AN INDEX STRUCTURE FOR MOVING OBJECTS IN VIDEO DATABASES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION SCIENCE AND THE INSTITUTE OF ENGINEERING AND SCIENCE OF BILKENT UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

By

Tuba Yavuz

August, 1999

AN INDEX STRUCTURE FOR MOVING OBJECTS IN VIDEO DATABASES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Tuba Yavuz

August, 1999

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Özgür Ulusoy(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Uğur Güdükbay

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Attila Gürsoy

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Science

ABSTRACT

AN INDEX STRUCTURE FOR MOVING OBJECTS
IN VIDEO DATABASES

Tuba Yavuz
M.S. in Computer Engineering and Information Science
Supervisor: Assoc. Prof. Dr. Özgür Ulusoy
August, 1999

Modeling moving objects and handling various types of motion queries are interesting topics to investigate in the area of video databases. In one type of motion queries, motion of multiple objects is specified by the changes in relative spatial positions of objects. Answering such kind of queries, that involve motion of multiple objects whose identifications are not specified, requires some type of indexing because the time complexity of processing such a query in the absence of an index structure is $O(N!/(N-n)!)$, where N is the number of objects in the database and n is the number of objects in the query. In this work, we propose a spatio-temporal index structure, which we call *SMIST*-index, and compare its performance against a similar scheme proposed in [18]. The scheme presented in [18] consists of a constraint satisfaction algorithm, which is called Join Window Reduction (JWR), combined with a spatial index structure (R*-tree). Experimental results indicate that *SMIST*-index outperforms the JWR algorithm. Also, *SMIST*-index is shown to be scalable to increasing number of frames and objects.

Key words: Motion, query, video, database, multimedia, spatial, temporal, spatio-temporal indexing.

ÖZET

VİDEO VERİ TABANLARINDA HAREKET EDEN NESNELER İÇİN BİR İNDEKS YAPISI

Tuba Yavuz
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Özgür Ulusoy
Ağustos, 1999

Video veri tabanı alanında, hareketli nesnelerin modellenmesi ve çeşitli hareket sorgularının cevaplanması oldukça ilgi çeken bir araştırma konusu olmuştur. Hareket sorgularının bir çeşidinde birden fazla nesnenin hareketleri birbirlerine göre olan yerlerindeki değişiklikle ifade edilmektedir. Birbirlerine göre uzaysal ilişkileri belirtilmiş fakat kimlikleri belirtilmemiş nesnelerden oluşan bu tip sorguların cevaplanması özel bir indeks yapısının kullanılmasını gerektirir. Bunun nedeni, böyle bir sorgunun herhangi bir indeks yapısı kullanılmaksızın cevaplanmasının hesaplama karmaşıklığı $O(N!/(N-n)!)$ olmasıdır. Burada N veri tabanındaki nesne sayısını, n ise sorguda bulunan nesne sayısını gösterir. Biz bu çalışmada *SMIST*-indeks diye isimlendirdiğimiz uzaysal ve zamansal bir indeks yapısı geliştirdik. Bu indeks yapısının performansını incelediğimiz sorgu çeşidinin cevaplanması için önerilmiş bir yöntemle ([18]) karşılaştırdık. Deney sonuçları *SMIST*-indeks yapısının karşılaştırdığımız yöntemden daha iyi bir performans sergilediğini gösterdi. Ayrıca, yapılan deneylerde önerdiğimiz indeks yapısının artan çerçeve ve nesne sayısı karşısında disk ulaşım sayısında keskin artışlar göstermediği saptanmıştır.

Anahtar kelimeler: Hareket, sorgu, video, veri tabanı, çoklu ortam, uzaysal, zamansal, uzaysal-zamansal indeksleme.

Dedicated to

my parents **Cahide** and **Mustafa Yavuz**,

my brothers **Osman Yüksel** and **Emre**,

and my husband **Tamer Kahveci**.

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to Assoc. Prof. Özgür Ulusoy for his patient supervision and invaluable guidance. I would like to thank Kasım Selçuk Candan for his invaluable comments. I am grateful to Asst. Prof. Uğur Güdükbay and Asst. Prof. Attila Gürsoy for reading this thesis and their comments. I would also like to mention the support and sacrifice of my family. Finally, I would like to thank my friend Kadriye Özbaş for her moral support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The type of data stored in database systems is no longer only alphanumeric but also image, audio and video. This is the natural consequence of emerging use of multimedia database systems in application domains such as surveillance systems, medicine, entertainment, and education. However, since new data types bring their own requirements such as large storage requirement and excessive content processing, in order to achieve effective use of multimedia data a number of design issues such as data modeling, query formulation, and efficient retrieval should be reconsidered.

Multimedia data requires excessive processing and a detailed modeling of its content. This is because it is rich in content and this fact is also stressed by the well-known saying "A picture worths a thousand words".

Two main approaches have been followed for retrieval of multimedia data. These approaches are called *annotation-based retrieval* and *content-based retrieval.* Annotation-based retrieval failed to be the appropriate paradigm for designing systems that utilize the information hidden in the database as much as possible. In [13], the reason for that is explained as follows:

> Multimedia objects, such as images, have many different attributes that could be of interest-there is no way that any tag could hope to capture them all. The trick, in any application context, is to identify commonly used characterizations and to build tags based on these.

Even so, there is loss of information.

The paper concludes that queries should be posed to objects themselves not to the tags. To put in other words, retrieval should be by content. Researchers, having realized the need for content-based retrieval, have developed various systems providing content-based retrieval of multimedia data. Some examples are [13], [1], and [8].

Content-based retrieval is a huge research area, which has a number of components such as preprocessing, data modeling, and indexing. The work presented in this thesis corresponds to content-based indexing for video databases. We propose a new spatio-temporal index structure. Such an index can facilitate answering queries in which the motion of multiple objects is defined in terms of the change in objects' relative spatial positions and the identities of the objects are undefined. We compare the performance of our query execution model that uses the proposed index structure with the framework presented in [9], which is the only work that deals with the same type of query as the one we are interested in our work.

In [18], Papadias et al. used binary string encoding scheme for 1-D relations. The scheme allows automatic derivation of similarity degree between spatial structures. Structural similarity queries are handled using this scheme. R*-trees are used as the index structure. A constraint satisfaction algorithm, which is called *forward checking with variable reordering* [6], is used to reduce the search space. The main idea behind the constraint satisfaction algorithm is that as each variable is instantiated, the search space for the uninstantiated variables is restricted according to the constraints of the instantiated one.

In our work, we propose a new spatio-temporal index structure which is based on B+-trees. We use topological-directional relations to represent the spatial relations between objects. The index structure we propose organizes the search space first according to spatial dimension and then according to temporal dimension. We also propose a query execution algorithm that uses our index structure efficiently. The algorithm consists of a filter step and a refine step. In the filter step, by using the spatial constraints in the query, time intervals that possibly include answer(s) to the query are selected. Then in the

refine step, the data corresponding to the filtered time intervals is analyzed and all the spatial constraints are checked to find exact matches.

The rest of the thesis is organized as follows. In Chapter 2, we summarize the related work. The new index structure we propose is explained in detail in Chapter 3. Chapter 4 describes the algorithm we developed for query execution. Implementation details are explained in Chapter 5. Experimental settings and the results obtained are provided in Chapter 6. Finally, in Chapter 7 conclusions and future work are discussed.

Chapter 2

Related Work

Modeling moving objects has attracted attention of researchers since motion of objects can serve as a valuable source of information that can be used to handle video sequences in multimedia databases. Majority of the work that has been done on motion analysis belongs to the signal processing field. The research in this field has enabled shot change detection [7], video segmentation [4], [17], [20], [21], camera motion extraction [2], [15], [7], and tracking moving objects [7] which provides valuable information that can be used for answering queries of video database users regarding motion.

Apart from these studies which focus on preprocessing of raw data to extract information regarding motion, there has also been research on modeling moving objects [10], [14], motion query formulation [1], and spatial indexing for motion queries [18].

In [10], Dimitrova et al. focus on both recovery of motion information and its use for classification and query processing. Therefore, their work covers both motion analysis in digital video and information modeling and retrieval. Besides storing physical information of video, they use the spatial and temporal properties of objects to derive the conceptual video data type. They define an algebraic model for multimedia data. In addition, their model is completed with an algebraic query language called EVA. EVA has the ability to deal with spatial and temporal dimensions of multimedia objects. As a result users

can formulate queries regarding motion of objects using EVA. The authors think that a knowledge-base, which includes all necessary rules, constraints and procedures, should be used to associate object trajectory representations, which is obtained as a result of the low-level motion analysis steps (motion detection and motion tracing), with the domain dependent "activities". For instance, for an application domain which includes only cars as the moving objects, a specific activity such as *driving straight* can be defined. EVA is further extended with a graphical user interface that uses a query language called VEVA. VEVA is a visual query language so that users can visually express their queries. As an example, a query, in which the path of an object is specified by drawing it, can be posed.

In [1], motion is considered as a primary attribute and differentiated as camera (global) motion and an object (local) motion. Since combination of the camera motion and object motion produces the recorded scenes, in order to capture the real motion of the object, camera's motion should be subtracted. This separation also helps explicit capture of camera motion so that shots in which particular types of camera operations performed can be retrieved. In their query formulation system prototype, which is called MovEase (Motion Video Attribute Selector), users are provided with querying by visual descriptions. It is discussed in the paper that this way is more effective for describing different kinds of motion, particular paths of object movement and combination of different motion types than in the case expressing with keywords and relational operators. The graphical user interface of MovEase consists of three regions: icon catalog browser, query builder, and result browser. User can select objects and motion types from the icon catalog browser and formulates a query in the query builder. Results are displayed in the result browser. The query system performs both exact match and fuzzy matching. The user can specify camera motion and associate motion information such as speed and path for each object. Objects with similar motion characteristics can be grouped. Scenes that are composed of multiple camera motions can also be queried.

In [14], a qualitative way of representing object trajectory and relative spatio-temporal relationships between moving objects is introduced. *Strict*

directional relations (north, south, west, and east) and *mixed directional relations* (northeast, northwest, southeast, and southwest) are used for trajectory representation. To represent the spatial relations between moving objects, combination of directional and topological relations are used. Directional relations consist of *strict directional relations*, *mixed directional relations*, and *positional relations* (left, right, above, and below). Topological relations are the eight fundamental *topological relations* that can hold between two planar regions. They are specified by Egenhofer [11] as disjoint, contains, inside, meet, equal, covers, covered-by, and overlap. Additionally, both exact match and fuzzy match algorithms are given for trajectories and spatio-temporal relations of moving objects. Li et al. have integrated their moving object model in an Object Oriented Database Management system called TIGUKAT [14].

In [18], Papadias et al. provide a framework for structural media similarity. Motion queries are considered as a sequence of structural similarity queries where the set of objects is the same for each step. Here the word "structural" refers to the spatial relations among the objects in a multimedia document. Binary string encoding is used for 1D relations. This encoding can be extended to multiple resolution levels and dimensions. Furthermore, since similarity measures can be automatically derived from this encoding, fuzzy queries are also supported. Given a motion query, which consists of object spatial relationship constraints valid for subsequent frames, a constraint satisfaction algorithm (forward checking with dynamic variable reordering) that is combined with R*-trees is applied. Detailed description of the framework is given in Section 2.1.

Our work deviates from the others except [18], as we consider multiple object motion queries in which object identities are undefined. The main difference between [18] and our work is that we use a spatio-temporal index while they use a spatial access method (R*-tree). Using a spatial access method requires examining each frame separately. However, a spatio-temporal index does not only eliminates objects which do not satisfy spatial constraints but also eliminates frame sequences which do not include similar positioning of objects as specified in the query. Another difference is that in [18] spatial relations between each object is represented by 1D strings in each dimension while we use

the combination of *topological* and *directional* relations, which are also used in [14].

2.1 A Framework for Querying Structural Similarity

In [18], Papadias et al. have analyzed the query type that involves the retrieval of a set of objects each of which is related to the others by a spatial constraint. They present a unified framework for structural similarity. This framework permits the definition of any type of spatial relation and automatic derivation of similarity measures.

In this framework, objects' spatial relations between each other are expressed in each dimension separately. For representing one dimensional spatial relations, 1D relations are used. Given a reference interval $[a, b]$, nine regions of interest are identified:

1. $(-\infty, a - \delta)$
2. $[a - \delta, a - \delta]$
3. $(a - \delta, a)$
4. $[a, a]$
5. (a, b)
6. $[b, b]$
7. $(b, b + \delta)$
8. $[b + \delta, b + \delta]$
9. $(b + \delta, +\delta)$

For each of the above regions the binary variables $r, s, t, u, v, w, x, y, z$ are assigned, respectively. Figure 2.1 illustrates the association of these variables with the above regions.

