CHAPTER 3

DYNAMIC CONSISTENCY CHECKING

3.1 EXISTING WORK

3.1.1 Consistency Checking

A graph theory based approach was proposed by Ma and Shin (2004) for analyzing the consistency of multimedia constraints. This approach uses a minimum spanning tree for checking the consistency of temporal and spatial constraints independently. The authors have proposed the three operators illustrated in Figure 3.1, to capture the temporal relations between the media objects. In each of the operators $A$ and $B$ represent media objects

\[\text{Sequential : Seq}(A,B,d)\]

while $d_1$, $d_2$ and $d$ capture the difference between the beginning and ending time of the media objects as shown in the Figure 3.1. These operators can be used to represent the thirteen traditional temporal relations between any two media objects $A$ and $B$. For instance, consider the temporal relation $B$ starts
15 secs after A starts and ends 5 secs before A ends. This can be modeled as a constraint using the Cov operator as Cov(A,B,15,5). The relation B starts 15 secs after A starts and ends 5 secs after A ends can be modeled using the Ovl operator as Ovl(A,B,15,5). The constraint Seq(A,B,0) could be used to model the relation B starts immediately after A ends. These operators were used to model the synchronization specifications as a set of constraints. The constraints were then used to generate the temporal consistency graph. The vertices of this graph represent media objects and the edges represent the constraints. The connectivity of this graph was used to check for completeness and a minimum spanning tree was constructed from this graph. The non-tree edges, that were dropped during the construction of the minimum spanning tree as they represented inconsistency, were then checked for possible inclusion using a suitable relaxation policy. The approach, presented Ma and Shin (2004), is illustrated here. Consider the following temporal constraints on the set of objects \{A, B, C, D, E\}.

\[
T_1 : Ovl(A, B, 6, 10) \\
T_2 : Cov(B, C, 10, 0) \\
T_3 : Ovl(A, C, 0, 10) \\
T_4 : Ovl(A, B, 3, 5) \\
T_5 : Seq(D, E, 5) \\
T_6 : Seq(A, F, 10)
\]

The priority values assigned for the constraints are: 1 for \(T_1 \) and \(T_6\), 2 for \(T_2\) and \(T_3\) and 3 for \(T_4\) and \(T_5\). The algorithm consists of five phases: pre-processing, completeness checking, minimum spanning tree construction, removing inconsistencies using a relaxation policy and the temporal layout generation. Each of their functions are briefly described below.

- **Pre-processing:** In the first phase, the algorithm checks each of the
constraints to ensure that there is at most one constraint between any pair of objects. In the illustration, $T_1$ and $T_4$ involve the same objects, but since $T_1$ has a higher priority $T_4$ is dropped.

- **Completeness checking:** In the next phase, the constraints remaining after the pre-processing phase are considered and modeled as a graph. Then the graph generated is examined for connectedness. If the graph is not connected, additional constraints are automatically generated in order to make it connected. By default, the system always resolves incompleteness by adding a constraint of the form $Seq(A_1, A_2, 0)$. Here, $A_1$ and $A_2$ are chosen so that they belong to different components of the disconnected graph. In the example, the graph generated is not connected as shown in Figure 3.2(a) and hence a constraint $T_7: Seq(A, D, 0)$ having priority 1 is added by the system (Figure 3.2(b)).

![Figure 3.2: Temporal consistency checking](image)

- **Minimum spanning tree construction:** The third phase takes the generated connected graph as input, and produces a minimum spanning tree
(an acyclic graph connecting all vertices such that the total weight of all included edges is the minimum possible) using Kruskal's algorithm (Kruskal 1956). The minimum spanning tree generated in the illustration is shown in the Figure 3.2(c). In order to generate an acyclic graph, some of the constraints would have to be dropped. These are maintained in a separate list to be used for comparison in phase four. The constraint dropped in the third phase of the example is $T_3$.

- **Removing inconsistencies using a relaxation policy:** The fourth phase takes as input the minimum spanning tree and the list of dropped constraints. Each of these constraints is in turn considered for possible inclusion into the spanning tree, so that the entire process would result in the best set of consistent constraints. The inclusion of any of the dropped constraints would result in dropping of another constraint from the minimum spanning tree. This decision is governed by the relaxation policy. In order to implement the relaxation policy, the implied constraint between the two vertices under consideration is first determined. For instance, the implied constraint between vertices $A$ and $C$ in the above example is $T_3' : \text{Ovl}(A, C, 16, 10)$ obtained by adding the value of the arguments in $T_1$ and $T_2$ (addition was done in this case as the direction of the edges along the path was the same). Hence, according to the relaxation policy employed by Ma and Shin (2004), the original constraint $T_3$ is modified as $T_3'' : \text{Ovl}(A, C, 8, 5)$, i.e. by taking the average of the values of the arguments in $T_3$ and the implied constraint $T_3'$. The constraint $T_3''$ is now included in the minimum spanning tree and $T_2$ is dropped instead. This is one of the cases that may arise. The final spanning tree obtained is shown in Figure 3.2(d).

- **Temporal layout generation:** The last phase of the algorithm is carried out as follows: A breadth-first search (BFS) is performed on the
minimum spanning tree to find a unique path from designated source to each vertex. Using the BFS and the constraints in the path, the begin and end time of each media object (vertex) is determined. Each of these are scheduled as events and they are sorted in the temporal order. The final temporal layout can be easily generated from this sorted list of events. The temporal presentation layout of the example is given in Figure 3.3. Here, the length of the first object $A$ is specified to be 30 secs, $D$ is 10 secs, $E$ is 15 secs and $F$ is 5 secs. Whenever the $Seq (A,B,d)$ operator is used as a constraint, the length of the media object $B$ also needs to be specified. This is one of the limitations of this operator.

