v_e behave as the source with weight function $w : E \rightarrow R$. For each vertex $v \in V$, we maintain a variable, $d[v]$, representing an upper bound on the weight of a shortest path from source s_v to v, i.e., $d[v]$ is a *shortest path estimate*. We also maintain the predecessor, $\pi[v]$, of each node v. The predecessor of a vertex is either another vertex or NIL. shortest path and their associated weights. When our algorithm is finishes its computations, $d[S]$ where S is the start node will be a negative number. We will obtain a solution to a priority-optimal or set-optimal relaxation of the constraints being considered as follows:

SOL $= \{v = d[v] + d[S] \mid v$ is a vertex in the graph G that is neither the start nor the end node$\}$.

We now present a sequence of sub-routines **(A1)–(A5),(A8)**. **(A6)** and **(A7)** are the final algorithm that compute a solution to the optimal relaxable constraint problem. **(A6)** is used initially when a presentation of the document is being created for the first time. However, once a document presentation has been created, the more efficient algorithm, **(A7)**, may be used subsequently.

In the following algorithm, the shortest path estimates and the predecessors of each vertex are initialized first by the procedure *Initialize-single-source*. This procedure basically assigns NIL to $\pi[v]$, for all $v \in V$, $d[v] = 0$ for $v = v_s$, and $d[v] = \infty$ for $v \in V - \{v_s\}$.

A1 : Initialize-single-source (G, v_s)

for each vertex $v \in V[G]$
    do $d[v] \leftarrow \infty$
        $\pi[v] \leftarrow NIL$
$d[s] \leftarrow 0$
all_cycles $= \emptyset$

The identification of the shortest path is by using the *relaxation* routine, *RELAX*, where a test is carried out to check whether the shortest "current" path to a vertex v can be improved by an edge (u, v); if so, we update $d[v]$ and $\pi[v]$. Hence, a relaxation step might both decrease the value of the current shortest-path estimate $d[v]$ and simultaneously modify the predecessor $\pi[v]$.

A2 : RELAX (u, v, w)

if $d[v] > d[u] + w(u, v)$
    then $d[v] \leftarrow d[u] + w(u, v)$
        $\pi[v] \leftarrow u$

Besides the *RELAX* routine described above, we need a similar routine which will detect the existence of a negative cycle while modifying $d[v]$.

A3 : RELAX_and_MARK_CYCLE (u, v, w)

relaxed $= 0$
if $d[v] > d[u] + w(u, v)$
    then if NOT_CYCLE (u, v)
        then $d[v] \leftarrow d[u] + w(u, v)$
            $\pi[v] \leftarrow u$
            relaxed $= 1$;
return(relaxed)

The above algorithm **(A3)** relaxes an edge unless performing the relaxation leads to the creation of a negative cycle in the graph. The *NOT_CYCLE* routine checks if the relaxation of the edge will cause a negative cycle, and if there is a negative cycle, it marks the cycle for later processing.

A4 : NOT_CYCLE (u, v)

cycle = PATH (u, v)
if cycle $\neq \perp$
    then *cycle = cycle* $\rightarrow v$
        all_cycles = all_cycles $\cup$ {cycle}
        return(0)
    else return(1)

The *PATH* algorithm returns the shortest path from v to u found so far. If there is no such a path, then the algorithm returns $\perp$.

A5 : PATH (u, v)

if $\pi[u] = \perp$
    then return($\perp$)
if $\pi[u] = v$
    then return($v \rightarrow u$)
    else
        temp_path = PATH $(\pi[u], v)$
        if temp_path = $\perp$
            return($\perp$)
        else return(temp_path$\rightarrow u$)

The algorithm for determining the shortest-path uses the *Initialize-single-source* routine for initializing the shortest-path estimates and the predecessor for each vertex and then uses the algorithms given below to solve the constraints:

A6 : SOLVE_DIFFERENCE_CONSTRAINT_and_MARK_CYCLE (G, w, s)

Initialize-single-source (G, s)
for $i = 1$ **to** $|V[G]| - 1$
    do for each edge $(u, v) \in E[G]$
        do RELAX_and_MARK_CYCLE (u, v, w)
if all_cycles $\neq \emptyset$
    then G' = REMOVE_CYCLES (G, all_cycles)
        SOLVE_DIFFERENCE_CONSTRAINT_without_CYCLE_CHECK(G', w, s)

Theorem 7.3 If there are no negative cycles in the input constraint graph, then the algorithm SOLVE_DIFFERENCE_CONSTRAINT_and_MARK_CYCLE works in time O(V^2.E).