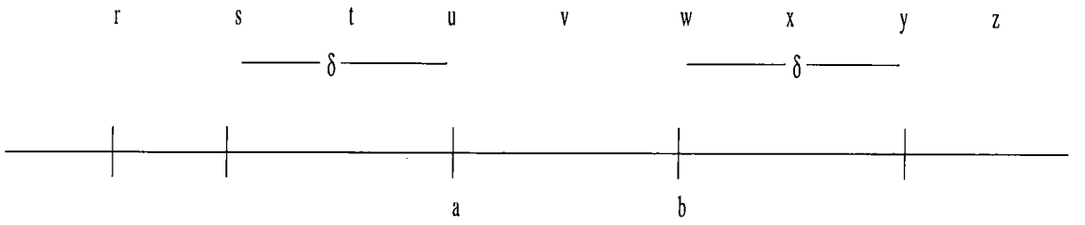


Figure 2.1: Nine region of interest for 1D relation representation

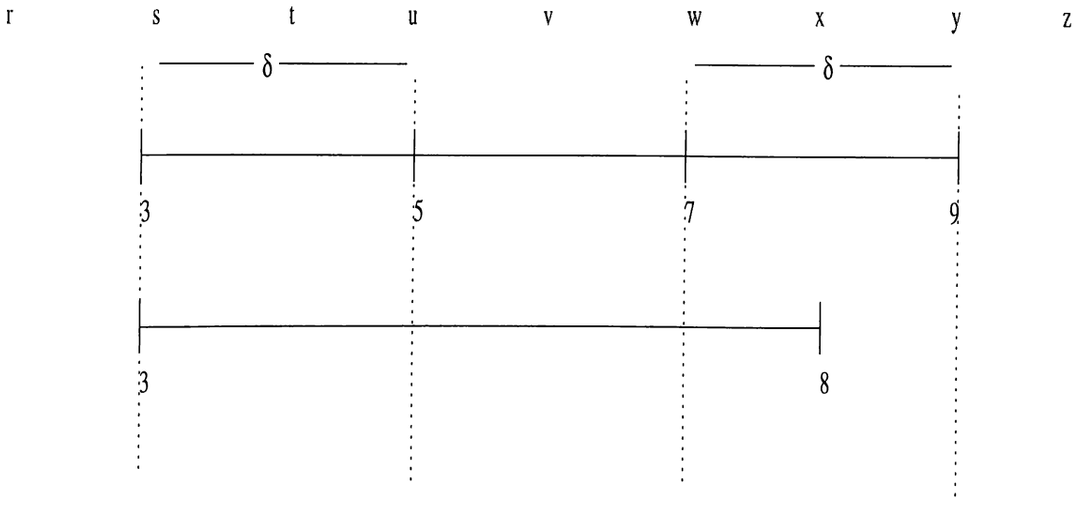


Figure 2.2: Relative positioning of the intervals [3,8] and [5,7]

Given an interval $[c, d]$, a 1D relation is constructed by setting the value of each variable according to the result of intersection between $[c, d]$ and the corresponding region for that variable. For example, the 1D relation between [3,8] and [5,7] is "011111100" in the case of $\delta = 2$. Figure 2.2 illustrates intersecting points of these two intervals.

In this framework, degree of the similarity is found by calculating the *distance* of 1D relations in each dimension and summing them up. Figure 2.3 shows the algorithm that calculates the distance between two 1D relations.

In this work, structural similarity queries are formalized as a binary constraint satisfaction problem [16]. It consists of:

1. A set of n variables $o_0, o_1, ..., o_{n-1}$, which are objects that appear in the query.

Algorithm *distance*(1D relation s1,1D relation s2)
1 *set* $s1$ OR $s2$ to s
2 *initialize distance* to 0
3 **for** each binary variable $b \in \{r, s, t, t, v, w, x, y, z\}$ **do**
4 **if** $s1.b == 0$ **then**
5 $distance = distance + 1$
6 **if** $s2.b == 0$ **then**
7 $distance = distance + 1$

Figure 2.3: The algorithm for finding the distance between two 1D relations

2. For each variable o_i a finite domain D_i of size N, which is the number of objects in the frame, exists. It is assumed that all domains are identical.

3. For each pair of variables o_i and o_j, a spatial constraint C_{ij}, which is expressed in terms of a set of 1D relations, exists.

One of the algorithms that has been proposed for solving constraint satisfaction problem is *forward checking* [5]. It has been shown that it outperforms the other constraint satisfaction algorithms for a wide range of problems [6]. The *forward checking* algorithm prunes the domain of variables $o_{i+1}, o_{i+2}, ..., o_{n-1}$ when o_i is instantiated. Therefore, when the first m, $1 \leq m \leq n$, variables are instantiated, the values of $o_0, o_1, ..., o_{m-1}$ constitute a partial solution. Besides, the domain of the uninstantiated variables include values that may lead to a complete solution. After applying the *Check Forward* algorithm, if an empty domain results, then the value of o_i is cancelled and domain values of the variables $o_{i+1}, o_{i+2}, ..., o_{n-1}$ are restored. If the values in the domain of variable o_i becomes exhausted, the algorithm returns to the previous variable, o_{i-1}, and assigns a new value to it.

Check forward algorithm is enhanced with a technique called *Dynamic Value Reordering* (DVR) [6]. *Dynamic Value Reordering* helps the *check forward* algorithm in choosing the next variable to be instantiated. To minimize the search path, it reorders the uninstantiated variables according to their domain size. As a result the variable with the smallest domain size becomes the next variable to be instantiated. Figure 2.4 and Figure 2.5 show the *Forward checking with dynamic variable reordering* algorithm.

```
Algorithm FC-DVO(Query q)
1   for j = 0 to n − 1 do
2      domain[0][j] = D
3   i = 0
4   while (True)
5      for each value u_j ∈ domain[i][i] do
6         set instantiations[i] to u_j
7         if i = n − 1 then
8            output instantiations
9         else
10           if check-forward(i) then
11              DVR(i + 1, n − 1)
12              i = i + 1
```

Figure 2.4: Instantiation algorithm

```
Algorithm check-forward(int i)
1   for j = i + 1 to n − 1 do
2      domain[i + 1][j] = domain[i][j]
3         for each value u_j ∈ domain[i + 1][j] do
4            if distance(C_ij, R(instantiations[i], u_j)) > 0 then
5               domain[i + 1][j] = domain[i + 1][j] − u_j
6         if domain[i + 1][j] is empty then
7            return False
8   return True
```

Figure 2.5: *Check Forward* algorithm

Domain of each variable in each instantiation level is kept in the *domain* table. It is of size $n \times n \times N$. At the beginning of the *FC-DVO* algorithm, domain of all the variables ($domain[0][j]$) is initialized to the whole search space, i.e., all the objects in the image.

Papadias et al., have developed three algorithms for answering structural similarity queries. These algorithms are called *Multilevel Forward Checking*, *Window Reduction*, and *Join Window Reduction*. The authors use R-trees in order to make their design to be applicable to large spatial databases. According to their experiments, *Join Window Reduction* algorithm outperforms the others. Therefore, we will compare our proposed scheme with their scheme that uses the *Join Window Reduction* algorithm. Before going on to describe the *Join Window Reduction* algorithm, we want to give some information about the other algorithms since they use some common methods.

The *Multilevel Forward Checking* algorithm performs multi-way spatial self-join. By using the enclosure property of the R-trees, it is decided to move to the lower level of the tree or not. This is because if two intermediate levels do not intersect then the Minimum Bounding Rectangles (MBRs) below them do not intersect, either.

The spatial constraint C_{ij} is used when two variables are instantiated. However, for the intermediate nodes and leaf nodes of the tree this constraint cannot be used in deciding on whether next level should be followed or not. For this reason, *internal node bounding conditions* and *leaf bounding conditions* are used at intermediate levels and leaf level, respectively. Tables 2.1 and 2.2 show the *internal node bounding conditions* and Tables 2.3 and 2.4 show the *leaf bounding conditions* according to a given 1D relation.

Since the constraints between intermediate nodes are in general too loose, a large number of intermediate nodes are visited in the *Multilevel Forward Checking* algorithm. An alternative algorithm, called *Window Reduction* algorithm, uses data MBRs in instantiating the variables and employs *forward checking* with R-trees to efficiently prune the domains. Since data MBRs are assigned to instantiations of variables the *domain* table shrinks to 2D from 3D and its size becomes $n \times n$. When the variable o_i is instantiated, *Check Forward* algorithm,

1D Relation	Bounding Condition
1XXXXXXXX	$N_i.l < N_j.u - \delta$
01XXXXXXX	$N_i.l \leq N_j.u - \delta$
001XXXXXX	$N_i.l < N_j.u$
0001XXXXX	$N_i.l \leq N_j.u$
00001XXXX	$N_i.l \leq N_j.u$
000001XXX	$N_i.l \leq N_j.u$
0000001XX	$N_i.l < N_j.u + \delta$
00000001X	$N_i.l \leq N_j.u + \delta$
000000001	$N_i.l$ unlimited

Table 2.1: Internal node bounding condition for left most bit

1D Relation	Bounding Condition
XXXXXXXX1	$N_i.u > N_j.l + \delta$
XXXXXXX10	$N_i.u \geq N_j.l + \delta$
XXXXXX100	$N_i.u > N_j.l$
XXXXX1000	$N_i.u \geq N_j.l$
XXXX10000	$N_i.u \geq N_j.l$
XXX100000	$N_i.u \geq N_j.l$
XX1000000	$N_i.u > N_j.l - \delta$
X10000000	$N_i.u \geq N_j.l - \delta$
100000000	$N_i.u$ unlimited

Table 2.2: Internal node bounding condition for right most bit

updates domain of the uninstantiated variables $o_{i+1}, o_{i+2}, ..., o_{n-1}$ to windows whose bounds are decided according to the *domain window bound* conditions given in Tables 2.5 and 2.6. At the beginning of the algorithm, domains of all the variables are set to the universal space.

In *Window Reduction* algorithm, in order to instantiate the first variable whole universe is searched. Instead of instantiating the first variable in the whole universe, the first two variables can be instantiated by using a pairwise spatial join. Then window reduction can be used to instantiate the rest of the variables. This execution plan is called *Join Window Reduction* algorithm. Figure 2.6 presents the algorithm.

1D Relation	Bounding Condition
1XXXXXXXX	$N_i.l < N[j].u - \delta$
01XXXXXXX	$N_i.l < N[j].u - \delta$,$N_i.l \geq N[j].l - \delta$
001XXXXXX	$N_i.l < N[j].u$,$N_i.l > N[j].l - \delta$
0001XXXXX	$N_i.l < N[j].u$,$N_i.l > N[j].l$
00001XXXX	$N_i.l < N[j].u$,$N_i.l > N[j].l$
000001XXX	$N_i.l < N[j].u$,$N_i.l > N[j].l$
0000001XX	$N_i.l < N[j].u + \delta$,$N_i.l > N[j].l$
00000001X	$N_i.l \leq N[j].u + \delta$,$N_i.l > N[j].l + \delta$
000000001	$N_i.l > N[j].l + \delta$

Table 2.3: Leaf node bounding condition for left most bit

1D Relation	Bounding Condition
XXXXXXXX1	$N_i.u > N[j].l + \delta$
XXXXXXX10	$N_i.u > N[j].l + \delta$,$N_i.u \leq N[j].u + \delta$
XXXXXX100	$N_i.u > N[j].l$,$N_i.u < N[j].u + \delta$
XXXXX1000	$N_i.u > N[j].l$, $N_i.u < N[j].u$
XXXX10000	$N_i.u > N[j].l$, $N_i.u < N[j].u$
XXX100000	$N_i.u > N[j].l$, $N_i.u < N[j].u$
XX1000000	$N_i.u > N[j].l - \delta$, $N_i.u < N[j].u$
X10000000	$N_i.u \geq N[j].l - \delta$, $N_i.u < N[j].u - \delta$
100000000	$N_i.u < N[k].u - \delta$