![Graph](image)

**Figure 3.3: Presentation layout**

The temporal layout generated for a given specification gives rise to a number of presentation intervals. Within each of these intervals the spatial consistency may be checked independently. In order to model these spatial constraints, four operators have been proposed by Ma and Shin (2004) to represent the six topological relations. The method for checking spatial consistency is similar to the approach used for temporal consistency checking with the same five phases being repeated.
3.1.2 Motivation

Dynamic consistency checking is one of the features that will enable a truly interactive authoring of multimedia presentations. An algorithm for dynamic consistency checking of temporal and spatial relations has been presented in this chapter. This work is a modification of the algorithm for consistency checking presented by Ma and Shin (2004). A reduction of the computation time by a constant factor, without changing the asymptotic upper bound unlike the algorithm by Ma and Shin (2004), is achieved by simultaneously introducing a dynamic approach towards consistency checking in the proposed algorithm. Here dynamism means that the consistency checking will be performed while each constraint is being specified during the authoring of the presentation, unlike in the approach presented by Ma and Shin (2004) where consistency checking commences after the entire set of constraints have been specified. The proposed approach would help in resolving inconsistencies when the presentation is being or has been created. New operators have also been proposed here for modeling temporal and the spatial relations, and these operators have helped to overcome the limitations of the existing operators used by Ma and Shin (2004). Mechanisms for creation and interaction with multimedia presentations have been studied extensively. While Huang and Wang (1998) deal with interaction in terms of navigation of the presentation, the proposed algorithm would enable interaction with the presentation, as discussed in the reference model presented by (Rogge et al 2004). Moreover, the composition of the spatial and temporal relations discussed by Vazirgiannis et al (1998) has been modeled using a spatio-temporal operator and this helps in the effective generation of the presentation layout. The proposed approach would allow the user to change the spatial and temporal layout i.e it permits interaction with the presentation while authoring, as well as during playout and also effectively resolves spatio-temporal consistency issues.
Two approaches for consistency checking has been presented in this Chapter. In the first approach the temporal and spatial constraints are dealt with separately. After the temporal layout are generated, the spatial constraints between the objects appearing in each presentation interval are examined for consistency. Although this approach allows dynamism in dealing with temporal constraints, the spatial constraints could be dealt with only after all the temporal constraints have been processed. Hence dynamism could not be extended to the spatial constraints. For instance, the temporal layout in Figure 3.3 has five presentation intervals (a presentation interval represents an interval of time during which the participating media objects remain the same). The algorithm accepts spatial constraints for each of the five intervals and generates spanning trees for each of them. Thus, for the above illustration, the algorithm would require the generation of seven (one for temporal constraints and six for spatial constraints) spanning trees and their respective layouts to generate a consistent presentation schedule.

For presentations with rapid changes in media objects, the first approach would not be efficient. Moreover, spatial specifications between pairs of media objects appearing in more than one interval need to be repeated in each interval. Although this method could be extended to support interactions with the presentation during playout, each time a temporal constraint is modified by the user, all the spanning trees for the spatial constraints may have to be regenerated. As a result, this approach would be inefficient in terms of the response time during interaction.

In order to overcome these limitations, a second approach that uses an integrated operator for representing both the temporal and spatial relations has been proposed. As a result, the entire process would require just one spanning tree regardless of the number of presentation intervals generated.
This leads to the development of a consistency checking mechanism which is highly responsive to interactions with the presentation during playout, and also achieves a considerable reduction in time by a constant factor in comparison with the approach presented by Ma and Shin (2004) and also with the first approach proposed in this chapter.

3.2 DYNAMIC CONSISTENCY CHECKING - I

3.2.1 Specification of Constraints

In the method discussed by Ma and Shin (2004), consistency checking could be performed only after the entire set of constraints were available. This was improved upon here and made dynamic by examining each constraint immediately on input. For the implementation of this method, the following functionally complete temporal operator was proposed:

\[
TEMPORAL(A, B, d_1, d_2, priority)
\]

(3.1)

where \( d_1 = b_B - b_A \) and \( d_2 = e_B - e_A \). Here, \( b_k \) and \( e_k \) for \( k \in \{A, B\} \) are respectively the beginning and ending times of the media objects \( A \) and \( B \) respectively, and \( priority \) is the priority value assigned to the constraint. This operator was used to specify the temporal relations as a set of constraints.

The consistency checking algorithm presented by Ma and Shin (2004) was made more efficient by eliminating some of the redundant steps. The new algorithm was designed to examine each constraint immediately on input for possible inconsistency with those already in the tree. This process helps to construct the spanning tree dynamically.

Unlike in the approach presented by Ma and Shin (2004), the proposed approach eliminates pre-processing totally and combines the rest
of the phases and performs them simultaneously, thereby reducing the total running time of the algorithm. As a result, consistency checking could commence immediately on input of the first constraint, unlike in the approach presented by Ma and Shin (2004), where the entire set of constraints is required to start the procedure. In case a cycle is identified while checking for possible inclusion in a spanning tree, it is recognised as an inconsistency. Then one of the constraints is dropped, using an appropriate relaxation policy. Finally, when all the constraints are given as input, the resulting graph is examined for completeness and the user is prompted to specify an appropriate constraint to resolve any possible incompleteness. Consider a multimedia presentation involving 5 multimedia objects A, B, C, D and E and 5 temporal constraints as follows:

\[
\begin{align*}
T_1 & : TEMPORAL(A, B, 5, 10, 1) \\
T_2 & : TEMPORAL(B, C, 10, 0, 1) \\
T_3 & : TEMPORAL(A, C, 0, 10, 2) \\
T_4 & : TEMPORAL(A, B, 3, 5, 3) \\
T_5 & : TEMPORAL(D, E, 5, 10, 3)
\end{align*}
\]

For this presentation, it is assumed that the object A starts at time 0 and its length is 30 seconds. The spanning tree created is shown in Figure 3.4(a) and Figure 3.4(b) and the temporal layout in Figure 3.4(c). The algorithm works as follows: The constraints \(T_1\) and \(T_2\) do not give rise to any inconsistency and are included in the spanning tree. Since \(T_3\) is inconsistent with the above constraints, it is dropped since it has a lower priority than \(T_1\) and \(T_2\). Again \(T_4\) is dropped since it is inconsistent with \(T_1\). Next \(T_5\) is included. Since all the constraints specified have been checked, the algorithm now checks for completeness and prompts the user to specify a new constraint \(T_6\) to generate
a complete set of constraints. Suppose the user specifies the new constraint as

\[ T_6 : TEMPORAL(A, D, 10, 10, 1) \]

the algorithm finds that the tree is complete and hence it generates the temporal layout. A similar algorithm is used to check the spatial constraints between the media objects present in each interval for consistency.

### 3.2.2 Dynamic Consistency Checking Algorithm

The proposed dynamic consistency checking approach has been formally presented in Algorithm 3.1. In Algorithm 3.1, \( V \) represents the set of nodes of the generated spanning tree \( T \), while \( E \) represents the set of its edges. The following subroutines are used in the algorithm:

- **MAKESET**\( (i) \): creates a new set with representative \( i \). The only member of this new set is the element \( i \). (A representative of a set is any one element chosen from the set to uniquely identify the set. The element
could be chosen using an arbitrary rule. For instance, here the first element in the lexicographic ordering of the elements of the set is considered to be the representative).