A7 : SOLVE_DIFFERENCE_CONSTRAINT_without_CYCLE_CHECK (G, w, s)

Initialize-single-source (G, s)
for $i = 1$ **to** $|V[G]| - 1$
    do for each edge $(u, v) \in E[G]$
        do RELAX(u, v, w)

The SOLVE_DIFFERENCE_CONSTRAINT_and_MARK_CYCLE algorithm first finds the negative cycles in the constraint graph. If no negative cycle exists, then the result is the shortest path. If, however, there are negative cycles in the graph, then the algorithm calls the REMOVE_CYCLES routine to get rid of the negative cycles, and then it calls SOLVE_DIFFERENCE_CONSTRAINT_without_CYCLE_CHECK to find the shortest path.

Theorem 7.4 SOLVE_DIFFERENCE_CONSTRAINT_without_CYCLE_CHECK works in time O(V.E).

The REMOVE_CYCLES algorithm given below eliminates cycles using a notion of *priority*. Suppose each edge in a constraint graph has a priority – the higher the priority, the more important the edge.

The elimination of a negative cycle requires the omission of at least one edge (constraint) from the negative cycle. The omission of a constraint can occur in two ways:

- **deletion of a constraint :** In this case, the constraint is permanently deleted from the constraint set.
- **marking of a constraint :** In this case, however, the constraint is kept within the constraint set, but marked as unsatisfiable. If, in the future, the inclusion of this constraint becomes safe (due to deletion of a conflicting constraint), then it can be unmarked.

In procedure **(A8)** below, all negative cycles are categorized by the system into two types – *sys_cycles* that the system will eliminate by itself, and *auth_cycles* that the system will present to the author(s) for their recommendations.

A8 : REMOVE_CYCLES (G, cycles)
1. Let cycles be (auth_cycles $\cup$ sys_cycles)
2. < `deleted_constraints`, `marked_constraints` > = CONSULT_AUTHORS (auth_cycles)
3. delete the constraints in `deleted_constraints` from the graph
4. mark the constraints in `marked_constraints` as unsatisfiable
{ At this point all the negative cycles in auth_cycles are removed }
5. let E be the set of edges in sys_cycles, and let $p(e)$ be the priority of the edge e
6. sort E with respect to the priorities in ascending order
7. remove duplicate negative cycles from sys_cycles
8. $e_m = 1$
9. **while** sys_cycles $\neq \emptyset$
10. **for** each unmarked edge $e \in E$ (starting from edge 1)
11. **do** $c[e]$ = COUNT_of_CYCLES(sys_cycles, e)
12. **if** $c[e] > c[e_m]$
13. **then** $e_m = e$
13. remove all the negative cycles containing e_m from sys_cycles
14. mark e_m as unsatisfiable

In the above algorithm the COUNT_of_CYCLES routine counts the number ($c[e]$) of negative cycles an edge e is involved. Note that, these counts must be recalculated during each iteration of the while loop.

Theorem 7.5 REMOVE_CYCLES works in time $O(E.V.C_s{}^2 + C_s.\log(C_s).V + E.\log(E) + C_a)$, where C_s is t he number of negative cycles in *sys_cycles*.

Theorem 7.6 If there are negative cycles in the input constraint graph, the algorithm SOLVE_TEMPORAL_CONSTRAINT_and_MARK_CYCLE works in time $O(V^2.E + E.V.C_s{}^2 + C_s.\ \log(C_s).V + E.\log(E) + C_a)$.

The REMOVE_CYCLES algorithm consults the multimedia authors for their preference about the negative cycles because these negative cycles embody constraints inserted by the authors of the multimedia document. For the negative cycles which contain system parameters, on the other hand, it may automatically decide which constraint to remove. The algorithm uses a greedy approach for removing the negative cycles. The main idea is to first remove edges involved in the highest number of negative cycles. In addition, it also tries to delay the removal of high priority edges as much as possible. The REMOVE_CYCLES algorithm is guaranteed to:

- always compute a shortest path in the constraint graph associated with a *priority-optimal* relaxation of C; and
- as a consequence, when the REMOVE_CYCLES terminates, `SOL` $= \{v = d[v] + d[S] \mid v$ is a node in the constraint graph and v is not the start or end node$\}$ is a solution to a priority optimal relaxation of C (resp. set-optimal relaxation of C if all priorities are set to 1).