Table 2.4: Leaf node bounding condition for right most bit

1D Relation	Bounding Condition
1XXXXXXXX	$W_i.l = -\infty$
01XXXXXXX,001XXXXXX	$W_i.l = N_j.l - \delta$
0001XXXXX,00001XXXX	$W_i.l = N_j.l$
000001XXX,0000001XX	$W_i.l = N_j.u$
00000001X,000000001	$W_i.l = N_j.u + \delta$

Table 2.5: Domain window bound conditions for left most bit

1D Relation	Bounding Condition
XXXXXXXX1	$N_i.u = +\infty$
XXXXXXX10,XXXXXX100	$W_i.u = N_j.u + \delta$
XXXXX1000,XXXX10000	$W_i.u = N_j.u$
XXX100000,XX1000000	$W_i.u = N_j.l$
X10000000,100000000	$W_i.u = N_j.l - \delta$

Table 2.6: Domain window bound conditions for right most bit

Algorithm *JWR*(Query q)
1 **for** $j = 0$ to $n - 1$ **do**
2 $domain[1][j] = U$ /* U is the Universal Space
3 instantiate the first two variables using multi way spatial join
4 $i = 1$
5 **while** (True)
6 **if** $i == 1$ **then**
7 *get* next pair of instantiations of the first variables
8 **else**
9 *get newvalue* from $domainWindow[i][i]$
10 **if** $newvalue = NULL$ **then**
11 $i = i - 1$
12 **if** $distance(C_{ij}, R(instantiations[i], newvalue)) = 0$ **then**
13 $instantiations[i] = newvalue$
14 **if** $i == n - 1$ **then**
15 *output instantiations*
16 **else**
17 **if** window-reduction(i) **then**
18 Window$_{DVO}$(i+1,n-1)
19 i=i+1

Figure 2.6: *Join window reduction* algorithm

Chapter 3

Index Creation

Designing an index structure for spatial relationships of video objects requires consideration of both spatial and temporal dimensions due to the temporal nature of video data. Considering this aspect, our index structure, which we call *SMIST*-index (**S**patial **M**aximal **I**ntersecting Interval **S**ets **T**ree) aims to organize video data both spatially and temporally. Our index classifies the data first according to spatial properties and then according to temporal properties of the data. We will follow this order in describing the construction of our index structure. Section 3.1 and Section 3.2 explain these two steps, respectively. In order to clarify the construction of *SMIST*-index we will use the example frame sequence given in Figure 3.1.

3.1 Spatial Organization

Before explaining the classification of data according to spatial properties, we present the formalism we use for representing the spatial relationships between objects in Subsection 3.1.1. Then in Subsection 3.1.2, we describe the way *SMIST*-index partitions data according to its spatial properties.

3.1.1 Spatial Representation

We assume that objects are 2-dimensional and have rectangular shapes. It is an accepted convention to represent object boundaries with their minimum bounding rectangles in spatial applications [19].

We choose to use 8 directional and 8 topological relations whose definitions are based on Allen's temporal interval relationships [3]. Table 3.1 lists Allen's temporal interval relationships, and Tables 3.2 and 3.3 list directional and topological relations and their definitions in terms of Allen's temporal interval relationships, respectively. In Table 3.1, $[s_i, e_i]$ denotes an interval where s_i is the starting point and e_i is the ending point of that interval. In Tables 3.2 and 3.3, o_i denotes an object with identification number i while $[x_1, x_2]_{o_i}$ and $[y_1, y_2]_{o_i}$ denote the extent of object o_i on the x and y dimensions, respectively. The set notation used in Definition column of Table 3.3 represents disjunction of all the relations in the set. For example, $[x_1, x_2]_{o_1}\{\text{b,m}\}[x_1, x_2]_{o_2}$ is equivalent to $[x_1, x_2]_{o_1}\text{b}[x_1, x_2]_{o_2} \vee [x_1, x_2]_{o_1}\text{m}[x_1, x_2]_{o_2}$.

Relationship	Symbol	Inverse	Condition
$[s_1, e_1]$ before $[s_2, e_2]$	b	bi	$e_1 < s_2$
$[s_1, e_1]$ meets $[s_2, e_2]$	m	mi	$e_1 = s_2 - 1$
$[s_1, e_1]$ overlaps $[s_2, e_2]$	o	oi	$s_1 < s_2 \wedge s_2 < e_1$
$[s_1, e_1]$ during $[s_2, e_2]$	d	di	$s_1 > s_2 \wedge e_1 < e_2$
$[s_1, e_1]$ starts $[s_2, e_2]$	s	si	$s_1 = s_2 \wedge e_1 < e_2$
$[s_1, e_1]$ finishes $[s_2, e_2]$	f	fi	$s_1 > s_2 \wedge e_1 = e_2$
$[s_1, e_1]$ equal $[s_2, e_2]$	e	e	$s_1 = s_2 \wedge e_1 = e_2$

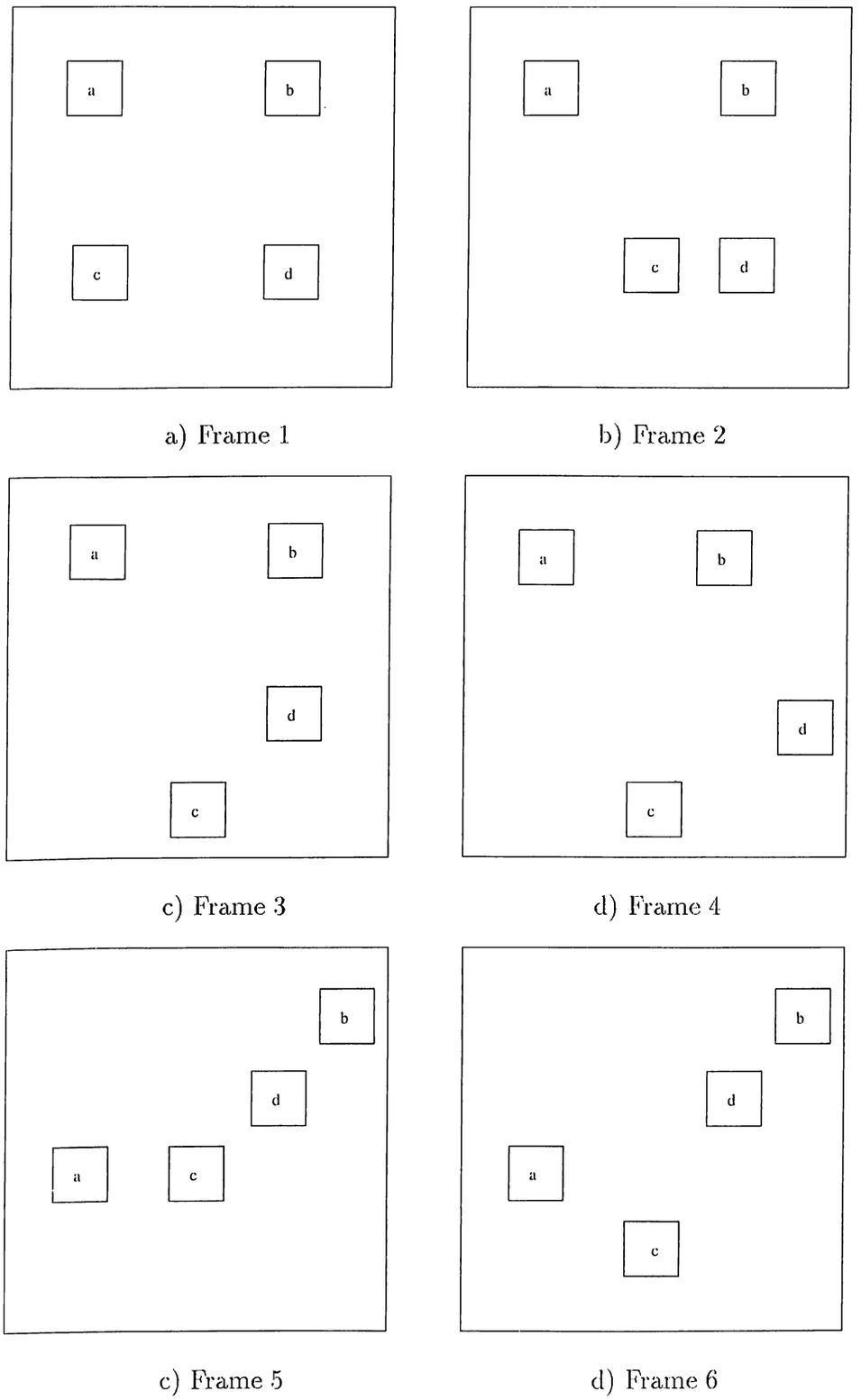
Table 3.1: Interval Relationships

Let S denote the set of 8 directional relations and null (for no directional relation),

$$S = \{north, south, east, west, northeast, northwest, southeast, southwest, null\}$$

and T denote the set of 8 topological relations,

$$T = \{equal, inside, contain, cover, coveredby, overlap, touch, disjoint\}.$$



a) Frame 1 b) Frame 2

c) Frame 3 d) Frame 4

c) Frame 5 d) Frame 6

Figure 3.1: Data for moving objects a, b, c, and d

Relation	Definition
o_1 south o_2	$[x_1, x_2]_{o_1}$ {d,di,s,si,f,fi,e} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {b,m} $[y_1, y_2]_{o_2}$
o_1 north o_2	$[x_1, x_2]_{o_1}$ {d,di,s,si,f,fi,e} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {bi,mi} $[y_1, y_2]_{o_2}$
o_1 west o_2	$[x_1, x_2]_{o_1}$ {b,m} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {d,di,s,si,f,fi,e} $[y_1, y_2]_{o_2}$
o_1 east o_2	$[x_1, x_2]_{o_1}$ {bi,mi} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {d,di,s,si,f,fi,e} $[y_1, y_2]_{o_2}$
o_1 south-west o_2	$([x_1, x_2]_{o_1}$ {b,m} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {b,m,o} $[y_1, y_2]_{o_2}) \vee$ $([x_1, x_2]_{o_1}$ {o} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {b,m} $[y_1, y_2]_{o_2})$
o_1 south-east o_2	$([x_1, x_2]_{o_1}$ {bi,mi} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {b,m,o} $[y_1, y_2]_{o_2}) \vee$ $([x_1, x_2]_{o_1}$ {oi} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {b,m} $[y_1, y_2]_{o_2})$
o_1 north-west o_2	$([x_1, x_2]_{o_1}$ {b,m} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {bi,mi,oi} $[y_1, y_2]_{o_2}) \vee$ $([x_1, x_2]_{o_1}$ {o} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {bi,mi} $[y_1, y_2]_{o_2})$
o_1 north-east o_2	$([x_1, x_2]_{o_1}$ {bi,mi} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {bi,mi,oi} $[y_1, y_2]_{o_2}) \vee$ $([x_1, x_2]_{o_1}$ {oi} $[x_1, x_2]_{o_2} \wedge [y_1, y_2]_{o_1}$ {bi,mi} $[y_1, y_2]_{o_2})$

Table 3.2: Definition of Directional Relations

Cartesian product of the sets S and T includes the elements which are impossible to occur relationships such as equal-north, inside-northwest, etc. Let I denote the set of such impossible topological-directional relations; ST, which will be used to denote the set of all possible topological-directional relations, can be defined as

$$ST = S \times T - I$$

or

$$ST = \{equal-null, inside-null, contain-null, cover-null, coveredby-null, overlap-null, touch-north, touch-south, touch-east, touch-west, touch-northwest, touch-northeast, touch-southwest, touch-southeast, disjoint-north, disjoint-south, disjoint-east, disjoint-west, disjoint-northwest, disjoint-northeast, disjoint-southwest, disjoint-southeast\}$$

3.1.2 Spatial Partitioning

On the basis of the definitions given in the previous subsection, we provide the following formulation for spatio-temporal data on which SMIST-index is constructed.

	a	b	c	d
a	–	(d-w,1,4) (d-sw,5,6)	(d-n,1,1) (d-nw,2,4) (d-w,5,5) (d-nw,6,6)	(d-nw,1,4) (d-sw,5,6)
b	(d-e,1,4) (d-ne,5,6)	–	(d-ne,1,6)	(d-n,1,3) (d-nw,4,4) (d-ne,5,6)
c	(d-s,1,1) (d-se,2,4) (d-e,5,5) (d-se,6,6)	(d-sw,1,6)	–	(d-w,1,2) (d-sw,3,6)
d	(d-se,1,4) (d-ne,5,6)	(d-s,1,3) (d-se,4,4) (d-sw,5,6)	(d-e,1,2) (d-ne,3,6)	–