- **FINDSET(i)**: is a function which returns a pointer to the representative of the set containing $i$.
- **UNION(i,j)**: merges the two sets corresponding to the elements $i$ and $j$ and assigns the representative of one of the two sets as the representative of the new set.
- **LIST(S)**: enumerates the elements of set $S$.
- **GETPATH(S,i,j)**: uses the depth first search algorithm to generate a search tree and traverses the generated tree to determine a path that starts at $j$ and ends at $i$.
- **RELAX(path)**: identifies the edge with the largest priority value in the path (determined by GETPATH) and implements the relaxation policy.
- **GETLENGTH(T,i)**: uses a modification of the breadth first search algorithm to traverse the spanning tree starting at $i$ and obtains the attributes of each of the media object.

In the following, the three core steps of the algorithm are explained. This algorithm could be easily extended to handle other authoring features like addition and deletion, and also to update the constraints and presentation objects.

- **Consistency checking**: Consistency Checking is done in steps 1 to 20. The spanning tree is created as follows: A collection of disjoint sets of vertices is initialized by invoking the subroutine **MAKESET** each time a new vertex appears. In order to check whether there is a path in $T$ between two vertices $i$ and $j$, the subroutine **FINDSET** is used on each of them. If the the sets containing vertices $i$ and $j$ have the same representative, it indicates that $i$ and $j$ belong to the same set and also that there exists a
Algorithm 3.1: Temporal-ConCheck

Input: constraints
Output: T, I
1 while user generates constraints do
2 \quad \text{TEMPORAL}(i, j, d_1, d_2, \text{priority}) \leftarrow \text{constraint};
3 \quad \text{if } (i \notin V) \text{ then}
4 \quad \quad V \leftarrow V \cup \{i\};
5 \quad \quad \text{MAKESET}(i);
6 \quad \text{end}
7 \quad \text{if } (j \notin V) \text{ then}
8 \quad \quad V \leftarrow V \cup \{j\};
9 \quad \quad \text{MAKESET}(j);
10 \quad \text{end}
11 \quad \text{if } (\text{FINDSET}(i) \neq \text{FINDSET}(j)) \text{ then}
12 \quad \quad E \leftarrow E \cup \{e(i,j)\};
13 \quad \quad \text{UNION}(i,j);
14 \quad \text{end}
15 \quad \text{else}
16 \quad \quad S \leftarrow \text{FINDSET}(j);
17 \quad \quad \text{path} \leftarrow \text{GETPATH}(S,i,j);
18 \quad \quad \text{RELAX}(\text{path});
19 \quad \text{end}
20 \text{end}
21 R \leftarrow V;
22 i \leftarrow \text{first object};
23 \text{while } R \neq \text{NULL do}
24 \quad S \leftarrow \text{FINDSET}(i);
25 \quad R \leftarrow R - \text{LIST}(S)
26 \quad \text{Choose } j \in R
27 \quad \text{input constraint } e(i,j)
28 \quad E \leftarrow E \cup \{e(i,j)\}
29 \quad i \leftarrow j
30 \text{end}
31 \text{Input length of the first object } i ;
32 T \leftarrow (V,E);
33 \text{GETLENGTH}(T,i);
34 \text{Sort all events in temporal order;}
35 \text{Determine the set of presentation intervals } I ;

path between them. Hence, in case at some point of time while processing an edge \( e(i, j) \), it is found that \( i \) and \( j \) are not connected by any path (i.e. they do not have the same representatives), then the edge can be safely
added to get a new graph, which is still acyclic. However, once this is
done, the sets containing these two vertices should be merged so that
any future invocation of FINDSET will return the same representative
for both sets. This is done by performing UNION(i,j). On the other
hand, if i and j are already connected by a path, FINDSET(i) in co-
ordination with GETPATH helps to identify the path which forms a
cycle along with the newly input edge. In order to ensure consistency
and maintain the acyclic nature of T, one of the edges on this path is
dropped. The choice of which edge is to be dropped is made by using
RELAX(path) which implements the relaxation policy discussed in the
next section. The DFS subroutine also keeps track of the edge with the
lowest priority encountered.

- **Completeness checking:** Completeness checking of constraints is done in
steps 21 to 30 where FINDSET and LIST are used to obtain all the
vertices in the set containing the first media object. If this generates
the whole of V, the set of constraints is complete, indicating that the
media objects that belong to the presentation are directly or indirectly
related to the first media object. Otherwise, the algorithm prompts the
user to supply an appropriate constraint to make the set of constraints
complete.

- **Temporal layout generation:** The temporal layout generation is handled
in steps 31 to 35. The user is prompted to give the length of the first
media object. The GETLENGTH function, beginning at this vertex, is
used to compute the start time, end time and duration of each media
object represented by the nodes. The start and end of media objects are
considered to be events that are then sorted. A temporal layout of the
entire presentation is formulated from this sorted list of events.
Relaxation Policy:

The algorithm Temporal-ConCheck is designed so as to resolve the inconsistencies based on the relaxation policy described. This policy could change from application to application, and could also be used to provide Quality-of-Presentation guarantees in presentations that permit transformable presentation spaces. As mentioned earlier, whenever a cycle is detected (step 16) on input of an edge \( e(i, j) \), the \( GETPATH \) function is employed (step 17), to find a path connecting the vertices \( i \) and \( j \). Now, the relaxation policy comes into play to resolve the inconsistency that is implied by the existence of a cycle. Here, the lowest priority in the path is compared with the priority of the incoming edge, and the corresponding edge is replaced only if the new edge has a higher priority. In all other cases, the original edge is retained in the tree.

Algorithm 3.2: \( GETPATH(S, i, j) \)

<table>
<thead>
<tr>
<th>Input: ( S, i, j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: ( path )</td>
</tr>
<tr>
<td>1 if ( j = i ) then</td>
</tr>
<tr>
<td>2 \quad insert ( i ) onto the front of a linked list - ( path );</td>
</tr>
<tr>
<td>3 end</td>
</tr>
<tr>
<td>4 else</td>
</tr>
<tr>
<td>5 \quad if ( \pi[j] = \text{nil} ) then</td>
</tr>
<tr>
<td>6 \quad \quad Print ”no path exist”;</td>
</tr>
<tr>
<td>7 \quad end</td>
</tr>
<tr>
<td>8 \quad else</td>
</tr>
<tr>
<td>9 \quad \quad \quad GETPATH( (S, i, \pi[j]) );</td>
</tr>
<tr>
<td>10 \quad \quad insert ( j ) onto the front of a linked list - ( path );</td>
</tr>
<tr>
<td>11 \quad end</td>
</tr>
<tr>
<td>12 end</td>
</tr>
<tr>
<td>13 return(path)</td>
</tr>
</tbody>
</table>

