AIRSTD: AN APPROACH FOR INDEXING AND RETRIEVING SPATIO-TEMPORAL DATA

by

Hatem F. Halaoui

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2006

UMI Number: 3204946

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

This manuscript has been read and accepted for Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy

	Professor Sean Ahearn
Date	Chair of Examining Committee
	Professor Theodore Brown
Date	Executive Officer
	Professor Sean Ahearn (Advisor)
	Professor Keith Harrow
	Professor Ioannis Stamos
	Professor E. Lynn Usery
	Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis advisor Sean Ahearn and the committee members Keith Harrow, Ioannis Stamos, and E. Lynn Usery. Their assistance and guidance was a great help to me and without it I would not been able to complete my dissertation.

I would like to thank Professor Theodore Brown for his help while studying at the Graduate Center. Moreover, I would like to add a great thank to Mr. Joe Driscoll.

I cannot forget my parents who provided me with all possible resources to help me in my studies, and my great wife and lovely son for being patient and helpful all these years.

Finally, the first and last thanks go always to God that always gives the strength, the power, and the patience to continue and believe in myself.

Abstract

AIRSTD: AN APPROACH FOR INDEXING AND RETRIEVING SPATIO-TEMPORAL DATA

by

Hatem Halaoui

Advisor: Sean Ahearn

Geographical information about the real world changes rapidly with time. We can see examples of these changes when we look at any city or area. New buildings are built, new roads and highways are constructed, and many other new constructions are added or updated. Moreover, there are changes to the geometries of physical (i.e., buildings) and abstract (i.e., land parcels) entities that need to be recorded. Another example of spatial information that is changing with time is moving objects like airplanes, cars or even moving people. The databases that hold these types of information should always be updated to maintain a record of the old information, which could be needed for analysis or studies by specialists. Databases that have geographical data or spatial data which is changing with time are called spatio-temporal databases. Most spatio-temporal databases

fall under three main categories of applications: (1) applications dealing with moving objects where continuous data is needed, (2) applications dealing with objects that have certain geometries and are located in space and (3) applications that combine the other two.

Spatio-temporal databases need to store information about space and the dynamic change of objects over time. We need to store these changes while keeping the old information in the database. Dealing with this issue increased the difficulty of finding efficient models that handle the time and space complexities of spatio-temporal databases.

The main goal our study is to find an efficient way to deal with spatio-temporal data, including the ability to store, retrieve, update, and query. We offer an Approach for Indexing and Retrieving Spatio-Temporal Data (AIRSTD). We concentrate on two main objectives:

1. Provide an efficient algorithm to retrieve spatio-temporal data at any time T by designing and using two indexes: (1) a temporal index using a BS- tree structure and (2) a spatial index using an R-Tree structure.
2. Provide an efficient algorithm to build a snapshot at any time T using the two indexes (temporal and spatial).

In order to achieve these goals we do the following:

- Build a spatio-temporal database model, associated indexing scheme, and the data structures used in the proposed algorithms.
- Propose a temporal indexing structure for clustering the spatio-temporal data according to their temporal and spatial attributes.
- Use a spatial indexing structure for indexing current information according to the spatial information.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Most spatial information in the real world is also temporal information. In other words, we rarely find geographical information that does not change. Many examples can be given: new buildings are built, new roads and highways are constructed, and other new constructions are added or updated. Moreover, there changes to the geometries of physical (i.e., buildings) and abstract (i.e., land parcels) entities that need to be recorded. In most cases people want their systems to keep the old and the current information and sometimes make estimates of the future information. Adding a temporal dimension to the spatial database increases the complexity of the managing information.

These issues lead to the following questions:

1- How can we save all the information both past and current?

2- Do we save snapshots of the information where a snapshot has full information?

3- What is the interval of validity of each snapshot?

4- What happens if a change occurs during this interval? Do we discard it and hence lose it?

5- Can we take a snapshot at each change? Is this efficient?

Analyzing these questions will lead to the following main points:

What is the most efficient way to save all the information at all times with no loss, taking into consideration these three issues:

- Having uniform intervals for efficient searching.
- Changes within intervals are recorded for consistency.
- Non-changing information is not repeated at all snapshots for space efficiency.