Figure 3.2: Spatio-temporal relations for data in Figure 3.1

Relation	Definition
o_1 equal o_2	$[x_1,x_2]_{o_1}\{\text{e}\}[x_1,x_2]_{o_2} \wedge [y_1,y_2]_{o_1}\{\text{e}\}[y_1,y_2]_{o_2}$
o_1 inside o_2	$[x_1,x_2]_{o_1}\{\text{d}\}[x_1,x_2]_{o_2} \wedge [y_1,y_2]_{o_1}\{\text{d}\}[y_1,y_2]_{o_2}$
o_1 contain o_2	$[x_1,x_2]_{o_1}\{\text{di}\}[x_1,x_2]_{o_2} \wedge [y_1,y_2]_{o_1}\{\text{di}\}[y_1,y_2]_{o_2}$
o_1 overlap o_2	$([x_1,x_2]_{o_1}\{\text{o,oi}\}[x_1,x_2]_{o_2}$ $\wedge [y_1,y_2]_{o_1}\{\text{d,di,s,si,f,fi,e,o,oi}\}[y_1,y_2]_{o_2}) \vee$ $([x_1,x_2]_{o_1}\{\text{d,di,s,si,f,fi,e,o,oi}\}[x_1,x_2]_{o_2}$ $\wedge [y_1,y_2]_{o_1}\{\text{o,oi}\}[y_1,y_2]_{o_2})$
o_1 cover o_2	$([x_1,x_2]_{o_1}\{\text{di}\}[x_1,x_2]_{o_2} \wedge [y_1,y_2]_{o_1}\{\text{fi,si,e}\}[y_1,y_2]_{o_2}) \vee$ $([x_1,x_2]_{o_1}\{\text{e}\}[x_1,x_2]_{o_2} \wedge [y_1,y_2]_{o_1}\{\text{di,fi,si}\}[y_1,y_2]_{o_2}) \vee$ $([x_1,x_2]_{o_1}\{\text{fi,si}\}[x_1,x_2]_{o_2} \wedge [y_1,y_2]_{o_1}\{\text{di,fi,si,e}\}[y_1,y_2]_{o_2})$
o_1 coveredby o_2	$([x_1,x_2]_{o_1}\{\text{d}\}[x_1,x_2]_{o_2} \wedge [y_1,y_2]_{o_1}\{\text{f,s,e}\}[y_1,y_2]_{o_2}) \vee$ $([x_1,x_2]_{o_1}\{\text{e}\}[x_1,x_2]_{o_2} \wedge [y_1,y_2]_{o_1}\{\text{d,f,s}\}[y_1,y_2]_{o_2}) \vee$ $([x_1,x_2]_{o_1}\{\text{f,s}\}[x_1,x_2]_{o_2} \wedge [y_1,y_2]_{o_1}\{\text{d,f,s,e}\}[y_1,y_2]_{o_2})$
o_1 touch o_2	$([x_1,x_2]_{o_1}\{\text{m,mi}\}[x_1,x_2]_{o_2} \wedge$ $[y_1,y_2]_{o_1}\{\text{d,di,s,si,f,fi,o,oi,m,mi,e}\}[y_1,y_2]_{o_2}) \vee$ $([x_1,x_2]_{o_1}\{\text{d,di,s,si,f,fi,o,oi,m,mi,e}\}[x_1,x_2]_{o_2}$ $\wedge [y_1,y_2]_{o_1}\{\text{m,mi}\}[y_1,y_2]_{o_2})$
o_1 disjoint o_2	$[x_1,x_2]_{o_1}\{\text{b,bi}\}[x_1,x_2]_{o_2} \vee [y_1,y_2]_{o_1}\{\text{b,bi}\}[y_1,y_2]_{o_2}$

Table 3.3: Definition of Topological Relations

Definition 3.1 A *spatio-temporal relation* is a 5-tuple (o_1, o_2, td_i, s, e) which specifies the spatial relation between two objects at a specified interval, where

- o_1 and o_2 are object identities,
- td_i is a topological-directional relation ($td_i \in ST$),
- s is the starting frame number and e is the ending frame number.

Figure 3.2 shows the *spatio-temporal relations* for the example data in Figure 3.1. In this figure, data is illustrated in grid format. Each cell x, y of the grid, represents the topological-directional relation between objects x and y, which is valid during an interval. For example, the data (d-w,1,4) in grid (a, b) shows that object a is to the west of object b and being disjoint to b during the frame interval [1,4], and it corresponds to *spatio-temporal relation* (a,b,d-w,1,4) according to Definition 3.1.

Relation Type	*Spatio-Temporal Relation* Set
disjoint-west	{(a,b,d-w,1,4),(c,d,d-w,1,2),(a,c,d-w,5,5)}
disjoint-east	{(b,a,d-e,1,4),(d,c,d-e,1,2),(c,a,d-e,5,5)}
disjoint-northwest	{(a,d,d-nw,1,4),(a,c,d-nw,2,4),(b,d,d-nw,4,4), (a,c,d-nw,6,6)}
disjoint-southeast	{(d,a,d-se,1,4),(c,a,d-se,2,4),(d,b,d-se,4,4), (c,a,d-se,6,6)}
disjoint-north	{(a,c,d-n,1,1),(b,d,d-n,1,3)}
disjoint-south	{(c,a,d-s,1,1),(d,b,d-s,1,3)}
disjoint-northeast	{(b,c,d-ne,1,6),(b,a,d-ne,5,6),(d,a,d-ne,5,6), (d,c,d-ne,3,6),(b,d,d-ne,5,6)}
disjoint-southwest	{(c,b,d-sw,1,6),(a,b,d-sw,5,6),(a,d,d-sw,5,6), (c,d,d-sw,3,6),(d,b,d-sw,5,6)}

Table 3.4: *SMIST*-index for data in Figure 3.1 after the first step of its construction

SMITS constructs a table, whose size is equal to the cardinality of ST and assigns an index for each topological-directional relation (for each element of ST) and adds each *spatio-temporal relation* to the corresponding entry of the table. Table 3.4 shows the table which is constructed for the data in Figure 3.1. Since only the topological-directional relations (disjoint-west, disjoint-east, disjoint-northwest, disjoint-north, disjoint-south, disjoint-northwest, disjoint-southeast, disjoint-northwest, disjoint-southwest) exist between the objects in Figure 3.1, only the entries corresponding to these relations are shown. The entry for disjoint-west, for instance, consists of (a,b,d-w,1,4),(a,c,d-w,5,5), and (c,d,d-w,1,2) which are all the *spatio-temporal relations* that represent the disjoint-west relation between the objects in Figure 3.1.

3.2 Temporal Organization

At the second step of the construction of *SMIST*-index, we construct an interval tree for each topological-directional relation type. This interval tree, which is our proposal, is called *MIS*-tree (**M**aximal **I**ntersecting Interval **S**ets). The main idea of constructing *MIS*-tree is partitioning the time dimension into intervals,

Algorithm *ProduceCommonTimeIntervals*(R)
R is a set of *spatio-temporal relations*

```
1   merge starting and ending frame numbers and place them in list L
2   set maximum of L to max
3   set minimum of L to min
4   for m = min to max do
5       produce an empty maximal set with common time interval [m,m]
6   for each spatio-temporal relation (o_i, o_j, td_k, s, e) ∈ R do
7     for each maximal set M do
8       if [s, e] intersects common time interval of M then
9           add (o_i, o_j, td_k, s, e) to M
10  sort the maximal sets according to their common time intervals
11  set M to the first maximal set
12  while M is not the last maximal set do
13    if M=next maximal set then
14        set ending point of common time interval of M to the ending
          point of common time interval of the next maximal set
15        remove the next maximal set
16    set M to the next maximal set
```

Figure 3.3: The algorithm for production of *maximal sets*

$$[s_0, e_0], [s_1, e_1], ..., [s_n, e_n]$$

which will be called *common time intervals*, and producing a *maximal set* M_i, whose elements are *spatio-temporal relations*, for each common time interval $[s_i, e_i]$ $(0 \leq i \leq n)$ on the basis of the following definition.

Definition 3.2 A *maximal set* M, with *common time interval* $< s_c, e_c >$, is a set of *spatio-temporal relations* satisfying the following conditions:

1. For each *spatio-temporal relation* $d < o_1, o_2, td_i, s, e > \in M$, $s \leq s_c$ and $e \geq e_c$.

2. For each *spatio-temporal relation* $d < o_1, o_2, td_i, s, e >$, $s \leq s_c$ and $e \geq e_c \Longrightarrow d \in M$.

Given the *spatio-temporal relations* $d_1, d_2, ..., d_m$, *common time intervals* $[s_0, e_0]$, $[s_1, e_1]$,... ,$[s_n, e_n]$-and the corresponding *maximal sets* $M_0, M_1, ..., M_n$ are produced using the algorithm given in Figure 3.3.

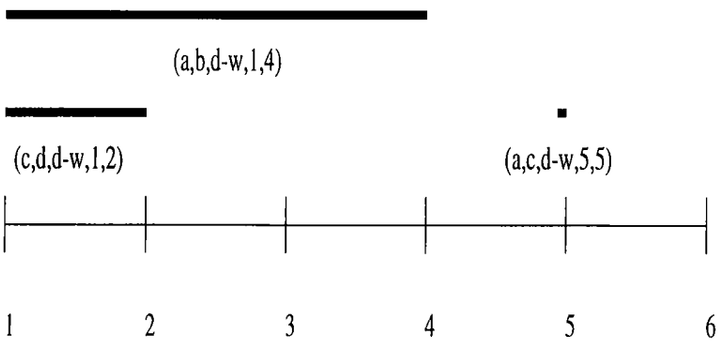


Figure 3.4: Time intervals of *Spatio-temporal relations* for disjoint-west

Figure 3.4 through 3.7 show time intervals of relations for each topological-directional relation type that exists in the example data of Figure 3.1, and Table 3.5 through 3.8 show *maximal sets* produced for the corresponding topological-directional relations. For each pair of inverse topological-directional relations, time intervals and *maximal sets* are given for only one relation of that pair.

Maximal Set	Common Time Interval
M_0 ={(a,b,d-w,1,4),(c,d,d-w,1,2)}	[1,2]
M_1 ={(a,b,d-w,1,4)}	[3,4]
M_2 ={(a,c,d-w,5,5)}	[5,5]

Table 3.5: Maximal sets and corresponding common time intervals for disjoint-west

Theorem 3.1 Let S denote the set of *maximal sets*. *Common time intervals* of all sets are totally ordered; i.e., let n be the number of *maximal sets* and $< s_0, e_0 >, < s_1, e_1 >, ..., < s_n, e_n >$ be the common time intervals of these sets. Then

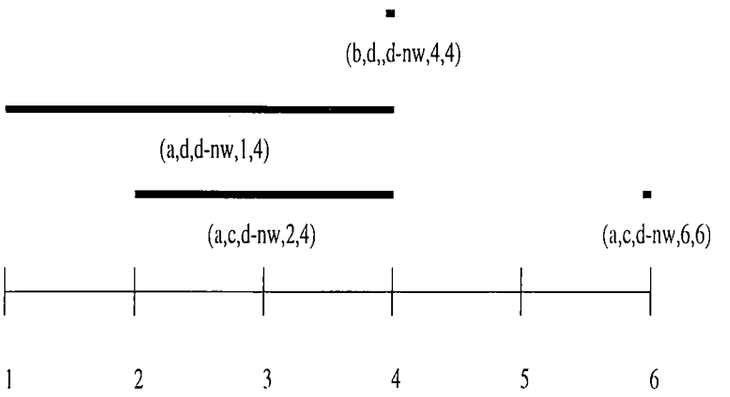


Figure 3.5: Time intervals of *Spatio-temporal relations* for disjoint-north

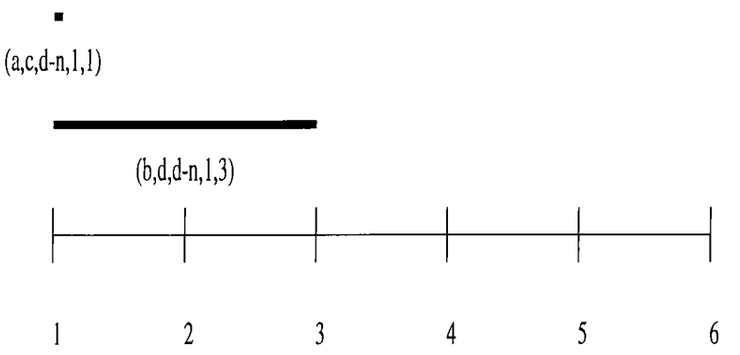


Figure 3.6: Time intervals of *Spatio-temporal relations* for disjoint-northwest

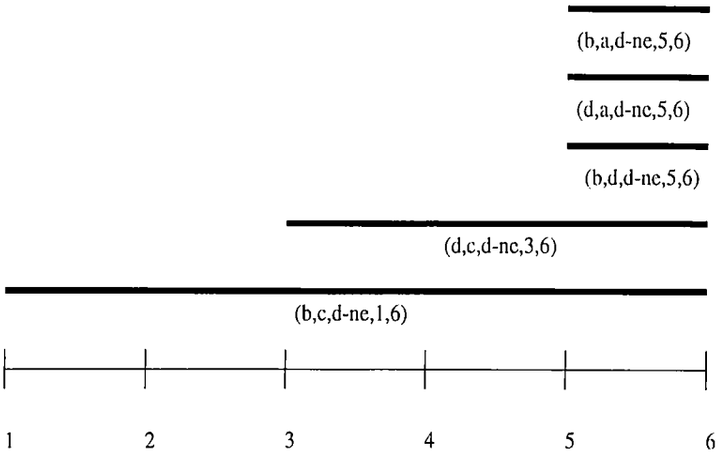


Figure 3.7: Time intervals of *Spatio-temporal relations* for disjoint-northeast

Maximal Set	Common Time Interval
M_1 ={(a,c,d-n,1,1),(b,d,d-n,1,3)}	[1,1]
M_2 ={(b,d,d-n,1,3)}	[2,3]

Table 3.6: Maximal sets and corresponding common time intervals for disjoint-north

$$e_0 < s_1 \wedge e_1 < s_2 \wedge ... \wedge s_{n-1} < e_n$$

holds.