We view the multimedia document as a dynamic entity which dynamically changes with the addition and deletion of objects and constraints by the authors of the document. The changes to the document may be initiated by the multimedia authors or by changes in the system parameters and the resource availability (e.g. changes in the expected throughput).

When changes occur due to the addition/deletion of objects, these are captured within the existing presentation schedule for the multimedia document as changes to the *constraints* governing the presentation of those objects. For example, when a new object o is added to a presentation and we want to "present" o immediately after an existing object o_1 and immediately before another existing object o_2, then this affects the existing presentation by the addition of new constraints involving this object. It is therefore easy to see that the insertion/deletion of objects into/from an existing presentation is captured by the addition/deletion of constraints. In the next two subsections, we present techniques from incremental updates of presentations based on the introduction of new constraints and/or the deletion of existing constraints.

7.4 Incremental Addition of Difference Constraints

The easiest way of handling the addition of a new constraint would be to use the shortest path algorithm described above. However, in a multimedia system where there are many dynamic changes, or in a system where there are hard deadlines for the presentation, this may not be the best approach. Hence, in this chapter we present an algorithm which dynamically computes the new presentation schedule given an existing solved set of constraints, and a new constraint to be added. Recall that the insertion of a constraint into a constraint set is equivalent to the insertion of an edge into the graph associated with that constraint set.

```
INSERT_CONSTRAINT (e)
let e be from u to v with weight w
insertion = normal
relaxed = RELAX_and_FIND_CYCLE (u, v, w)
if all_cycles ≠ ∅
        then G' = REMOVE_CYCLES (G, all_cycles)
                if only e is marked/deleted
                        then insertion = marked (or deleted)
                if there is an edge f (other than e) which is marked/deleted
                        then DELETE_CONSTRAINT(f)
                                insertion = normal
if insertion≠ deleted
        then if relaxed = false
                then insert e as a normal/marked non-tree edge to the graph
                else mod = {v}
                        while mod ≠ ∅
                                do let i be a node in mod
                                        mod = mod − {i}
                                        for each edge k = (i, j, w2)
                                                relaxed2 = RELAX (i, j, w2)
                                                if relaxed2 = true
                                                        then mod = mod ∪ {j}
```

7.5 Deleting Difference Constraints

When there are no marked constraints waiting to be reinserted, the deletion of a constraint is easy to handle: a solution to the original set of constraints is a solution to the modified set of constraints. Hence, in this case, a constraint can be deleted from the graph in O(1) time.

However, the existence of marked constraints makes the problem much harder. These are constraints that were previously deleted (perhaps due to some negative cycles that caused an inconsistent set of constraints), but were saved just in case future changes invalidated the cause of the inconsistency. In order to reinsert marked constraints, we first need to first incorporate the effects of the deleted constraint from the document. Only after that can one safely reinsert a constraint to the graph. We have developed two algorithms, **DELETE_CONSTRAINT$_1$** and **DELETE_CONSTRAINT$_2$** that handle such constraint deletions. They are described below:

The first algorithm is quite simple – it merely deletes the edge in the constraint graph associated with the constraint being deleted and re-applies the algorithm for computing priority optimal relaxations of the graph.

DELETE_CONSTRAINT$_1$ (e)
1. $G' = (V, E - \{e\})$
2. SOLVE_DIFFERENCE_CONSTRAINT_and_MARK_CYCLE (G', w, s)

Theorem 7.7 The running time of the DELETE_CONSTRAINT$_1$ algorithm is O(V^2.(E-1) + (E-1).V.C$_s$ + C$_s$.log(C$_s$).V), where C_s is the number of negative cycles in *sys_cycles* in the graph.

The second algorithm, on the other hand, takes a "bottom-up" approach. It first eliminates from the graph G the edge associated with the constraint being deleted as well as all marked edges. It then attempts to use the "Incremental Addition" algorithm **INSERT_CONSTRAINT**, to re-insert the marked edges in order of priority.