The Algorithm 3.2 describes the \( GETPATH(S, i, j) \) function. This function returns the shortest path from \( i \) to \( j \) assuming the a DFS has already been run to compute the shortest path tree. The \( GETPATH \) function is a
modified version of the PRINT-PATH() procedure presented by Cormen et al (1990). The DFS procedure builds a depth first tree as it searches the graph and records the predecessor of each node \( v \) in the field represented by \( \pi[v] \). The \textit{GETPATH} is called recursively each time using \( \pi[v] \) (step 9). The linked list that maintains the path is finally returned in the last step of the algorithm.

\begin{algorithm}
\textbf{Algorithm 3.3: GETLENGTH}
\begin{algorithmic}
\State \textbf{Input:} \( T,i \)
\State \textbf{Output:} beginning time, ending time and length of each
\hspace{1cm} media object
\For {each vertex \( u \in V[T] - \{i\} \)}
\State \text{colour}[u] \leftarrow \text{white};
\State \( \pi[u] \leftarrow \text{nil} \);
\EndFor
\State colour[\( i \)] \leftarrow \text{gray}
\State \( \pi[i] \leftarrow \text{nil} \);
\State \( Q \leftarrow \text{NULL} \);
\State \text{ENQUEUE}(Q,i);
\While {\( Q \neq \text{NULL} \)}
\State \text{u} \leftarrow \text{DEQUEUE}(Q);
\For {each \( \text{v} \in \text{Adj}[u] \)}
\If {\text{colour}[\text{v}] = \text{white}}
\State \text{colour}[\text{v}] \leftarrow \text{gray};
\State \( \pi[\text{v}] \leftarrow \text{u} \)
\State \text{ENQUEUE}(Q,\text{v})
\EndIf
\EndFor
\State \text{colour}[\text{u}] \leftarrow \text{black};
\State \( b_v = d_1 + b_{\pi[\text{v}]} \);
\State \( e_v = d_2 + e_{\pi[\text{v}]} \);
\State \( l_v = e_v - b_v \);
\EndWhile
\end{algorithmic}
\end{algorithm}

The \textit{GETLENGTH} function described in Algorithm 3.3 is a modification of the BFS procedure (Cormen et al 1990). At each node in addition to the predecessor field \( \pi[v] \) the colour attribute \( \text{colour}[v] \) is also used to keep track of the progress during the BFS traversal. BFS uses three colours \textit{white},
gray or black. All nodes are initialized to white. If edge \((u,v) \in E\) and vertex \(u\) is black, then vertex \(v\) is either gray or black i.e all vertices adjacent to black vertices are considered to be nodes that have been discovered. Gray vertices may have some adjacent white vertices. The procedure \textit{GETLENGTH} obtains the begin time \(b_v\), end time \(e_v\) and the length \(l_v\) for each media object as the nodes are discovered and assigned the colour black. The queue \((Q)\) is used to keep track of the visited nodes and the \(\text{Adj}[v]\) field contains the list of nodes that are adjacent to \(v\).

\textbf{Illustration:}

The multimedia scenario first presents a logo consisting of a \textit{Logo-Animation}(A) and \textit{Logo-Music}(B) for a duration of 20 seconds. Then an \textit{Audio}(C) is played with a \textit{Video}(D) for a duration of 20 seconds. A \textit{Text}(E) will be displayed while playing the \textit{Video}(D). The presentation ends with 10 seconds of an \textit{Exit-Animation}(F) accompanied by an \textit{Exit-Music}(G). This scenario can be specified using the following constraints:

\[
\begin{align*}
T_1 & : \text{TEMPORAL}(A,B,0,0,1) \\
T_2 & : \text{TEMPORAL}(D,E,4,0,2) \\
T_3 & : \text{TEMPORAL}(C,D,0,0,1) \\
T_4 & : \text{TEMPORAL}(C,F,20,10,2) \\
T_5 & : \text{TEMPORAL}(F,G,0,0,1) \\
T_6 & : \text{TEMPORAL}(C,E,0,-4,2) \\
T_7 & : \text{TEMPORAL}(D,G,15,15,3)
\end{align*}
\]

The consistency checking used by the algorithm is illustrated in Figure 3.5 and in Table 3.1. Figure 3.5 illustrates how the spanning tree is
dynamically generated on input of each constraint, while Table 3.1 helps to understand the role of the three functions MAKESET(i), FINDSET(i) and UNION(i,j). Figure 3.5 shows that the input of the constraints from T1 up to T5 did not give rise to any inconsistency, though an incompleteness condition exits (i.e. the existence of a spanning forest). The existence of two independent sets can be observed from Table 3.1. The input of the next constraint T6 leads to an inconsistency and the algorithm drops T6. The input of the last constraint T7 also leads to an inconsistency and is also dropped eventually. The algorithm now checks for the completeness and prompts the user to specify a constraint T8 to relate objects A and D. Assume that the user specifies the following constraint:

\[ T_8 : TEMPORAL(A, D, 10, 20, 1) \]

The inclusion of this constraint would generate a set of constraints that is complete and consistent. The temporal layout is then generated from a sorted list of temporal events. In this example, it would consist of four intervals [0-10], [10-14], [14-30] and [30-40] within which the spatial objects and their relations do not change.

**Spatial Relations:**

In order to model the spatial constraints, the approach presented by Ma and Shin (2004) describes each of the spatial objects in terms of its Minimum Bounding Box (MBB) expressed as \((x, y, w, h, z)\), where \(x\) and \(y\) specify the center, \(w\) and \(h\) specify the width and height of the box, and \(z\) the depth information. The four operators, \(\text{Contain(Cnt)}\), \(\text{Cover(Cvr)}\), \(\text{Intersect(Ist)}\) and \(\text{Disjoint(Djt)}\), proposed by Ma and Shin (2004) to represent all possible topological relations, are given in Figure 3.6. In this figure, \(d_x\) and \(d_y\) respectively represent the distance between the \(x\) co-ordinates and the \(y\)
Figure 3.5: Illustration of temporal constraints