Proof 3.1 Let L be the set of *common time intervals* $[s_l, e_l]$ that has been produced by the algorithm *ProduceCommonTimeIntervals* (lines 1-5). All the intervals in L are totally ordered since for each $[s_l, e_l]$, $s_l = e_{l-1} + 1$ ($1 \leq l \leq n$). After the placement of *spatio-temporal relations* to the appropriate *maximal sets*, all the *maximal sets* are examined in the ascending order of the *common time intervals*. A *common time interval* is either expanded or it remaines unchanged. If the *common time interval* $[s_l, e_l]$ is expanded, then it becomes $[s_l, e_{l+1}]$ and the *maximal set* with the *common time interval* $[s_{l+1}, e_{l+1}]$

Maximal Set	Common Time Interval
M_0 ={(a,d,d-nw,1,4)}	[1,1]
M_1 ={(a,d,d-nw,1,4),(a,c,d-nw,2,4)}	[2,3]
M_2 ={(a,d,d-nw,1,4),(a,c,d-nw,2,4),(b,d,d-nw,4,4)}	[4,4]
M_3 ={(a,c,d-nw,6,6)}	[6,6]

Table 3.7: Maximal sets and corresponding common time intervals for disjoint-northwest

Maximal Set	Common Time Interval
M_0 ={(b,c,d-ne,1,6)}	[1,2]
M_1 ={(b,c,d-ne,1,6),(d,c,d-ne,3,6)}	[3,4]
M_2 ={(b,c,d-ne,1,6),(d,c,d-ne,3,6),(b,a,d-ne,5,6), (d,a,d-ne,5,6),(b,d,d-ne,5,6)}	[5,6]

Table 3.8: Maximal sets and corresponding common time intervals for disjoint-northeast

is deleted. Since $e_{l+1} \leq s_{l+2}$, the total order between the *common time intervals* does not change.

After producing all the *maximal sets*, these sets are stored in a B+-tree according to their common time intervals. Each entry in the *SMIST*-index points to a B+-tree. Since all the common time intervals of the sets are totally ordered, starting or ending frame number can be used as key in organizing B+-trees.

Chapter 4

Query Execution

As we have stated before, queries of concern in this work are the ones that involve motion of multiple objects which is specified by the change in objects' relative spatial positions. Therefore, a query consists of specification of relative positions of objects in a sequence of frames. This kind of a query is different from the one in which the trajectory of each object is explicitly defined. Thus, spatial relationships between objects gain special importance for the queries of our concern. Another important property of such queries is that identities of the objects are not specified. Therefore, to answer a query we have to find a similar spatial structure that is a set of objects whose spatial relations between each other is the same as that in the first frame in the query. Then, in order to decide whether it is actually the answer, consequent frames should be compared with that of the query.

The expensive part of execution of such a query is matching the first frame of the query with some frames in the database. Once a matching frame is found and object identities are specified, checking whether the objects' spatial structure in the following frames is similar to the one defined in the query is straightforward. Therefore, we focus our attention on finding a similar frame for the first frame of the motion query.

Definition 4.1 A *structural query* Q is a set specifying the spatial relations

between n objects $o_1, o_2, ..., o_n$ where each spatial relation td_i is a topological-directional relation:

$$Q = \{(o_1, o_2, td_{i11}), (o_1, o_3, td_{i12}), ..., (o_1, o_n, td_{i1n-1}),$$
$$(o_2, o_1, td_{i21}), (o_2, o_3, td_{i22}), ..., (o_2, o_n, td_{i2n-1}),$$

$$(o_n, o_1, td_{in-1}), (o_n, o_2, td_{in-2}), ..., (o_n, o_{n-1}, td_{inn-1})\}$$

Definition 4.2 Let Q be a *structural query*, td_i be one of the topological-directional relations in Q, and m be the number of relations of type td_i, then the projection of Q on td_i, $PQtd_i$, is specified as:

$$PQtd_i = \{(o_{l1}, o_{j1}, td_i), (o_{l2}, o_{j2}, td_i), ..., (o_{lm}, o_{jm}, td_i)\}.$$

Definition 4.3 Let $d(o_1, o_2, td_i)$ denote a spatial relation between objects o_1 and o_2. o_1 is called the *first object* and o_2 is called the *second object* for spatial relation td_i.

In order to clarify these definitions, we provide the first frame of an example motion query in Figure 4.1 and the application of the above definitions to this query is provided below.

Example 4.1 The *structural query* Q for the frame in Figure 4.1, which contains three objects that are identified as x, y, and z, is

$$Q = \{(y, x, d{-}ne), (z, x, d{-}ne), (x, y, d{-}sw), (z, y, d{-}n), (x, z, d{-}sw), (y, z, d{-}s)\}$$

The projection of the structural query Q on topological-directional relation *disjoint-northeast* is

$$PQ_{d-ne} = \{(y, x, d-ne), (z, x, d-ne)\}$$

The set of *first objects* in PQ_{d-ne} is $\{y, z\}$ and the set of *second objects* in PQ_{d-ne} is $\{x\}$.

After providing the basic definitions and examples, we can now present an

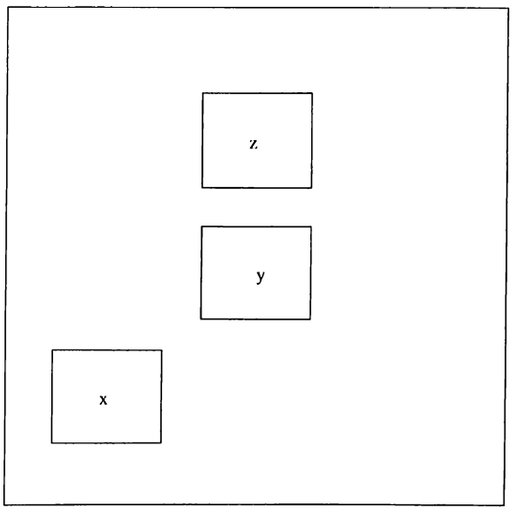


Figure 4.1: An example of the first frame of a structural query

algorithm, which we call *FindIntersectionofMatchSpace*, for query execution. The algorithm uses *SMIST*-index that was described in the previous chapter and is composed of a filter and a refine steps. As the names suggest, the filter step eliminates some of the irrelevant data so that the whole search space shrinks into a much smaller one, and the refine step uses this smaller search space and eliminates the remaining irrelevant data by checking all the spatial relationship constraints between the objects. As a result, exact matches, if they exist, are found. The next section explains the algorithm *FindIntersectionofMatchSpace* in detail.

4.1 An Algorithm for Query Execution

We call the filter step of our query execution algorithm *FindValidIntervals*. The main function of this step is to find intervals in which there exist enough number of relations from each relation type in the query. An interval $[s, e]$ is a valid interval for *structural query* Q, if for each topological-directional relation td_i in Q, the following conditions hold:

1. There exists a *maximal set* M whose common time interval $[s_c, e_c]$ intersects $[s, e]$ and is composed of relations of type td_i.

Algorithm *Match*

```
1   for each valid interval [s, e] do
2       set match set of each object in the query to empty set
3       for each td_i in motion query Q do
4           find the maximal set M, whose common time interval [s_c, e_c]
            intersects [s, e] and is composed of relations of type td_i
5           for each element e = (o_lk, o_jk, td_i) of PQtd_i do
6               if match set for o_lk is empty then
7                   assign S to match set for o_lk
8               else
9                   compose a set S from the first elements in M
10                  assign intersection of S and match set for o_lk to match set
                    for o_lk
11              if match set for o_jk is empty then
12                  assign S to match set for o_jk
13              else
14                  compose a set S from the second elements in M
15                  assign intersection of S and match set for o_jk to match set
                    for o_jk
16          find all possible permutations of objects by using the match set
            for each object
17          check consistency of each permutation
```

Figure 4.2: An algorithm for constructing the match set of each object

2. Cardinality of $M \geq$ cardinality of $PQtd_i$.

3. Number of *first objects* in $M \geq$ number of *first objects* in $PQtd_i$.

4. Number of *second objects* in $M \geq$ number of *second objects* in $PQtd_i$.

The valid intervals found by using the data given in Figure 3.1 for the query shown in Figure 4.1 are [1,1], [2,2], and [3,3].

After collecting such intervals in the refine step, the algorithm constructs a set for each object, which we call the *match set*. These *match sets* are constructed in the following way (Figure 4.2):

For each valid time interval $[s, e]$ and topological-directional relation td_i, a *maximal set* M is found. The values an object can take are decided according to its position in PQ_{td_i}. If the object is one of the *first objects* in PQ_{td_i}, then

its *match set* is intersected with the *first object* set of the *maximal set* M. If the object is one of the *second objects* in PQ_{td_i}, then its *match set* is intersected with the *second object* set of the *maximal set* M. Table 4.2 shows the values each object can take for each valid time interval $[s, e]$ and topological-directional relation td_i.

As the next step, for each valid time interval $[s, e]$, the match set for each object is found by taking the intersection of match sets from each topological-directional relation td_i the object can take. Table 4.3 shows the match set of each object for each valid time interval $[s, e]$.

If at least one object's match set is empty in interval $[s, e]$, then there is no solution in that interval. Table 4.4 shows possible matches. The algorithm checks whether these matches satisfy all the spatial constraints.

Example 4.2 For the example query in Figure 4.1, Table 4.1 gives the *first object* sets and *second object* sets for each of the topological-directional relations d-n and d-ne in Q for each selected valid time interval (i.e., for each of intervals [1,1], [2,2], and [3,3]). In Table 4.2, the set of objects that can be assigned to the objects x, y, and z for spatial relations d-n and d-ne is listed. In Table 4.3, the match set of each object at a particular valid time interval is determined by taking intersection of the sets derived for spatial relations d-n and d-e for that time interval. After each object's match set is determined, permutations are produced. For [1,1] and [2,2], the permutation cannot be produced since the match set of object y is empty for these time intervals. Only one permutation can be produced for the time interval [3,3] and it satisfies all the constraints specified in the query.