DELETE_CONSTRAINT$_2$ (e)
1. $G' = (V, E - \{e\} - \textit{all the marked edges})$
2. SOLVE_DIFFERENCE_CONSTRAINT_without_CYCLE_CHECK (G', w, s)
3. $E' = \text{SORT}(\textit{all the marked edges}, p)$
4. **while** $E' \neq \perp$
5. $\quad e' = head(E')$
6. $\quad E' = tail(E')$
7. $\quad$ INSERT_CONSTRAINT (e')

Theorem 7.8 The running time of the DELETE_CONSTRAINT$_2$ algorithm is O(V^2.(E-C$_m$-1) + C$_m$.log(C$_m$) + C$_m$.E), where C_m is the number of marked edges in the graph.

As can be seen from the complexity results, which of these two algorithms will perform better depends on the specific constraint graph being considered and the edge being deleted. A hybrid deletion algorithm would first evaluate the two quantities:

- $QTY1$ = V^2.(E-1) + (E-1).V.C$_s$ + C$_s$.log(C$_s$).V
- $QTY2$ = V^2.(E-C$_m$-1) + C$_m$.log(C$_m$) + C$_m$.E

It would then use **DELETE_CONSTRAINT$_1$** if $QTY1 < QTY2$ and **DELETE_CONSTRAINT$_2$** otherwise.

8 Related Work

In this paper, we have given a formal definition of a multimedia document, and presented a single difference constraint-based model using which, temporal, spatial, and QoS constraints may all be expressed within a unified framework. The advantage of this is that a "core" set of algorithms, such as those provided in Section 7.2, may be used to create, and maintain, the presentations of multimedia documents, as changes are made to the document by its authors. Furthermore, these algorithms are provably correct and their complexity has been analyzed and proved to be always polynomial-time.

Ahuja's group at AT&T [7] also has had significant contributions in collaborative services. They propose a method for generating visual representations of recorded histories of distributed collaborations, so that remote collaborators can easily access information that will let them understand how the collaborative environment evolved to a particular state. In [10], Imai et al. show how to record the artifacts of a realtime collaboration so that when the collaboration is concluded, the collaborators have access not only to the final document, but also to the artifacts (handwritten notes, voice annotations etc.) that led them to this document. Using our work in conjunction with these two works to maintain *versions* of documents as they are altered over a period of time.

Gong [8] studies some of the important issues in multimedia conferencing over packet switched networks, and provides solutions to the problems that arise in multipoint audio and video control. The Argo system [6] on the other hand, is built to let users collaborate remotely using video, audio, shared applications, and whiteboards. Wolf et al [24] show how an application can be shared among heterogeneous systems. They compare two methods for heterogeneous sharing: one optimizes transmission in the system and other optimizes conversions between objects. Candan et. al. [2] develop a formal framework within which objects may be routed and transformed from one network node to the site of an author in such a way that the desired quality is maintained and the author's host machine capabilities are adequate to process the object. All these efforts target one or more aspects of Problem (3) specified in Section 2. In contrast, in this paper, we address the complementary problem (Problem 4 of Section 2) – here we try to develop optimal ways of *presenting* documents to users in the face of constantly changing specifications. Furthermore, we develop techniques that will optimally relax the presentation constraints in the event that these constraints are inconsistent. Finally, our framework applies uniformly not just to temporal aspects of multimedia systems, but to spatial and QoS as well.

Little [23] has presented an elegant document management system for shared data and provided a data model (POM) which permits dynamic compositions of mixed-media documents. Wray et al. [25] have built an experimental collaborative environment called Medusa which integrates data from heterogeneous hardware devices. Medusa provides an environment which facilitates rapid prototyping of new applications. Rajan, Vin et al. [20] started some work on formalizing the notion of multimedia collaboration. They provide a basis which can support a wide spectrum of structured multimedia collaborations. Their formalization captures the requirements of various types of interactive and non-interactive collaborations. They also implemented a prototype collaboration management system based on their formalism.