Table 3.1: Illustration of Algorithm Temporal-ConCheck

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Employed Relaxation</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>${A, B}$</td>
<td>no</td>
<td>-</td>
</tr>
<tr>
<td>$T_2$</td>
<td>${A, B} {D, E}$</td>
<td>no</td>
<td>-</td>
</tr>
<tr>
<td>$T_3$</td>
<td>${A, B} {D, E, C}$</td>
<td>no</td>
<td>-</td>
</tr>
<tr>
<td>$T_4$</td>
<td>${A, B} {D, E, C, F}$</td>
<td>no</td>
<td>-</td>
</tr>
<tr>
<td>$T_5$</td>
<td>${A, B} {D, E, C, F, G}$</td>
<td>no</td>
<td>-</td>
</tr>
<tr>
<td>$T_6$</td>
<td>${A, B} {D, E, C, F, G}$</td>
<td>yes</td>
<td>$T_6$ dropped</td>
</tr>
<tr>
<td>$T_7$</td>
<td>${A, B} {D, E, C, F, G}$</td>
<td>yes</td>
<td>$T_7$ dropped</td>
</tr>
<tr>
<td>$T_8$</td>
<td>${A, B, D, E, C, F, G}$</td>
<td>no</td>
<td>complete set</td>
</tr>
</tbody>
</table>

co-ordinates of the centers of the MBBs. Since the different types of spatial constraints are not differentiated, a single spatial operator is used, in the thesis, to capture the relationships specified in the constraints and it requires the
Figure 3.6: Existing spatial operators

Spatial locations of the first object only. The proposed operator is defined as:

\[
SPATIAL(\text{A, B, } d_{x_1}, d_{x_2}, d_{x_3}, d_{y_2}, d_{y_3}, \text{priority})
\]

where \(d_{x_1} = x_{l_B} - x_{l_A}, d_{x_2} = x_{r_B} - x_{r_A}, d_{y_1} = y_{b_B} - y_{b_A}, d_{y_2} = y_{o_B} - y_{o_A}\), and 
\(d_z = z_B - z_A\). Here, \((x_{l_A}, y_{b_A})\) and \((x_{r_A}, y_{u_A})\) are the left bottom (lower left) corner and top right (upper right) corner respectively of the MBB of the object \(A\). The \(d_z\) value in the operator is used to determine whether \(A\) lies in front of \(B\) or vice versa. For instance, if the lower-left corner and the upper-right corner of the MBB of spatial object \(A\) are \((0,0)\) and \((512,512)\) respectively and that of \(B\) are \((56,56)\) and \((456,456)\), then the above operator could be used to represent the spatial relation between \(A\) and \(B\) as \(SPATIAL(A, B, 56, -56, 56 - 56, 0, 1)\).

The spatial constraints appearing within each of the temporal intervals need to be specified or considered independently.

Spatial consistency checking within each temporal interval is done in the same way as the algorithm \textit{Temporal-ConCheck} handles temporal consistency checking. Then the spatial layout is generated for each interval indepen-
dently as done in the approach presented by Ma and Shin (2004). To illustrate this, the example presented by Ma and Shin (2004) is considered again for the spatial consistency checking. Here, three objects are appended to the scenario: a background (A), a caption_1 (B) and a caption_2 (C). Objects A and B will be present throughout the entire presentation and object C will be present only during the third presentation interval. Assume the screen resolution to be \( 512 \times 512 \) with the origin at the lower left corner and the background occupying the entire screen having 0 as its z value. The location and size of the other objects can be obtained from the constraints.

Let \( \text{Background} (A), \text{Caption}_1 (B), \text{Caption}_2 (C), \text{Video} (D), \text{Logo-Animation} (E), \text{Exit-Animation} (F) \) and \( \text{Text} (G) \) be the spatial objects. For each presentation interval generated by the algorithm \( \text{Temporal-ConCheck} \), the user inputs constraints and the consistency is checked dynamically as in the \( \text{Temporal-ConCheck} \). Assume that the user gave the inputs for the four temporal intervals as shown in Table 3.2. The consistency and completeness checking for the third interval [14-30] is illustrated in Figure 3.7. Here, the inconsistency that arises is resolved by dropping the constraint between objects A and G, and the user is prompted to input a constraint to resolve the incompleteness of the spatial constraints in the interval [14-30]. Assume that the user inputs the constraint \( \text{SPATIAL}(B,G,50,-50,440,440,10,1) \). This would lead to a complete set of constraints and the spatial layout being generated. The spatial output that would be generated for each temporal interval is indicated in Table 3.3.

### 3.2.3 Comparison and Analysis

- In the approach presented by Ma and Shin (2004), three temporal operators are required to represent all possible temporal relations, whereas in the proposed approach, one operator is sufficient. Further, the presence
Table 3.2: Illustration of spatial constraints

<table>
<thead>
<tr>
<th>Interval</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0-10]</td>
<td>SPATIAL(A,E,56,-56,56,-56,20,1)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPATIAL(E,B,50,-50,-40,-400,-10,1)</td>
</tr>
<tr>
<td>[10-14]</td>
<td>SPATIAL(A,D,0,0,0,0,20,1)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPATIAL(D,B,106,-106, 16,-456,-10,1)</td>
</tr>
<tr>
<td>[14-30]</td>
<td>SPATIAL(A,D,0,0,0,0,20,1)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPATIAL(D,G,156,-156,456,-16,0,1)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPATIAL(A,G,156,-156,436,-36,20,2)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPATIAL(B,C,40,-50,40,40,0,1)</td>
</tr>
<tr>
<td>[30-40]</td>
<td>SPATIAL(A,F,56,-56,56,-56,20,1)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPATIAL(F,B,50,-50,-40,-400,-10,3)</td>
</tr>
</tbody>
</table>

(a) Inconsistent  (b) Incomplete

(c) Consistent and complete

Figure 3.7: Illustration of spatial constraints for interval [14-30]

of three operators makes the algorithm and the relaxation policy more complicated to handle. Similarly, one spatial operator has been designed instead of the four used in the approach presented by Ma and Shin (2004).
- During layout generation, the length of the objects are required for scheduling the begin and end events in the temporal order. In the proposed algorithm, this information can be retrieved from the length of
the first object and the final set of consistent temporal relations. However
in the approach presented by Ma and Shin (2004), layout generation
cannot be successfully completed whenever the operator $Seq$ is encoun-
tered on the path. This situation arises in the approach presented by Ma
and Shin (2004), since the length information of object $A$ and the value
$d$ in the operator $Seq(A,B,d)$ gives us only the begin time of object $B$
and not the end. Hence, the end event for $B$ can be scheduled only if its
length is known. This has not been mentioned in the approach presented
by Ma and Shin (2004).
- The approach presented by Ma and Shin (2004) arbitrarily assigns a \textit{Seq} constraint for temporal and a \textit{Djt} to resolve spatial incompleteness. This may satisfy the condition of completeness of the set of constraints, but may not satisfy the presentation requirements as envisaged by the author. Hence, in the proposed algorithm user intervention is insisted upon to resolve this problem.