$[s,e]$	td_i	Maximal Set	First Obj. Set	Second Obj. Set
[1,1]	$d-ne$	M_0 ={(b,c,d-ne,1,6)},	$\{b\}$	$\{c\}$
	$d-n$	M_0 ={(a,c,d-n,1,1),(b,d,d-n,1,3)}	$\{a,b\}$	$\{c,d\}$
[2,2]	$d-ne$	M_0 ={(b,c,d-ne,1,6)}	$\{b\}$	$\{c\}$
	$d-n$	M_1 ={(b,d,d-n,1,3)}	$\{b\}$	$\{d\}$
[3,3]	$d-ne$	M_1 ={(b,c,d-ne,1,6),(d,c,d-ne,3,6)}	$\{b,d\}$	$\{c\}$
	$d-n$	M_2 ={(b,d,d-n,1,3)}	$\{b\}$	$\{d\}$

Table 4.1: First object and Second object sets for each valid time interval $[s,e]$ and topological directional relation td_i

$[s,e]$	td_i	First Obj. Set	Second Obj. Set	Obj. x	Obj. y	Obj. z
[1,1]	d-ne	$\{b\}$	$\{c\}$	$\{b\}$	$\{b\}$	$\{c\}$
	d-n	$\{a,b\}$	$\{c,d\}$	$\{a,b\}$	$\{c,d\}$	-
[2,2]	d-ne	$\{b\}$	$\{c\}$	$\{b\}$	$\{b\}$	$\{c\}$
	d-n	$\{b\}$	$\{d\}$	$\{b\}$	$\{d\}$	-
[3,3]	d-ne	$\{b,d\}$	$\{c\}$	$\{b,d\}$	$\{b,d\}$	$\{c\}$
	d-n	$\{b\}$	$\{d\}$	$\{b\}$	$\{d\}$	-

Table 4.2: Match sets for each object for each valid time interval $[s,e]$ and topological-directional relation td_i

$[s,e]$	Obj. x	Obj. y	Obj. z
[1,1]	$\{b\}$	{}	$\{c\}$
[2,2]	$\{b\}$	{}	$\{c\}$
[3,3]	$\{b\}$	$\{d\}$	$\{c\}$

Table 4.3: Final matchset for each object for each valid time interval $[s,e]$

$[s,e]$	Matches
[1,1]	no match
[2,2]	no match
[3,3]	$x=b$, $y=d$, $z=c$

Table 4.4: Possible matches for each valid time interval $[s,e]$

Chapter 5

Implementation Details

One of the most important decision criteria for evaluating an index structure is the disk space it consumes. *SMIST*-index is based on the *maximal set* structure. Since the elements of a *maximal set* are *spatio-temporal relations*, the upper bound for the cardinality of a *maximal set* is $O(n^2)$ where n is the number of objects that appear in a frame. Since each *maximal set* will not be of the same size, there is no choice other than using variable length leaf nodes.

If variable length leaf nodes are used, the maximum number of leaf nodes that fit in a page will vary which is a problematic situation from performance point of view. Therefore, in order not to sacrifice the performance, we choose to save some part of the data in a *maximal set* M. The data consists of the following fields: *Common time interval* of M, the number of *first objects* in M, the number of *second objects* in M, and the cardinality of M. In order not to lose object identifications, which exist either as a *first object* or a *second object* in M, we store the objects in a suitable index structure. Before going into implementation details, we give the definition of a key term for this framework.

Definition 5.1 *Relation set*, R_i, of an object o_i is the set of topological-directional relations where for each *spatio-temporal relation* (o_i, o_j, td_k, s, e), $td_k \in R_i$.

Definition 5.2 *First object set*, F_i, for topological-directional relation td_i is $o_0, o_1, \ldots, o_{n-1}$ where for each $o_j \in F_i$ there exists at least one *spatio-temporal*

relation (o_j,o_m,td_i,s,e).

To recall what algorithm *Match* performs mainly, for a query object o_{qi} it finds the relation set, R_{qi}, of o_{qi}. Then, for each topological-directional relation, $td_k \in R_{qi}$, *first object set*, F_k is derived. The match set of object o_{qi} is set to $F_{j0} \cap F_{j1} \cap ... \cap F_{jm}$. We can assign the same match sets to the objects in the query as the ones assigned by algorithm *Match*, by comparing the *relation set* R_i of object o_i in the query with the *relation sets* of the objects in the database. As a result if *relation set*, $R_{o_{d_j}}$, of an object o_{d_j} in the database, is subset of the *relation set*, R_i , of object o_i then o_{d_j} is added to the match set, Ms_i, of o_i.

In order to perform efficient query execution, we produce *relation set* of each object in the database and store the *relation sets* in an RD-tree [12]. We keep a separate RD-tree for each frame.

RD-tree is an index structure for sets and it is a variant of the R-tree. The analogy between R-trees and RD-trees is that, a leaf node in an R-tree is of the form (*data object*, *bounding box*) while a leaf node in an RD-tree is of the form (*base set*, *bounding set*). Bounding box is the smallest n-dimensional rectangle that contains the data object. Bounding set is the smallest super set of the base set. An internal node of an R-tree is of the form (*child pointer*, *bounding box*) while an internal node of an RD-tree is of the form (*child pointer*, *bounding set*). Here, the *bounding box* is the smallest n-dimensional rectangle that contains all the *bounding boxes* of the entries in the child node and *bounding set* is the smallest super set of all the *base sets* in the child node. Figure 5.1 shows an RD-tree, where at most two entries exist in a node, with the base sets:
$S1$=1, 2, 3, 5, 6, 9
$S2$=1, 2, 5
$S3$=0, 5, 6, 9
$S4$=1, 4, 5, 8
$S5$=0, 9
$S6$=3, 5, 6, 7, 8
$S7$=4, 7, 9.

In the leaf nodes, only the *relation sets* are kept. For each *relation set* R_i in a leaf node, there exists a file containing the record of objects whose *relation*

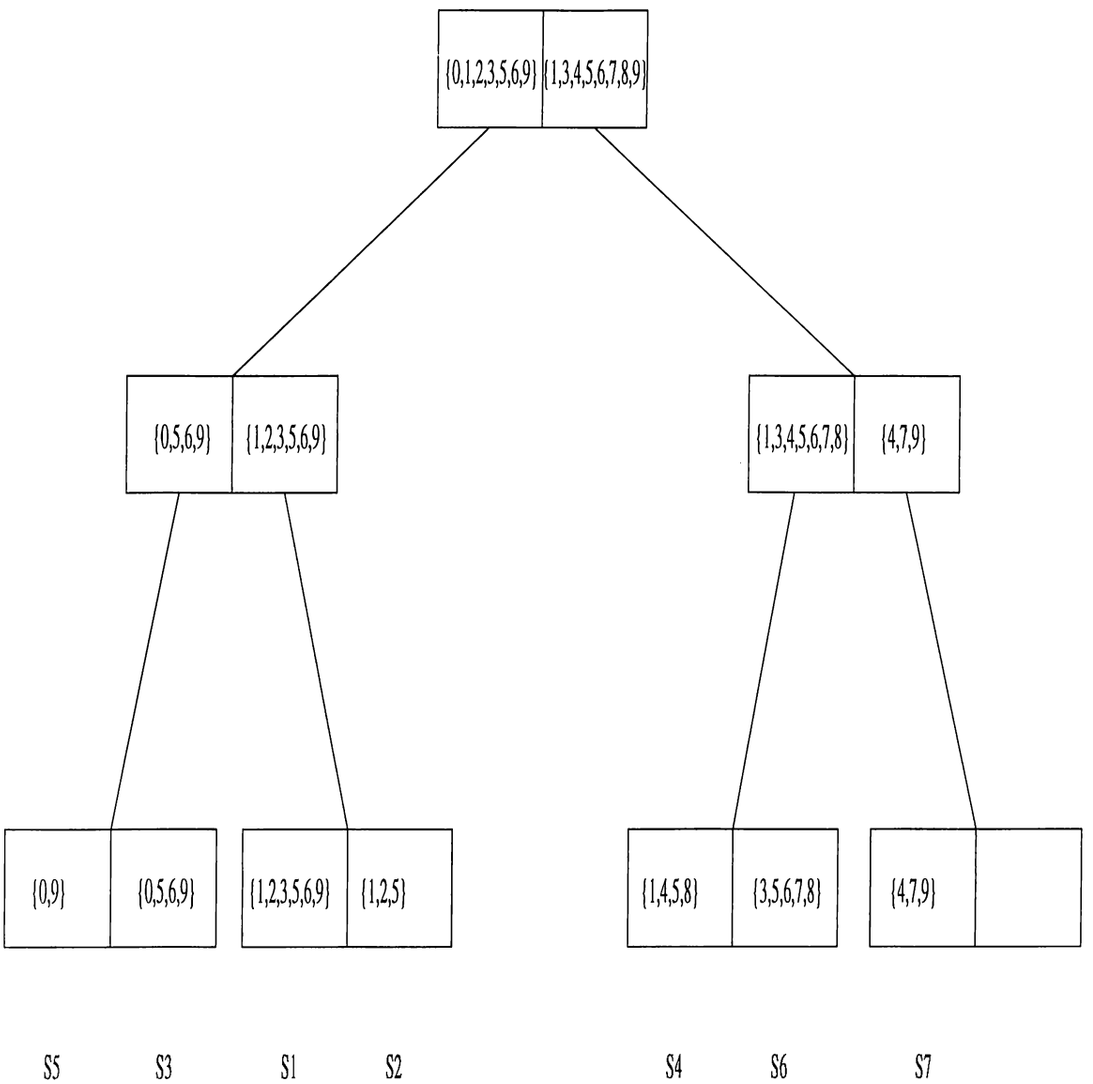


Figure 5.1: A sample RD-tree

sets are subsets of R_i. The record of an object consists of identification number of the object and its spatial positioning in the corresponding frame. Tables 5.1 through 5.3 show the *relation sets* of the objects for the first three frames of the example data given in 3.1.

Object	*Relation Set*
a	{d-w, d-nw, d-n}
b	{d-e, d-n, d-ne}
c	{d-w, d-s, d-sw}
d	{d-e, d-se, d-s}

Table 5.1: *Relation sets* of the objects for the first frame of Figure 3.1

Object	*Relation Set*
a	{d-w, d-nw}
b	{d-e, d-n, d-ne}
c	{d-w, d-se, d-sw}
d	{d-e, d-se, d-s}

Table 5.2: *Relation sets* of the objects for the second frame of Figure 3.1

Object	*Relation Set*
a	{d-w, d-nw}
b	{d-e, d-n, d-ne}
c	{d-se, d-sw}
d	{d-s, d-se, d-ne}

Table 5.3: *Relation sets* of the objects for the third frame of Figure 3.1

Figure 5.2 presents the algorithm *MatchRelationSet* which is analog of the algorithm *Match* given in Figure 4.2. Table 5.4 shows the *relation sets* of objects in the query given in Figure 4.1. In Table 5.5, match sets of objects, which are found by applying the algorithm *MatchRelationSet*, are shown for each of the valid time intervals [1,1], [2,2], and [3,3]. Note that Table 5.5, which is constructed by comparing the *relation sets* of the objects, is the same as Table 4.3, which is constructed by using the original *SMIST*-index.

Algorithm *MatchRelationSet*

```
1   for each valid interval [s, e] do
2       set match set of each object in the query to empty set
3   for each o_i ∈ Q do
4       search the index I for relation set, R_i, of o_i
5       add all the relation sets to R
6       for each S ∈ R do
7           find the corresponding file F
8           add all the objects in F to the match set of o_i, M_{o_i}
9   find all possible permutations of objects by using match set for each object
10  check consistency of the permutations
```

Figure 5.2: An algorithm for constructing the match set of each object by using the *relation set* of the objects

Object	*Relation Set*
x	{d-sw}
y	{d-s, d-ne}
z	{d-n, d-ne}

Table 5.4: *Relation sets* of the objects of the query given in Figure 4.1

$[s, e]$	Obj. x	Obj. y	Obj. z
[1,1]	$\{b\}$	{}	$\{c\}$
[2,2]	$\{b\}$	{}	$\{c\}$
[3,3]	$\{b\}$	$\{d\}$	$\{c\}$

Table 5.5: Matchset for each object for each valid time interval $[s, e]$

Chapter 6

Experiments

In order to evaluate the performance of the *SMIST*-index structure combined with the algorithms that we proposed with the framework in [18], and to evaluate the properties of *SMIST*-index, i.e. to see whether it is scalable to the number of frames and number of objects, we conducted a number of experiments.

Before going into details of the specific experiments, we want to give some information about the experimental setup valid for all the experiments we conducted. In our data, all the frames have equal number of objects. Objects are specified by their minimum bounding rectangles. Coordinates of the objects were produced randomly. Page size was set to 512 bytes and size of the buffer was set to 256K bytes. Least Recently Used (LRU) page replacement policy was used. The programs were implemented in C programming language and all the experiments were run on a SUN SPARC workstation running the Solaris operating system.