Significant contributions have been made in the area of temporal specification of multimedia presentations. Petri nets based models have been suggested in [13, 16, 19, 17] for specifying the temporal and synchronization characteristics of a multimedia presentation. Concurrent programming language based approach has been suggested in [21]. A Context Free Grammar based approach has been proposed in [19] for describing the synchronization characteristics of an orchestrated presentation and for translating the characteristics into the network traffic that might be generated by an orchestrated presentation. In [15] user views of a document are represented by means of attribute based selection of a Petri nets based specification. However, these works do not address the issues that arise in an collaborative environment. Also, the specifications of the requirements are *fixed* in nature. Synchronization has also been studied by Manohar [14]. They study methods to enable the faithful replay of multimedia objects under varying system parameters. To accomplish synchronization of different session objects, they provide an adaptive scheduling algorithm. In [12], a Time-flow Graph (TFG) model has been proposed to represent "fuzzy" or imprecise temporal relationships. Multimedia objects are described by their presentation intervals. Given any two time intervals, there are thirteen ways in which they can be related. In the TFG model, temporal relationships can be specified in terms of temporal durations despite the lack of duration information about the involved intervals. In contrast to these efforts, we have provided a unified treatment of different types of constraints governing multimedia documents. We have developed, for the first time, optimal ways of *presenting* documents to users in the face of constantly changing specifications and techniques that will optimally relax the presentation constraints in the event that these constraints are inconsistent.

9 Conclusions

A collaborative multimedia system (COMS) must support a wide range of functionalities so as to enable a set of cooperating authors to jointly create a multimedia document. In order to support the construction of COMS systems, we have provided a formal, mathematical definition of a *multimedia document* as a set of media objects that are constrained to be presented according to certain spatial, temporal and QoS criteria. We have shown that all these criteria may be expressed mathematically using a small class of constraints well known in operations research called *difference constraints.* Thus, difference constraints provide a unifying framework within which different aspects of creating multimedia presentations may be studied.

As multimedia documents are typically constructed over a period of time, and as the objects constituting such a document are edited by different people over time, both the set of objects in a document, and the set of constraints linking these objects together, will change with time. We have developed incremental algorithms that will:

1. determine if such constraints are solvable, and
2. incrementally find a new solution to a set of constraints when some new constraints are added, and
3. incrementally find a new solution to a set of constraints when some old constraints are deleted, and

4. develop algorithms that will compute *optimal relaxations* of a set of constraints (according to different notions of optimality) when a set of constraints is not solvable.

Furthermore, these algorithms work independently of whether temporal constraints, spatial constraints, or QoS constraints are being considered.

This paper is part of a joint effort between the University of California, San Diego. In our first paper on this topic [2], we developed techniques whereby objects could be routed across a network (and possibly transformed along the way) in such a way that the person (I.e. author) requesting the object received it at the lowest possible cost and at the desired quality. In future work, we will study the interactions between spatial constraints, QoS constraints and temporal constraints – in particular, we will study the problem of how spatial temporal constraints are affected by changes in the QoS constraints and vice versa.

Acknowledgements. We are grateful to Sibel Adali, Eenjun Hwang, and Charlie Ward for making useful comments on the paper, and to Venkat Rangan for conversations on the topic of this paper.

References

[1] J.F. Allen. (1984) *Towards a General Theory of Time and Action*, Artificial Intelligence, 23, pps 123–154.

[2] K.S. Candan, V.S. Subrahmanian, and P. Venkat Rangan. (1995) *Collaborative Multimedia Systems: Synthesis of Media Objects*, Submitted for publication, Nov. 1995.

[3] T.H. Cormen, C.E. Leiserson and R.L. Rivest, "Introduction to Algorithms", McGraw Hill Publishers.

[4] D. Ferrari, 'Client Requirements For Real-Time Communication Services', IEEE Communication Magazine, Vol. 28, No. 11, Nov. '90, pp. 65-72.

[5] D. Ferrari, J. Ramaekers and G. Ventre, 'Client-Network Interactions in Quality of Service Communication Environments', Proc. of High Performance Networking, 4th IFIP Conf. on High Performance Networking, Liege, Belgium, 14-18, December '92.

[6] H. Gajewska, "Argo: A System for Distributed Collaboration", ACM Multimedia 94, Pages 433-440.

[7] A. Ginsberg and S. Ahuja, "Automating envisionment of virtual meeting room histories", ACM Multimedia 95, Pages 65-76.

[8] F. Gong, "Multipoint Audio and Video Control for Packet-Based Multimedia Conferencing", ACM Multimedia 94, Pages 425-432.

[9] F. Hillier and G. Lieberman. (1974) *Operations Research*, Holden-Day.

[10] T. Imai, K. Yamaguchi, T. Muranaga, “Hypermedia Conversation Recording to Preserve Informal Artifacts in Realtime”, ACM Multimedia 94, Pages 417-424.

[11] H. Korth and A. Silberschatz. (1986) “Database System Concepts”, McGraw Hill.