- As mentioned earlier, the algorithm presented by Ma and Shin (2004) consists of five phases - pre-processing, completeness checking, spanning tree generation, removal of inconsistencies and layout generation. Since redundant constraints may be treated as inconsistent constraints, phase 1 (pre-processing) does not serve any useful purpose. Hence, in the proposed approach, phase 1 is dropped. This leads to the reduction in the total running time required for the algorithm.

- During phase 4 (removal of inconsistencies) in the approach presented by Ma and Shin (2004), each of the constraints dropped in phase 3 is re-examined for possible inclusion in the minimum spanning tree. If the decision is to include this new constraint, some other constraint of the same or lower priority has to be dropped. This means that the final set of constraints generated need not necessarily be optimal in terms of priority. Moreover, the relaxation policy employed in the approach presented by Ma and Shin (2004) bases its decision on the priority values. Consider a constraint involving two media objects \( A \) and \( B \). Here, if the lowest priority in the path from \( A \) to \( B \) already available in the spanning tree is 1, the decision is to drop the new constraint unless the priority of the new constraint is also 1. In all other cases, a weighted average of the values of the arguments appearing in the relations (the new relation and the implied relation obtained from the path) is computed and a new constraint with these arguments is included, dropping the edge with
the lowest priority in the path. This may lead to unplanned temporal relationships that could be quite annoying while viewing the presentation in the approached proposed by Ma and Shin (2004). Hence, in the proposed algorithm, one edge is replaced by another only if the incoming edge has a higher priority. This is possible in the proposed approach since the constraints based on priorities are not sorted in phase 3. In addition, a relaxation policy is implemented while creating the minimum spanning tree and hence does not require phase 4 to be executed separately. This leads to a further reduction in the running time.

- An attempt to simplify the algorithm presented by Ma and Shin (2004) led to the development of a dynamic consistency checking mechanism which permits the system to respond to inconsistencies while the presentation is being created. The approach is referred to as dynamic because each constraint is being processed for possible inclusion in the minimum spanning tree immediately on input. On the other hand, in the algorithm presented by Ma and Shin (2004) the preprocessing and completeness checking phases followed by the consistency checking process could be performed only after the entire set of constraints is available. If completeness checking is done prior to consistency checking, it will not allow the system to be dynamic. Hence completeness checking is performed only after all the constraints have been input and checked for inconsistency.

- Although the approach presented by Ma and Shin (2004) supports interactive authoring, its scope is limited to a mere static editing feature. It is evident from the approach proposed in this thesis that the same features (adding a constraint/object, removing a constraint/object, and modifying a constraint) could be performed along with the dynamic creation of the presentation.

- In the approach presented by Ma and Shin (2004), the value of $x$, $y$ and $z$
had to be provided for the first object and the \( w \) and \( h \) values of the MBB had to be specified for each of the spatial objects. The corresponding \( x \), \( y \) and \( z \) values were assigned during the generation of the spatial layout after the consistency and completeness were checked. In the approach proposed in this thesis, the location and dimensions have to be specified only for the first object. The spatial location of all other objects could be calculated during layout generation using the arguments specified in the constraints.

### 3.2.4 Proof of Correctness

Assume on the contrary, that the graph \( T \) generated is not consistent. Let \( e(i, j) \) be the first constraint which gives rise to an inconsistency on inclusion in \( T \) i.e. until \( e(i, j) \) is added, \( T \) is consistent. Since \( e(i, j) \) gives rise to an inconsistency, \( T \) must have had a path from vertex \( i \) to vertex \( j \) before adding \( e(i, j) \). In this case, the condition checked in step 11 of the algorithm \( Temporal-ConCheck \) is not satisfied. Obviously, \( e(i, j) \) has a higher priority than one of the edges on the path involving the objects \( i \) and \( j \). In such a case, this already existing edge is dropped and \( e(i, j) \) is added ensuring that the graph \( T \) remains acyclic. Hence, the graph generated upto this point is consistent. This contradicts the assumption that the addition of the new edge \( e(i, j) \) creates an inconsistency. Suppose on the other hand, \( T \) is not complete. In this case steps 21 to 29 detects this and prompts the user to specify the necessary constraint, say \( e(i, j) \). This does not give rise to an inconsistency since the system prompts the user for a constraint only when \( FINDSET(i) \) is not the same as \( FINDSET(j) \). Thus the algorithm \( Temporal-ConCheck \) generates a consistent and complete temporal layout.
3.2.5 Complexity Analysis

Assume that the presentation has a total of \( m \) constraints dealing with \( n \) media objects. The algorithm has \( n \) MAKESET operations and that requires \( O(n) \) time. Also, the algorithm is required to perform \( m \) FINDSET and \( n - 1 \) UNION operations, which requires \( O(m \alpha(m, n)) \) time, as \( m \geq n \) where \( \alpha \) is an inverse Ackermann’s function (Tarjan 1975). The algorithm also has \( m - n + 1 \) DFS operations (used by the GETPATH function) requiring time \( O(n(m-n)) \) and one sort operation requiring time \( O(n \log n) \). Hence, the total time complexity of the algorithm is \( O(mn) \) (Cormen et al 1990).

In the approach presented by Ma and Shin (2004) also, the worst case complexity is \( O(mn) \) and not \( O(n \log n) \) as claimed. However, since one out of the five phases (pre-processing) of the algorithm presented by Ma and Shin (2004) is dropped in the proposed approach, and two others (spanning tree generation and removal of inconsistencies) are merged, a considerable reduction in the running time is achieved. Although the process that has been used for generating the spanning tree is very similar to Kruskal’s algorithm, the difference is that the constraints are not sorted based on the priority prior to the construction of the spanning tree. Instead, the simplified relaxation policy ensures that the generated tree is a minimum spanning tree. Hence, there is a further saving in the time required for sorting the constraints. A similar discussion on the time complexity for the spatial consistency checking yields \( O(lmn) \), where \( l \) is the number of intervals in the temporal layout.
3.3 DYNAMIC CONSISTENCY CHECKING - II

3.3.1 Spatio-Temporal Relations

In order to design a truly dynamic approach for consistency checking, spatial and temporal relations between two objects need to be combined and represented the same operator. For this purpose, a new composite spatio-temporal operator $ST$ is introduced, which helps to capture all aspects of the relationship between two media objects in an effective way. It is defined as:

$$ST(A, B, d_{t_1}, d_{t_2}, d_{t_3}, d_{l_4}, d_l, d_r, d_b, d_u, d_z, \text{flag, priority})$$  \hspace{1cm} (3.3)

where $d_{t_1} = b_B - b_A$, $d_{t_2} = t_B - t_A$, $d_{t_3} = b_B - t_A$, $d_{l_4} = t_B - b_A$, $d_l = x_{l_B} - x_{l_A}$, $d_r = x_{r_B} - x_{r_A}$, $d_b = y_{d_B} - y_{d_A}$, $d_u = y_{u_B} - y_{u_A}$, $d_z = z_B - z_A$. The flag in the operator is used to indicate the presence or absence of the spatial relationship between $A$ and $B$, using 1 or 0 respectively. The absence of a spatial relation occurs when either $A$ or $B$ or both of them are audio elements. The priority is assigned by the algorithm and is used by the relaxation policy to resolve the inconsistency.