In the first experiment, we compared *SMIST*-index combined with the *Match RelationSet* algorithm against the *Join Window Reduction* algorithm (see Figure 2.1) combined with R*-trees. We conducted the experiment for varying number of objects (from 10 to 100 in steps of 10) with 100 frames. The number of objects in the query was set to 10. The performance metric we used in the experiments is the number of disk accesses. Experimental results provided

in Figure 6.1 and Table 6.1 show that as the number of objects increases the number of disk accesses performed by the Join Window Reduction algorithm increases faster than that of the *MatchRelationSet* algorithm. In order to explain the reason for that result, we would like to give a formula for the number of disk accesses performed by our proposed model.

$$\#of Diskaccess_{SMIST} = Diskaccess_{FS} + \sum_{i=1}^{\#of validintervals} Diskaccess_{RS_i} \quad (1)$$

Here, *FS* and *RS* denote filter and refine steps, respectively. $Diskaccess_{FS}$ represents the number of disk accesses performed during analysis of the *MIS*-tree, which is a B+-tree storing the common time intervals of the maximal sets produced for a particular topological-directional relation. Since common time intervals of the maximal sets for a particular relation type are totally ordered according to *Theorem* 3.1, the maximum value the number of maximal sets can take is the number of frames[1]. In determining the valid time intervals in the filter step, *MIS*-trees of the relation types that exist in the query are analyzed. At the beginning, two *MIS*-trees are joined to find the intersecting time intervals. These intersecting time intervals are used to search the *MIS*-trees of the other remaining relations. Therefore, the number of disk accesses performed in the filter step depends on the number of frames, the number of relation types that exist in the query, and the length of the intervals during which the spatial relation between two objects remains the same. In our experiment, we only increased the number of objects in the database. The number of frames remained constant. Since we posed the same query (with the same number of objects and the same topology) each time, the factor that changed the number of disk accesses performed in the filter step is the length of the interval of the *spatio-temporal* relations which depends on the frequency of change in the topology of objects. In other words, it depends on how long the *spatio-temporal relation* between a couple of objects remains the same. Therefore, topology of objects is a stronger factor than the number of frames in determination of the number of entries in *MIS*-tree. For instance, it is possible that even the number of frames increases, the number of entries in the *MIS*-trees may decrease. Table 6.2 shows the number of disk accesses performed in the filter step.

[1] It is the case when each common time interval is of length 1.

Disk accesses performed in the refine step ($Diskaccess_{RS}$) is composed of two parts: accesses for finding the supersets of the *relation sets* of the objects in the query by searching the RD-tree and accesses for reading the files corresponding to the found *relation sets*. The maximum number of entries in the RD-tree can be the number of objects in the database; and the maximum number of searches done on the RD-tree is equal to the number of objects in the query. To optimize the number of files read, *relation sets* which are subsets of other *relation sets* are not considered. The number of files read is $O(n_d)$ where n_d is the number of objects in the database. Table 6.3 shows the average number of files read for each valid time interval. As it can be observed, the average values are much smaller than the number of objects in the database and sometimes it may decrease as the number of objects in the database increases. This is because the number of files read, which is equal to the number of *relation sets* that are not subset of any other *relation set* in the search result obtained from the RD-tree, mainly depends on the topology of the objects in each frame. Topology of the objects determines the number of different *relation sets* and the subset-superset relations between the *relation sets* hence the number of entries in the RD-tree. The number of files read to obtain the coordinates of the objects is determined by the number of entries in the RD-tree and the topology of objects in the query.

Since the refine step is executed once for each valid time interval, the number of valid intervals also affects the total number of disk accesses. As the number of objects in the database increases the number of valid intervals also increases. Therefore, the most important factor that leads to the increase in the total number of disk accesses is the increase in the number of valid time intervals. This is the natural consequence of the increasing probability of the existence of time intervals which satisfy the constraints[2] used in deciding whether a frame interval is valid or not.

On the other hand, the number of disk accesses performed by *JWR* algorithm is

$$\#ofDiskaccesses_{JWR} = Numberofframes \times Diskaccess_{match} \quad (2)$$

[2]Number of *first objects*, number of *second objects*, and cardinality of set of *spatio-temporal relations* of objects for a particular relation type in the query.

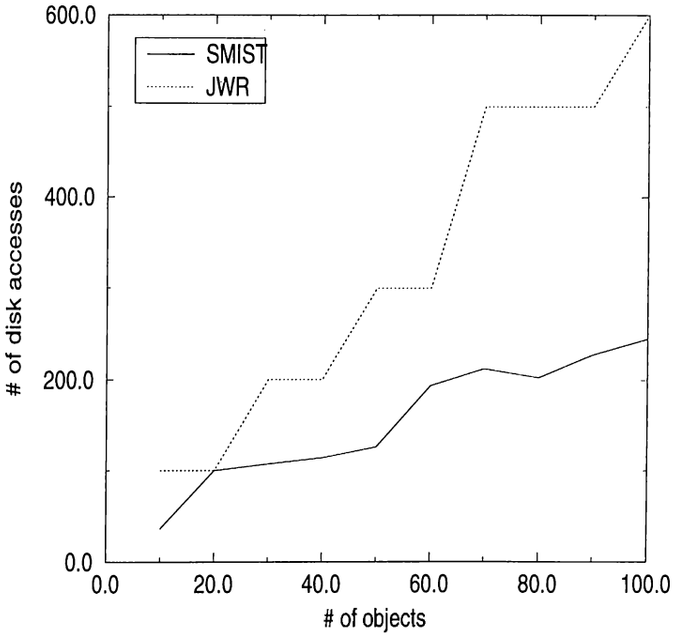


Figure 6.1: Disk accesses as a function of number of objects in the database

since each frame is analyzed separately. During the match operation, an R*-tree is used. The R*-tree has entries as many as the number of objects in the database, and the number of disk accesses during the match is $O(n_d)$. Therefore, an increase in the number of objects in the database sometimes causes a sharp increase in the number of disk accesses. We use the word "sometimes" because as it can be observed from Figure 6.1, sometimes the number of disk accesses remains the same even the number of objects in the database increases. This is because, the number of disk accesses changes only when the number of pages used by the R*-tree changes, and the number of pages used depends on the page size. Comparison of the two methods in the case of taking CPU time as the performance metric yields results parallel to the ones obtained in this experiment.

In the second experiment, we evaluated the *MatchRelationSet* algorithm for varying number of frames. The evaluation was done for frame numbers=1000, 2000, 3000, 4000, and 5000. The number of objects in each frame was set to 50 and the number of objects in the query was set to 10. According to the results

Number of Objects	*SMIST*-index	*JWR*
10	32	100
20	89	100
30	93	200
40	101	200
50	114	300
60	174	300
70	200	500
80	189	500
90	218	500
100	233	600

Table 6.1: Number of disk accesses performed with *SMIST*-index and *JWR* algorithm. Number of objects in the query is 10.

given in Figure 6.2 and Table 6.5, our model scales well with the increasing number of frames. According to the formula (1) given above, an increase in the number of frames affects both the number of disk accesses performed in the filter step and the number of valid intervals detected. Since both the number of objects in the database and the number of objects in the query are kept constant, the number of disk accesses performed for a single frame in the refine step does not change. We also analyzed the increase in the number of disk accesses performed in the filter step, and the increase in the number of valid time intervals detected. The results are presented in Tables 6.6 and 6.7.

Since the number of frames is not the main factor determining the size of the *MIS*-trees but the length of the intervals of the *spatio-temporal relations* is, the number of disk accesses performed in the filter step does not increase sharply as the number of frames increases.

For the number of valid time intervals, the rate of increase as a function of the number of frames is approximately constant. As it can be seen from Table 6.7, for each number of frames, about 20% of frames are analyzed.

For increasing number of frames *SMIST*-index performs less number of disk accesses than *JWR* algorithm.

In the third experiment, we calculated *Precision* for the results of the filter

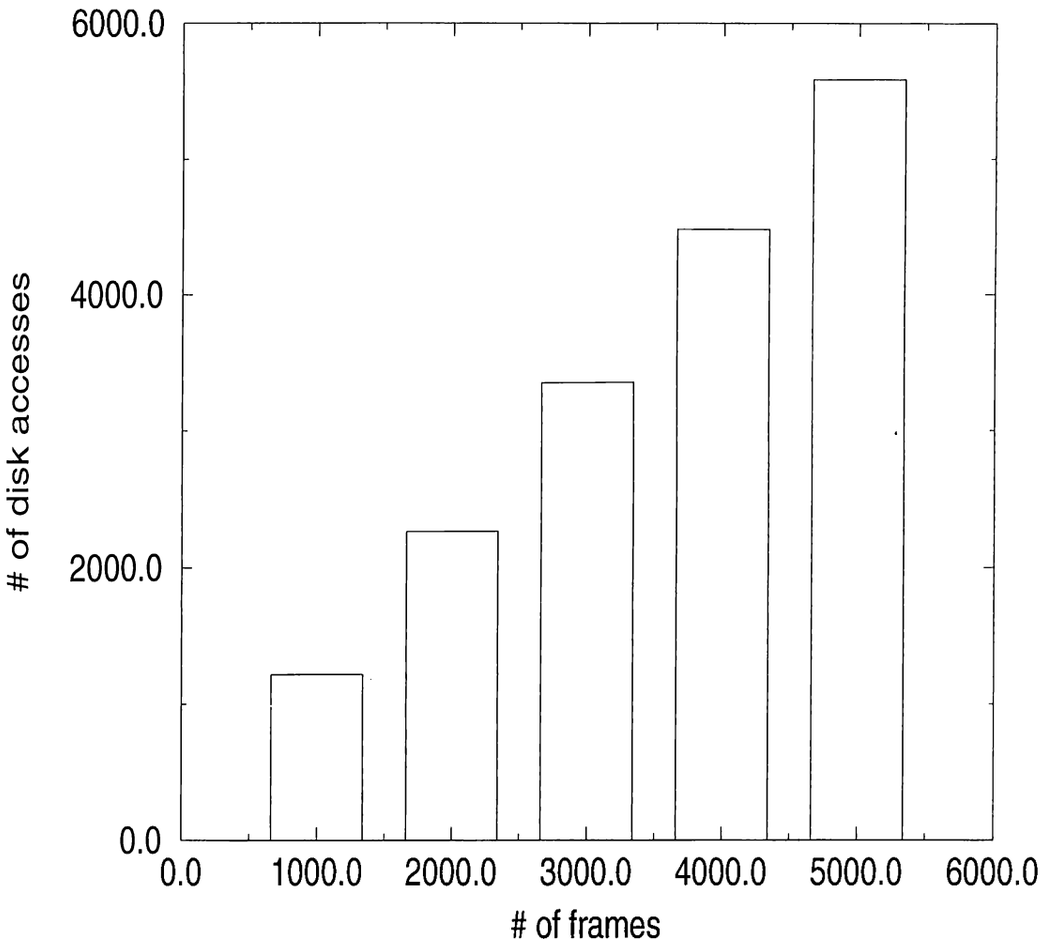


Figure 6.2: Disk accesses with *SMIST*-index as a function of the number of frames.

Number of Objects	Disk accesses in FS
10	4
20	11
30	13
40	13
50	15
60	18
70	18
80	21
90	22
100	23

Table 6.2: Number of disk accesses performed in the filter step.

step. That is, we found out the percentage of the intervals in which exact solution exists out of the valid intervals detected. We conducted the experiment with 10 objects in the database and 100 frames. We posed queries with varying number of objects (3 to 10). The results obtained are shown in Figure 6.3. As the number of objects in the query increased, *Precision* tended to decrease. This may stem from the fact that as the number of the objects in the query increases, the number of the constraints also increases. The filter step analyzes only the number of *first objects*, number of *second objects*, and cardinality of the set of the *spatio-temporal* relations for each topological-directional relation type in the query. This analysis is useful for checking whether a frame satisfies the minimums for the indicated information; but in this analysis the constraints between couple of objects are not checked. Therefore, the increase in the number of objects in the query increases the retrieval of false frames.

We also computed the number of valid intervals detected by the filter step for the same data used in obtaining the results provided in Figure 6.3. According to Table 6.8, on the average 17% of the frames are considered in deciding the exact matches in the refine step, which means that approximately 83% of the frames are not considered. Considering the indicated value, the functionality of *SMIST*-index in reducing the search space is promising.