We can summarize these considerations with a simple question:

What is the most efficient way to store, retrieve, and query spatio-temporal data?

We will attempt to answer these questions in the coming chapters. We start by building a spatio-temporal model. Later we introduce our approach for indexing spatio-temporal data using a combination of spatial and temporal indexing structures. Finally, we present two algorithms: (1) finding a spatial object at time T and (2) building a snapshot at time T. We also present an analysis of the algorithms and the proposed structures.

In this chapter we introduce spatio-temporal databases briefly, and we go into more detail in later chapters.

1.1 Spatio-Temporal Databases

Most spatio-temporal databases fall under three main categories of applications:

- Applications dealing with moving objects where continuous data is needed.
- Applications dealing with objects that have certain geometries and are located in space.
- Applications dealing with a combination of the above two.

We introduce spatial databases, temporal databases and spatio-temporal databases in the next few subsections.

1.1.1 Spatial Databases

Databases dealing with geometries, locations, or any kind of space or objects in space, are called geodatabases or spatial databases. There is a huge demand for spatial databases in order to manage and manipulate geographical information. Examples of spatial information that must be managed are land-ownership, place-locators, environmental and agricultural information of lands and territories, tracking moving airplanes or objects, road networks, phone cable networks, and so on. The demand for spatial databases resulted work in modeling and representing spatial databases, as well as extensions of existing query languages in order to process and manipulate such data. There are two main kinds of spatial data that need to be represented:

(1) Stationary entities: This includes physical entities like roads, areas, land, boundaries in maps, and others. These objects might also have non-spatial attributes.

(2) Mobile entities: Some non-spatial objects need spatial attributes to identify their position, like moving cars, airplanes, and similar things.

In order to handle the complex nature of spatial databases, different models and query languages were proposed in this field. ROSE Algebra [21, 23] is one of the best-known proposed models for spatial databases. The approach, which is called "Realm-based approach," introduces spatial data types with spatial predicates and operations that can be applied to these predicates. The approach discusses how continuous data can be discretized, so that predicates and operations can be applied. Another approach, by the same authors, called "The Dual Grid approach" [18], is considered as an extension or update of the Realm-Based Approach [21]. The Dual Grid Approach overcomes some of the disadvantages and loss of precision that could happen when applying operations in the Realm-Based approach. Chapter 2 will have a detailed discussion of spatial databases.

1.1.2 Temporal Databases

Temporal databases represent databases that are changing with time. This leads to adding a time dimension to the database. Research has been done in the area of modeling temporal databases, and many different approaches have been proposed; the relational algebra and calculus were extended (as well as query languages) in order to handle such databases. Proposed approaches to model temporal databases fall under two main categories:

1. First Normal Form approach (1NF): This requires a fixed length attribute. In this case time instants or intervals represent time stamps.
2. Non First Normal Form approach (N1NF): A single tuple tries to capture all the history of the object.

First Normal Form approaches are mostly homogeneous, where time is added as an attribute in the relational model. On the other hand, there are two main forms to model temporal data with N1NF relations. These are: (1) homogeneous data modeling where the temporal domain of all attributes in a tuple is the same, or (2) heterogeneous data modeling where different attributes in a single tuple could have different temporal domains. Details of temporal database modeling are discussed in Chapter 2.

1.1.3 Spatio-Temporal Databases

Spatio-Temporal databases can be seen as a combination of spatial and temporal databases-that is, a way to represent the changes of spatial information over time. As mentioned before, there are three main kinds of applications that deal with Spatio-Temporal databases.

- Applications dealing with moving objects.
- Applications dealing with objects that have some geometry and are located in space.
- A third kind of spatial database that is a combination of the above two.

Designing and modeling spatio-temporal databases can also be seen as extensions of spatial and temporal database models. We present an example of the spatio-temporal modeling approach in Chapter 2.

1.2 Spatial Indexing and Temporal Indexing

Indexing databases is crucial for most databases that contain large amounts of data. The main idea behind indexing is to cluster data according to some criteria that makes manipulation and querying much faster. These criteria are chosen according to the nature of the database and the kind of queries that will be applied to the database. In our case, we need two kind of indexing for spatio-temporal databases. The first kind is spatial indexing, where data is clustered according to its spatial information; the second kind is temporal indexing, where data is clustered according to its time validity. We briefly explain each kind of indexing in this section and give details in later chapters.