3.3.2 Integrated Consistency Checking Algorithm

The composite operator defined above enables the algorithm to generate a consistent set of constraints by building a single spanning tree for the entire presentation. Thus temporal and spatial constraints are checked simultaneously. The revised relaxation policy used with Algorithm 3.4 has been presented here. The rest of the functions and the steps are the same as in Algorithm 3.1.
Algorithm 3.4: Integrated_ConCheck Procedure

Input: constraints

Output: T, I

1 while user generates constraints do
2  \[ ST(i, j, d_t, d_z, d_t, d_t, d_t, d_t, d_t, d_t, d_t, d_t, d_t, d_t, d_t, flag, priority) \leftarrow \text{constraint}; \]
3  if \((i \notin V)\) then
4     \[ V \leftarrow V \cup \{i\}; \]
5     \[ \text{MAKESET}(i); \]
6  end
7  if \((j \notin V)\) then
8     \[ V \leftarrow V \cup \{j\}; \]
9     \[ \text{MAKESET}(j); \]
10 end
11 if \((\text{FINDSET}(i) \neq \text{FINDSET}(j))\) then
12     \[ E \leftarrow E \cup \{e(i, j)\}; \]
13     \[ \text{UNION}(i, j); \]
14 end
15 else
16     \[ S \leftarrow \text{FINDSET}(j); \]
17     \[ \text{path}_1 \leftarrow \text{GETPATH}(S, i, j); \]
18     \[ \text{RELAX}(	ext{path}_1); \]
19 end
20 end
21 \[ R \leftarrow V; \]
22 \[ i \leftarrow \text{first object}; \]
23 while \(R \neq \text{NULL}\) do
24     \[ S \leftarrow \text{FINDSET}(i); \]
25     \[ R \leftarrow R - \text{LIST}(S); \]
26     \text{Choose } j \in R;
27     \text{input constraint } e(i, j);
28     \[ E \leftarrow E \cup \{e(i, j)\}; \]
29     \[ i \leftarrow j; \]
30 end
31 Input length of the first object \(i:\)
32 \[ T \leftarrow (V, E); \]
33 \[ \text{GETLENGTH}(T, i); \]
34 Sort all events in temporal order;
35 Determine the set of presentation intervals I;

Relaxation policy:

Chbeir et al (2002), have proposed a meta-model which was used to classify relations. This model helped spatial and temporal relations to have
more expressive power. For instance, the authors have identified the existence of 33 temporal relations between two intervals instead of the 13 that are traditionally used by several applications. On the basis of the proposal provided by Chbeir et al (2002), a relaxation policy was designed to resolve inconsistencies. In order to classify the temporal relations, as in Table 3.4 a threshold value is chosen. Given a constraint, the values assigned to $d_{t_1}, d_{t_2}, d_{t_3}$ and $d_{t_4}$

<table>
<thead>
<tr>
<th>Class</th>
<th>Relation</th>
</tr>
</thead>
</table>
| 1.    | Begins together  
     | Ends together  |
| 2.    | Begins just before the beginning  
     | Begins just before the ending  
     | Begins just after the beginning  
     | Begins just after the ending  
     | Ends just before the beginning  
     | Ends just before the ending  
     | Ends just after the beginning  
     | Ends just after the ending  |
| 3.    | Begins before the beginning  
     | Begins before the ending  
     | Begins after the beginning  
     | Begins after the ending  
     | Ends before the beginning  
     | Ends before the ending  
     | Ends after the beginning  
     | Ends after the ending  |

are checked to determine which class they belong to. Then the constraint is assigned a priority as indicated in Table 3.5. Consider the threshold value to be 5 secs. In such a case, a relation is assigned priority 1 if two out of the four values are zero (i.e begin and end together). If one of these values is zero and another is less than the threshold, then it is assigned a priority 2. In a similar manner, priorities are assigned in other cases. The edge in consideration is dropped when its priority happens to be equal to or less than the value
Table 3.5: Conditions for equivalence

<table>
<thead>
<tr>
<th>Temporal Relation</th>
<th>Priority assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combinations of Class 1</td>
<td>1</td>
</tr>
<tr>
<td>Combinations of Class 1 &amp; 2</td>
<td>2</td>
</tr>
<tr>
<td>Combinations of Class 1 &amp; 3</td>
<td>3</td>
</tr>
<tr>
<td>Combinations of Class 2</td>
<td>4</td>
</tr>
<tr>
<td>Combinations of Class 2 &amp; 3</td>
<td>5</td>
</tr>
<tr>
<td>Combinations of Class 3</td>
<td>6</td>
</tr>
</tbody>
</table>

detected in the path. Thus the relaxation policy which is employed when a cycle is detected in the spanning tree, has been designed to retain temporal relations that are closer (defined by the threshold) by assigning them higher priority. In case the priority of the edge under consideration is higher than that detected in the path, it is inserted into the tree and the constraint in the path having the lowest priority is removed to avoid a cycle in the spanning tree, thus resolving inconsistency.