Finally, we found out the number of matches done by our proposed model and compared it with the maximum number of possible matches that can be

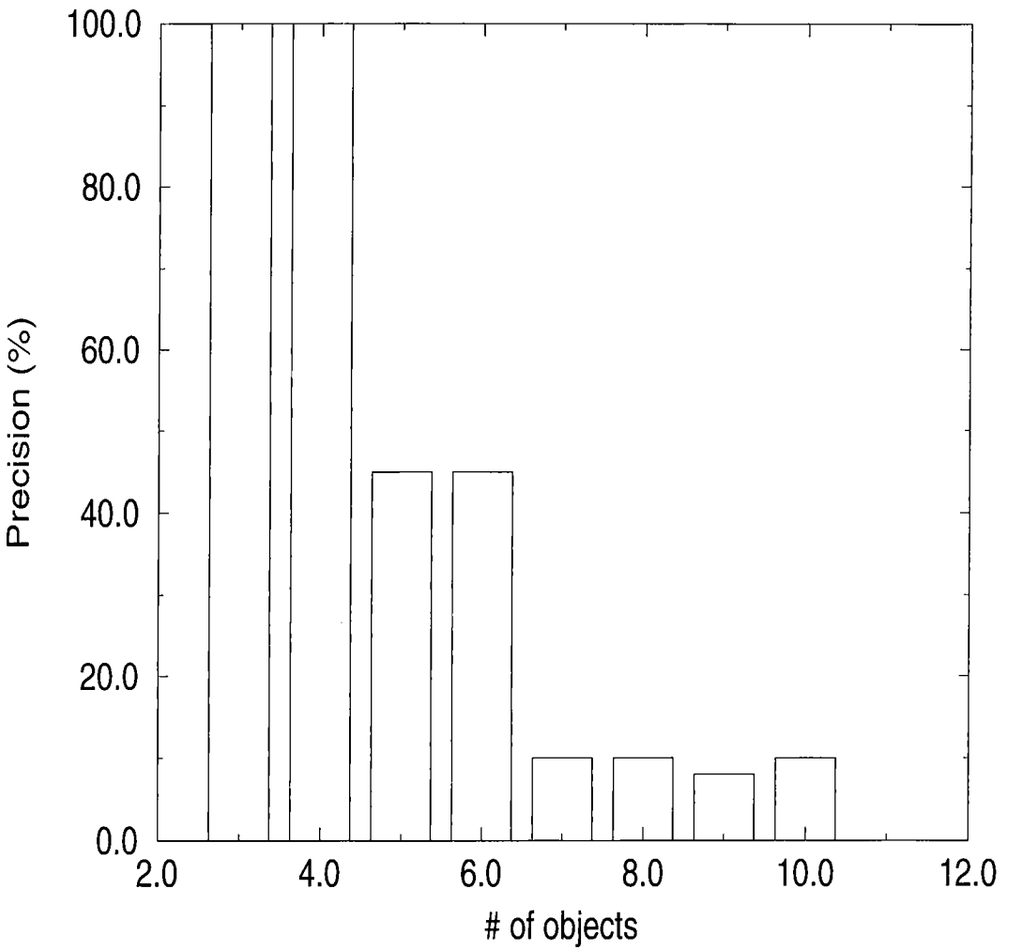


Figure 6.3: *Precision* for the filter step

Number of Objects	Number of Files Read
10	3
20	5
30	3
40	3
50	3
60	5
70	6
80	5
90	6
100	6

Table 6.3: Average number of files read for each valid time interval.

done. Table 6.9 lists those numbers for queries with varying object numbers (3 to 10). Data consists of 100 frames with 10 objects in each frame. Number of answers for each query is also provided in the same table. As it can be seen from the table, there is a huge gap between the actual number of matches performed by using *SMIST*-index and the maximum number of possible matches that can be done. Therefore, this is another evidence for the success of *SMIST*-index in reducing the search space.

Consequently, experimental results combined with the analysis of the properties of the *SMIST*-index imply that *SMIST*-index is scalable both to increasing number of frames and objects for the values we used in our experiments. The main reason for its scalability to increasing number of frames and number of objects is that topology of the objects in the database is more effective than the number of frames and the number of objects in determination of the values of important parameters such as the number of entries in *MIS*-trees and the RD-trees.

Number of Objects	Number of Valid Time Intervals
10	7
20	13
30	20
40	22
50	24
60	26
70	26
80	27
90	28
100	30

Table 6.4: Number of valid time intervals found in during the filter step.

Number of Frames	Number of disk accesses
1000	1222
2000	2266
3000	3360
4000	4480
5000	5587

Table 6.5: Number of disk accesses performed with *SMIST*-index for varying number of frames. Number of objects in the database is 50 and number of objects in the query is 10.

Number of Frames	Number of disk accesses in filter step
1000	112
2000	206
3000	345
4000	440
5000	532

Table 6.6: Number of disk accesses performed in the filter step for varying number of frames. Number of objects in the database is 50 and number of objects in the query is 10.

Number of Frames	Number of Valid Time Intervals
1000	222
2000	412
3000	603
4000	808
5000	1013

Table 6.7: Number of valid time intervals detected for varying number of frames. Number of objects in the database is 50 and number of objects in the query is 10.

Number of Objects in Query	Number of Valid Intervals
3	20
4	20
5	20
6	20
7	20
8	20
9	12
10	5

Table 6.8: Number of valid intervals found in the filter step. Number of frames=100 and number of objects=10.

Number of Objects in Query	Number of Matches	Number of Possible Matches	Number of Answers
3	1935	72000	372
4	2612	504000	87
5	2678	3024000	15
6	2704	1512000	19
7	2709	60480000	3
8	1617	1995840000	3
9	245	3991680000	1
10	94	3991680000	1

Table 6.9: Number of matches performed compared with possible number of matches for queries with varying number of objects. Number of frames=100 and number of objects=10.

Chapter 7

Conclusion and Future Work

In this thesis, we have proposed a new spatio-temporal index structure, which we call *SMIST-index*. The motivation stemmed from providing an efficient execution plan for one type of motion queries, in which objects' spatial relations are specified for a sequence of frames. Motion information is obtained from the change in objects' spatial positions relative to each others. Assuming that object identification is not provided in the query specification, for a video with N objects in each frame and a motion query with n objects there are $N!/(N-n)!$ possible matches for the objects in the query. As N gets larger compared to n, the number of possible matches becomes exponential to the number of objects in the query. Therefore, index structures whose use will decrease the number of object matches is of great importance as they will make it possible to answer such queries in a reasonable amount of time.

We have shown through experimental results that *SMIST*-index reduces the search space considerably since it initially performs a filtering operation on the time dimension in order to find frames which probably include an answer to the query. In order to find out such frames, some of the spatial constraints between the objects in the query are compared against the constraints stored in the database. Following the filtering, a refine step further analyzes the filtered frames to find exact matches.

We have compared *SMIST*-index with the *Join Window Reduction* algorithm combined with R*-trees (see [18]). In the Join Window Reduction algorithm, there is no reduction of the search space on the time dimension. As a result, the frames of a video have to be analyzed separately. The experimental results lead to the observation that *SMIST*-index clearly outperforms the *Join Window Reduction* algorithm. The lack of temporality in the index structure that is used in their framework is the primary reason for the performance gap between our index structure and theirs.

We have also tested scalability of our index structure to increasing number of frames and increasing number of objects. The experimental results show that *SMIST*-index scales well as the number of frames or the number of objects increases. The main reason for its scalability is that the topology of the objects in the database is more effective than the number of frames in determining the number of entries in the *MIS*-tree. That is, the increase in the number of frames is not sufficient to increase the number of entries in the *MIS*-tree, which determines the number of disk accesses performed in the filter step. Similarly, the topology of objects in the query is more effective than the number of objects in the database and the query in determining the number of files read to obtain the coordinates and identifications of the objects. Only the number of valid time intervals is directly related with the number of objects in the database and the number of frames. However, the number of valid intervals detected is approximately 20% of the total number of frames and we believe that this is a promising reduction in the search space.

For additional future work, we would like to explore the flexibility of *SMIST*-index for its adaptability to applications that require frequent changes in the relative positions of objects. For instance, in multimedia authoring systems data is more open to such changes and we would like to find out the time complexity of reflecting such changes to *MIS*-trees. We would also like to investigate whether the execution of queries in which spatial constraints between the objects are given in a disjunctive form[1] employs reuse of the information or not, and if so to what degree it does.

[1] E.g., object a is to the north or north-east of object b, object a is to the north-west of object c, and object b is to the west of object c.

Bibliography

[1] Ahanger G., Benson D., and Little T.D.C. "Video Query Formulation". In Proceedings of *Storage and Retrieval for Images and Video databases II, IS T/SPIE Symposium on Electronic Imaging Science & Technology*, San Jose, CA, February 1995.

[2] Akutsu A and Tonomura Y. "Video Tomography: An Efficient Methos for Camera-work Extraction and Motion Analysis". In Proceedings of *ACM Multimedia'94*, pages 349-356, San Fransisco, CA, 1994.

[3] Allen J. "Maintaining Knowledge about Temporal Intervals". In Proceedings of *CACM*, volume 26, 1983.

[4] Arman F., Hsu A., and Chiu M-Y. "Image processing on encoded video sequences". *Multimedia Systems Journal*, 1:211-219, 1994.

[5] Bacchus F. and Grove A. "On the Forward Checking Algorithm". In Proceedings of *Principles and Practice of Constraint Programming*, 1995.

[6] Bacchus F. and Run V. "Dynamic Variable Ordering in CSPs". In Proceedings of *1st International Conference on Principles and Practice of Constraint Programming*, 1995.

[7] Bouthemy P. and Francois E. "Motion Segmentation and Qualitative Dynamic Scene Analysis from an Image Sequence". *Int. Journal of Computer Vision*, 10(2):157-182, 1993.

[8] Chang S., Chen W., Meng H.J., Sundaram H., and Zhong D. "VideoQ: An Automated Content Based Video Search System Using Visual Cues". Technical report, Columbia University, 1996.

[9] Delis V., Papadias D., and Mamoulis N. "Assessing Multimedia Similarity: A Framework for Structure and Motion". In Proceedings of *ACM Multimedia'98*, Bristol, UK, 1998.

[10] Dimitrova N. and Golshani F. *Handbook of Multimedia Information Management*, chapter Video and Image Content Representation and Retrieval, pages 95-138. Prentice Hall, 1997.

[11] Egenhofer M. and Franzosa R. "Point-set topological spatial relations". *Int. Journal of Geographical Information Systems*, 5(2):161-174, 1991.

[12] Hellerstein M. J. and Pfeffer A. "The RD-tree: An Index Structure for Sets". Technical Report 1252, University of Wisconsin Computer Sciences Department, November 1994.

[13] Jagadish H.V. *Handbook of Multimedia Information Management*, chapter Content-Based Indexing and Retrieval, pages 69-93. Prentice Hall, 1997.

[14] J. Z. Li, M. T. Ozsu, and D. Szafron. "Modeling of Moving Objects in a Video Database". In Proceedings of *IEEE International Conference on Multimedia Computing and Systems*, pages 336-343, June, 1997.

[15] Maeda J. "Method for Extracting Camera operations to Describe Sub-Scenes". In Proceedings of *Storage and Retrieval for Images and Video Databases II, IS & T/SPIE Symposium on Electronic Imaging Science & Technology*, San Jose, CA, February 1994.

[16] Nadel B. "Constraint Satisfaction Algorithms". *Computational Intelligence*, pages 188-224, 1989.

[17] Nagasaka A. and Tanaka Y. *Visual Database Systems II*, chapter Automatic Video Indexing and Full Video Search for Object Appearances, pages 113-128. Prentice Hall, 1992.

[18] Papadias D., Mamoulis N., and Delis V. "Algorithms for Querying by Spatial Structure". In Proceedings of *the 24th VLDB Conference*, New York, USA, 1998.

[19] Papadias D., Theodoridis Y., Sellis T. and Egenhofer M. J. "Topological relations in the world of minimum bounding rectangles: A study with

R-trees". In Proceedings of *ACM SIGMOD International Conference on Management of Data*, pages 92-103, San Jose, CA, May, 1995.

[20] Perez-Lopez K. and Sood A. "Comparison of Subband Features for Automatic Indexing of Scientific Image Database". In Proceedings of *Storage and Retrieval for Images and Video Databases II, IS & T/SPIE Symposium on Electronic Imageing Science and Technology*, San Jose, CA, February 1994.

[21] Zhang H.J., Kankanhalli A., and Smoliar S.W. "Automatic Partitioning of Full-Motion Video". *Multimedia Systems*, 1:10-28,1993.