[12] L. Li, A. Karmouch and N.D. Georganas, “Multimedia Teleorchestra With Independent Sources : Part 1 and Part 2”, ACM/Springer-Verlag Journal of Multimedia Systems, vol. 1, no. 4, February 1994, pp.143-165.

[13] T.D.C. Little and A Ghafoor, ‘Synchronization and Storage Models for Multimedia Objects’, IEEE J. on Selected Areas of Communication, vol. 8, no. 3, April 1990, pp. 413-427.

[14] N.R. Manohar and A. Prakash, “Dealing with synchronization and timing variability in the playback of interactive session recordings”, ACM Multimedia 95, Pages 45-56.

[15] N. Pahuja, B.N. Jain and G.M. Shroff, ‘Multimedia Information Objects: A Conceptual Model for Representing Synchronization’, to appear in International Conference on Computer Networks, Networks’96, Bombay, India, January 1996.

[16] B. Prabhakaran and S.V. Raghavan, ‘Synchronization Models For Multimedia Presentation With User Participation’, ACM/Springer-Verlag Journal of Multimedia Systems, vol.2, no. 2, August 1994, pp. 53-62. Also in the Proceedings of the First ACM Conference on MultiMedia Systems, Anaheim, California, August 1993, pp.157-166.

[17] S.V. Raghavan, B. Prabhakaran and Satish K. Tripathi - ‘Synchronization Representation and Traffic Source Modeling in Orchestrated Presentation’, to appear in the special issue on Multimedia Synchronization, IEEE Journal on Selected Areas in Communication.

[18] S.V. Raghavan, B. Prabhakaran and Satish K. Tripathi - ‘Quality of Service Considerations For Distributed, Orchestrated Multimedia Presentation’, Proceedings of High Performance Networking 94 (HPN’94), Paris, France, July 1994, pp. 217-238. Also available as Technical Report : CS-TR-3167, UMIACS-TR-93-113, University of Maryland, College Park, Computer Science Technical Report Series, October 1993.

[19] S.V. Raghavan, B. Prabhakaran and Satish K. Tripathi - ‘Handling QoS Negotiations in Distributed Orchestrated Presentation’, to be published in Journal of High Speed Networking.

[20] S. Rajan, P.V. Rangan, and H.M. Vin, “A Formal Basis for Structured Multimedia Collaborations”, IEEE Intl. Conf. on Multimedia Computing and Systems, 1995.

[21] R. Steinmetz, ‘Synchronization Properties in Multimedia Systems’, IEEE J. on Selected Areas of Communication, vol. 8, no. 3, April 1990, pp. 401-412.

[22] P.D. Stotts and R. Furuta, ‘Temporal Hyperprogramming’, Journal of Visual Languages and Computing, Sept. 1990, pp. 237-253.

[23] T.M. Wittenburg and T.D.C. Little, “An Adaptive Document Management System for Shared Multimedia Data”, IEEE Intl. Conf. on Multimedia Computing and Systems, 1994, Pages 245-254.

[24] K.H. Wolf, K. Froitzheim and P. Schulthess, "Multimedia Application Sharing in a Heterogeneous Environment", ACM Multimedia 95, Pages 57-64.

[25] S. Wray, T. Glauert, and A. Hopper, "The Medusa Applications Environment", IEEE Intl. Conf. on Multimedia Computing and Systems, 1994, Pages 265-274.

10 Appendix: Proofs of Results

Proof of Theorem 7.2. Suppose $NC = \{nc_1, \ldots nc_k\}$ denote the negative cycles in the constraint graph G. Suppose also that each negative cycle nc_i has the form $nc_i =< e_1, \ldots .e_{k(i)} >$, where e_j denotes an edge, and $k(i)$ denotes the number of edges involved in negative cycle nc_i. We will prove that card-optimal removal of negative cycles is NP-hard by reducing the vertex-cover problem to the card-optimal removal of negative cycles problem.

A vertex cover of an undirected graph $G_{in} = (V_{in}, E_{in})$ is a set $V' \subseteq V_{in}$ such that if $(u, v) \in E_{in}$, then $u \in V'$ or $v \in V'$ or both. The vertex-cover problem is to find a vertex cover of minimum cardinality in a given graph G_{in}.

Reduction of vertex-cover problem into card-optimal removal of negative cycles problem: We are going to create, in polynomial time, a constraint graph G from G_{in} such that, if we can find a card-optimal relaxation of the difference constraints associated with the edges of G, then we can also find a minimal cover of G_{in} in polynomial time.