1.2.1 Spatial Indexing

In the case of spatial indexing, data is clustered according to its spatial information. There are many famous indexes which can be used, such as R-tree, Quad-tree, Grid file, and so on. Most of these indexes cluster the data using criteria that check the spatial information. For example the R-tree index is a tree of MBR's (minimum bounding rectangles) where each rectangle is like a window that contains one or more spatial objects that are close to each other. Chapter 3 contains details about some spatial index structures.

1.2.2 Temporal Indexing

Temporal indexing is clustering data according to its temporal information. The temporal information is usually a time or an interval. One of the criteria is to cluster each group of objects (data), where the time validity of all the objects in the group satisfies a specified period of time. A temporal index is usually used when the user queries uses time to search for objects. The DBMS goes to the group of data that satisfy the time in the query and then searches for the queried objects. Details of temporal indexing are covered in Chapter 3 as well.

1.3 Organization of the Thesis

In this thesis, we build a spatio-temporal model, design a temporal indexing structure, and use a spatial indexing structure (R-tree). The objective is to have an efficient approach for indexing and retrieving spatio-temporal data. We will refer to our approach by "AIRSTD" (an Approach for Indexing and Retrieving Spatio-Temporal Data). The next chapter will discuss briefly spatio-temporal models, and we build our model for later use. In Chapter 3, we explain spatial and temporal indexing, and we present our proposal for a temporal indexing design. Chapter 4, the main core of our dissertation, discusses our work that contains a spatio-temporal model, spatial and temporal indexes and the algorithms for retrieving data and building a database snapshot at time T. Finally, Chapter 5 summarizes our work and highlights some future research.

Chapter 2: Modeling Spatial, Temporal and Spatio-Temporal Databases

2.1 Introduction

A database contains information that needs to be stored, manipulated and retrieved. A number of computer applications that use databases deal only with the most recent or current data. Such a database is called a snapshot database, which represents the data at the current time with recent values. However, in some applications we might need to know not just the current information but also past or future information as well. Databases dealing with past, present, and future data are called temporal databases or historical databases. The need for temporal databases is crucial in many fields, and there is a large interest in modeling and building such databases. These databases support many kinds of applications such as managerial information, geographical information, and others. An example of this is a bank account where the customer might need to know old balances or old transactions. In such a case, we need a history of this kind of information. These kinds of databases process a temporal dimension to store and manipulate time-varying data.

There is also an interest in databases dealing with geographical information. Databases dealing with geographical information are called spatial databases or geographical information systems (GIS). These systems contain complex data that cannot be simply

represented by atomic values. Such databases deal with a variety of applications, including locations, road networks, paths, lands, weather, environment, airplane networks, moving cars, moving planes and so on. In order to handle geographical information, spatial databases process space dimensions to manage and manipulate the geometries of the spatial data.

It is obvious that most spatial databases are also changing with time. For example, road networks will be constantly changing, areas of lands on maps may change, a moving plane will change its position over time, and so on. In such a case, we have spatial or geographical data that is changing with time. In this case, a spatial database becomes temporal as well; this is called a spatio-temporal database.

When modeling spatio-temporal databases, we need to be aware of many issues. First in the temporal domain, information about the same object might have multiple values at different times. Also we may need to compare database states at two different time points, capture the periods for concurrent events, access times beyond these concurrent periods, represent and restructure temporal data, etc. Spatial databases have also important issues like modeling and structuring different geometries, discretizing continuous data (like computing areas, distances, velocities, etc), defining operations and relations between spatial data, having different layers of spatial information of the same object, and so on. Issues in temporal and spatial databases have been handled in many different ways. A number of these approaches will be presented in this paper. Moreover, issues in both directions will interact when modeling spatio-temporal databases and will

increase the complexity of the model since both temporal and spatial databases are complex by their nature. In addition to previously mentioned issues, the combination of temporal and spatial data introduces even more complications. Examples of these issues include: how to handle relations of spatio-temporal data that have different values at different times, how to deal with moving objects that are continuously changing, operations on two or more discretized spatial data might result into inconsistent answers, etc.