3.3.3 Illustration

The example presented by Ma and Shin (2004) is considered again. The multimedia scenario first presents a logo consisting of a Logo-Animation(A) and Logo-Music(B) for a duration of 20 seconds. Then an Audio(C) is played with a Video(D) for a duration of 20 seconds. A Text(E) is displayed while playing the Video(D). The presentation ends with 10 seconds of an Exit-Animation(F) accompanied by an Exit-Music(G). A background (H) and a caption (I) is present throughout the entire presentation. Assume the screen resolution to be 512 × 512 with the origin at the lower left corner. The background object is assumed to be the source of the presentation, occupying the entire screen having 0 as its z value and commencing at time zero. The above scenario can be specified in the form of the following constraints. The
priority that would be assigned by the algorithm is also indicated.

\[
\begin{align*}
ST1 & : ST(A, B, 0, 0, -10, 10, 0, 0, 0, 0, 0, 1) \\
ST2 & : ST(D, E, 4, 0, -16, 20, 156, -156, 456, -16, 0, 1, 2) \\
ST3 & : ST(C, D, 0, 0, -20, 20, 0, 0, 0, 0, 0, 1) \\
ST4 & : ST(C, G, 20, 10, 0, 30, 0, 0, 0, 0, 0, 3) \\
ST5 & : ST(G, F, 0, 0, -10, 10, 0, 0, 0, 0, 0, 1) \\
ST6 & : ST(C, E, 0, -4, -30, 16, 0, 0, 0, 0, 0, 2) \\
ST7 & : ST(D, F, 15, 5, 0, 30, 56, -56, 56, -56, 0, 1, 2) \\
ST8 & : ST(H, A, 0, -30, -40, 10, 56, -56, 56, -56, 20, 1, 3) \\
ST9 & : ST(A, I, 0, 30, -10, 40, 50, -50, -40, -400, -10, 1, 3) \\
ST10 & : ST(H, D, 10, -10, -30, 30, 0, 0, 0, 20, 1, 6) \\
ST11 & : ST(D, I, -10, 10, -30, 30, 106, -106, 16, -456, -10, 1, 6) \\
ST12 & : ST(H, E, 14, -10, -26, 30, 156, -156, 436, -36, 20, 1, 6) \\
ST13 & : ST(H, F, 30, 0, -10, 40, 56, -56, 56, -56, 20, 1, 3) \\
ST14 & : ST(F, I, -30, 0, -40, 10, 50, -50, -40, -400, -10, 1, 3) 
\end{align*}
\]

In the above example, the relaxation policy would be employed by the algorithm six times, each time dropping a constraint that is deemed least important. The output spanning tree is given in Figure 3.8(c). The structure of the tree after the input of \(ST6\) is shown in Figure 3.8(a) and after \(ST7\) in Figure 3.8(b).

### 3.3.4 Simulation Results

Assume that the presentation has a total of \(m\) constraints dealing with \(n\) media objects. The algorithm has \(n\) \textit{MAKESET} operations and that requires \(O(n)\) time. Also, the algorithm is required to perform \(m\) \textit{FINDSET}
and \( n - 1 \) \( UNION \) operations, which requires \( O(n \log^2(m, n)) \) time, as \( m \geq n \) where \( \alpha \) is an inverse Ackermann’s function (Tarjan 1975). The algorithm also has \( m - n + 1 \) DFS operations requiring time \( O(n(m - n)) \) and one sort operation requiring time \( O(n \log n) \). Hence, the total time complexity of the algorithm is \( O(mn) \) (Cormen et al 1990).

A prototype of the interactive multimedia presentation system described in this chapter has been developed using .NET and has been used effectively to evaluate the \textit{average startup latency} of the presentation system and its \textit{response} to user interactions. The prototype can be used to create multimedia presentations and also to play the presentations. The authoring phase of the prototype helps in understanding how consistency checking is handled dynamically while the presentations are being created. During the playout phase the prototype permits interaction with the structure of the presentation and helps in studying the response time. Figure 3.9 and Figure 3.10 shows the screen shot of the prototype during the authoring phase and playout respectively.

In the prototype the maximum number of media objects that can be involved in a presentation has been restricted to 20. This was done to
ensure the visibility of all the nodes (that represent the media objects) in the graphical representation of the multimedia system. The graphical representation, displayed on one part of the screen, shows how inconsistency is resolved during the authoring process and how incompleteness is dealt with prior to the playout. During playout, the nodes that represent the media objects which are currently being played, are highlighted in the graph. Interactions in the form of changes to the spatio-temporal relations can be modeled and incorporated in the presentation while it is being played. A variety of media objects such as video files (MPEG) of length ranging from 10 secs to 5 min, audio files (MP3)
ranging from 10 secs to 60 secs, apart from text, images and animations were used to study the model.

The prototype has been used to evaluate the initial delay referred to as startup latency. The startup latency depends on the time required to resolve inconsistency, incompleteness and to compute the exact location in time and space for each media object in the presentation. During the execution of the algorithm, the start and end events of each of the media objects are scheduled on a time line. These are used in an event based simulation to control the flow of the presentation. The graph in Figure 3.11 shows the average startup latency when the number of media objects differ in a presentation of duration 20 mins. Figure 3.12 shows the percentage of delay due to startup at the presentation intervals, while that in Figure 3.13 depicts the prototype’s response time to user interaction in the form of deletion of media objects during the presentation. This prototype is being further enhanced to make it a distributed multimedia presentation system. In the enhancement of the prototype, several consistent presentations will be stored in a server which can be accessed by clients over a network. The extended prototype will be able to efficiently handle synchronization of the media objects as well as provide support for interaction, i.e each client would be permitted to customize the presentation during the playout or prior to the playout.

3.4 SUMMARY

The research work presented in this chapter focuses on improving an existing approach presented by Ma and Shin (2004) for consistency checking of temporal and spatial relations. In addition to the simplicity, the algorithm proposed here achieved a reduction of the computation time by a multiplicative constant without changing the asymptotic value, while simul-
Figure 3.11: Startup latency vs media objects

Figure 3.12: Percentage delay

Figure 3.13: Response time for interactions
taneously introducing a dynamic approach toward consistency checking. This chapter also introduces an operator to model spatio-temporal relations to be used as constraints in a multimedia presentation specification. The usage of this operator, further has helped to achieve a constant time reduction in the time complexity while simultaneously being highly responsive to interactions with the presentation during playout. The specification set of the presentation may require support for editing, like addition and deletion of objects and constraints. Addition is handled in the same way as the algorithm Integrated_ConCheck handles any new object or constraint during its execution. Updating of constraints, i.e. changing the temporal and spatial values specified in the spatio-temporal (ST) operator can also be permitted. But this would require that appropriate changes be made to all other related constraints, followed by a regeneration of the layout, as in steps 31 to 35 of Algorithm 3.4. Deletion of a constraint would require the algorithm to check again for completeness and then generate the layout as given in steps 21 to 35. However, deletion of an object requires in addition the deletion of all the constraints associated with that object. Navigation of the authored presentation can be easily supported in the approach presented by Huang and Wang (1998). Interaction with the presentation, (i.e. making permitted alteration to the temporal and spatial layouts) can also be effectively supported by the method. This is done by ensuring that the spanning tree built during authoring is made available by regenerating it in the background while the presentation is being played. The interactions with the presentation are captured again as constraints, and dealt with appropriately for addition, deletion and updation as discussed above.