Let $card(V_{in})$ be CV and let $card(E_{in})$ be CE. Furthermore, let the vertices in G_{in} be $\{v_1, \ldots, v_{CV}\}$ and let the edges in G_{in} be enumerated as $\{e_1 =< f_{1:1}, f_{1:2} >, e_2 =< f_{2:1}, f_{2:2} >, \ldots, e_{CE} =< f_{CE:1}, f_{CE:2} >\}$, where $f_{i:1}$ and $f_{i:2}$ are vertices in V_{in}. The reduction works as follows:

$V = \emptyset$; $E = \emptyset$
for $z = 1$ to CV do
    create two vertices $v'_{z:1}$ and $v'_{z:2}$
    create an edge $e'_z =< v'_z : 1, v'_{z:2} >$ with 0 weight
    $V = V \cup \{v'_{z:1}, v'_{z:2}\}$
    $E = E \cup \{e'_z\}$
for $y = 1$ to CE do
    /* let e_y be $< v_i, v_j >$ */
    create a directed edge $e^{\circ}{}_y =< v'_{i:2}, v'_{j:1} >$ with 0 weight
    create a directed edge $e^{\diamond}{}_y =< v'_{j:2}, v'_{i:1} >$ with -1 weight
    $E = E \cup \{e^{\circ}{}_y, e^{\diamond}{}_y\}$

The result of the algorithm is a weighted directed graph $G = (V, E)$. Note that, the algorithm works in polynomial time.

Claim 1: If G_{in} has a minimal cover of size s, then G has a card-optimal relaxation of size s: Each vertex v in G_{in} corresponds to an edge in G, and each edge in G_{in} corresponds to a negative cycle in G. Furthermore, if two edges e_1 and e_2 in G_{in} share a vertex v, then the corresponding negative cycles on G share the corresponding edge. Hence, if there are s vertices that cover the edges in G_{in}, then there are s edges that cover the negative cycles in G. If these negative edges are deleted from the graph G, then G will be negative-cycle free.

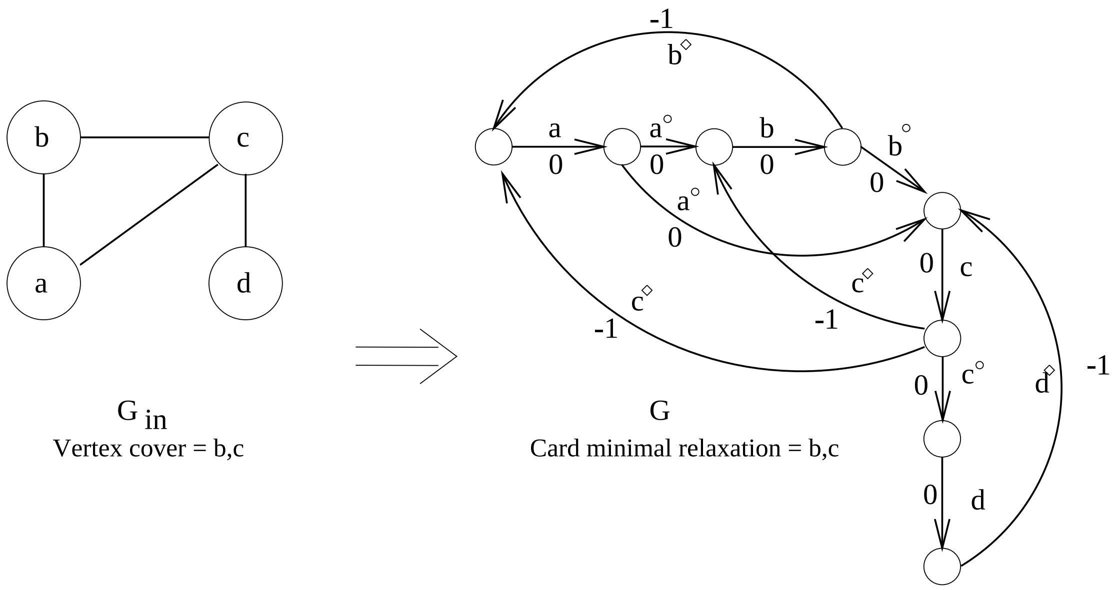

Figure 12: An example reduction from G_{in} to G.

Claim 2: If G can be card-optimally relaxed by the removal of s edges, then G_{in} has a minimal vertex cover of size s: The removed edges will not contain any edge marked with superscript $^\circ$ or $^\diamond$, because these edges cannot be shared among two negative cycles, hence it is advantageous to remove these edges (the only exception is when the negative cycle does not share an edge with any other negative cycle, and hence the edges marked with $^\circ$ or $^\diamond$ can also be chosen for removal. However, since in this case the negative cycle is independent of the other negative cycles, we can assume that the corresponding non-superscripted edge will be chosen for removal.) If it is possible to remove all the negative cycles, by removing s edges from the graph G, then it is possible to cover all the edges in G_{in} by s vertices, because each vertex v in G_{in} corresponds to an edge in G, and each edge in G_{in} corresponds to a negative cycle in G.

Using the above claims and the polynomiality of the reduction algorithm, we can conclude that the card-optimal relaxation problem is NP-hard. To see how the reduction works, consider figure 12. □

Proof of Theorem 7.3. If there are no negative cycles, then *all_cycles* will be $\emptyset$, and the REMOVE_CYCLES and SOLVE_DIFFERENCE_CONSTRAINT_without_CYCLE_CHECK routines will not be called.

Initialize-single-source works in O(V). The routine RELAX_and_MARK_CYCLE is called V.E times. The worst case running time of RELAX_and_MARK_CYCLE is V (because it may need to check all vertices to see if it is on the path). Hence, if there is no negative cycle, then the running time of the SOLVE_DIFFERENCE_CONSTRAINT_and_MARK_CYCLE $O(V^2.E + V) = O(V^2.E)$. □

Proof of Theorem 7.4. The routine RELAX is called V.E times, and RELAX runs in O(1). Hence the running time of SOLVE_DIFFERENCE_CONSTRAINT_without_CYCLE_CHECK is O(V.E). □

Proof of Theorem 7.5.

The cycles in *auth_cycles* can be removed in C_a time where C_a is the number of negative cycles in *auth_cycles*.

The sorting of the edges with respect to the priorities can be done in O(E.log(E)) time. Finding and removing the duplicate negative cycles in the list requires $O(C_s.\log(C_s).V)$ time: there are $C_s.\log(C_s)$ comparisons each requiring O(V) time for checking the identicality.

The last while loop removes the cycles, and it will take at most C_s iterations where C_s is the number of negative cycles in *sys_cycles*. Each iteration of the loop is $O(E.V.C_s)$: there are E edges, and counting the number of negative cycles for an edge requires $V.C_s$ comparisons. Hence, the overall running time of the last while loop is $O(E.V.{C_s}^2)$.

Therefore, REMOVE_CYCLES works in $O(E.V.{C_s}^2 + C_s.\log(C_s).V + E.\log(E) + C_a)$. □

Proof of Theorem 7.6. If there are negative cycles, then the running time of SOLVE_TEMPORAL_CONSTRAINT_and_MARK_CYCLE is the some of the running times in theorems 7.3,7.4, and 7.5 which is equal to $O(V^2.E + E.V.{C_s}^2 + C_s.\log(C_s).V + E.\log(E) + C_a)$. □

Proof of Theorem 7.7. The running time of this algorithm is equivalent to the running time of the SOLVE_DIFFERENCE_CONSTRAINT_and_MARK_CYCLE algorithm except that there is one less constraint in the graph.

Proof of Theorem 7.8. This algorithm consists of three parts. First part involves the computation of the shortest path tree using the constraints that are known to be negative cycle free. Hence for this part we use the SOLVE_DIFFERENCE_CONSTRAINT_without_CYCLE_CHECK algorithm which runs in $O(V^2.(E\text{-}C_m\text{-}1))$. In the second part of the algorithm, the edges that are omited from the first part are sorted in descending order of priority. the running time of the sort routine is $O(C_s.\log(C_s))$. In the last part of the algorithm, we try to insert the marked constraints into the graph one by one. Note that, since the constraints that are being inserted are already marked, if they result in conflict, they can be omited without any further investigation. Since checking of the conflicts can be done in O(1) time, and since each edge in the graph can be relaxed only once for each new constraint, the running time of this part of the algorithm is $O(C_m.E)$. Hence, the overall running time of the DELETE_CONSTRAINT$_2$ algorithm is $O(V^2.(E\text{-}C_m\text{-}1) + C_m.\log(C_m) + C_m.E)$.