In this chapter we will present various approaches for modeling spatial, temporal and spatio-temporal databases. We will present the OGIS proposal [49] and the Realm based approach [21] in spatial database modeling, Tansel's [51, 52, 53, 54, 55] and Snodgrass' [51] approach in temporal database modeling, and finally Worboys' [60, 61, 62] approach in spatio-temporal modeling.

2.2 Related Work

This section presents important work done in the fields of spatial, temporal and spatio-temporal database modeling. We go over approaches offered in these three fields for the sake of having a clear view of the structures of the database models that are used in these complex databases.

2.2.1 Spatial Database Models

Models and query languages were proposed in the field of spatial databases to handle the complex nature of these databases. One of these proposals is the OGIS proposal [49], in which a set of spatial data types and operations are offered. Another approach is the ROSE Algebra [21], which is one of the best-known models for spatial databases. The approach is often called the "Realm-based approach." It proposes spatial data types with spatial predicates and operations that can be applied on them. In this section we will discuss these two approaches.

The OGIS Proposal

The OGIS [49] proposal shows a hierarchy of geometric data types. A *geometry* is a point, curve, surface, or a collection of these three. Also, a curve is a line string and a surface is a polygon and so on. The detailed representation of the relations (composition and inheritance) between these objects is shown in Figure 2.1.

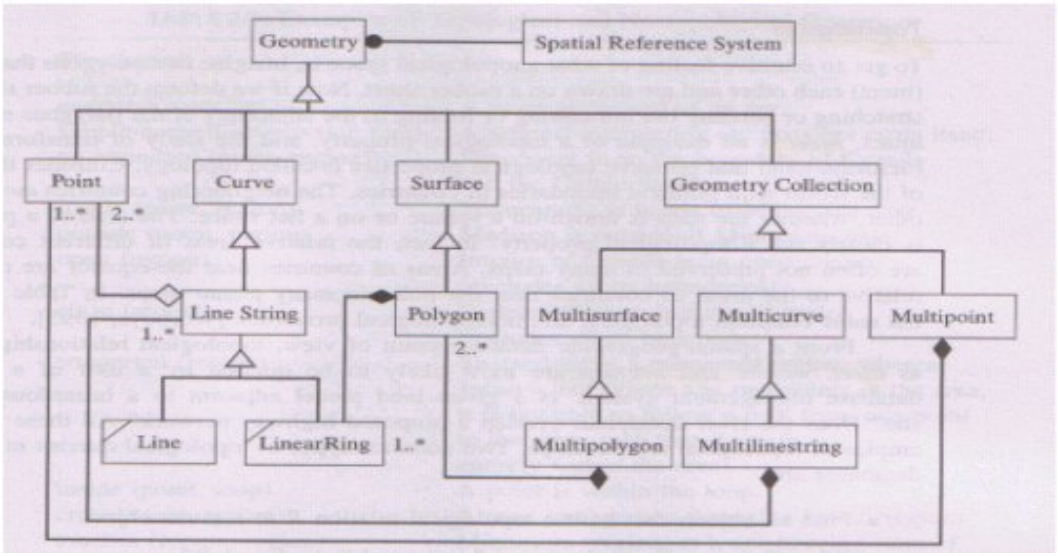

Figure 2.1: An OGIS proposal for building blocks of spatial geometry presented in UML Notation

All of these data types can be used as abstractions with operations and algorithms that manipulate and process them.

The Realm-Based Approach

The Realm Based Approach or ROSE algebra [21] offers the spatial data types points (a set of isolated points), lines (a set of line segments), and regions (a set of non-overlapping regions with holes). A realm is basically a finite set of points and non-intersecting line segments over a discrete domain (a grid) with the following rules:

- If any new point P is inserted, any segment S containing p is split into two segments. Two new segments connecting at P replace the original segment.
- When a new segment S is inserted, its two end points are also inserted
- For each intersection of two segments S1 and S2 at point P, the closest grid point P' is determined. S1 and S2 will be redrawn as polylines (sequence of segments) passing through P' (redrawing a segment is shown in Figure 2.2)

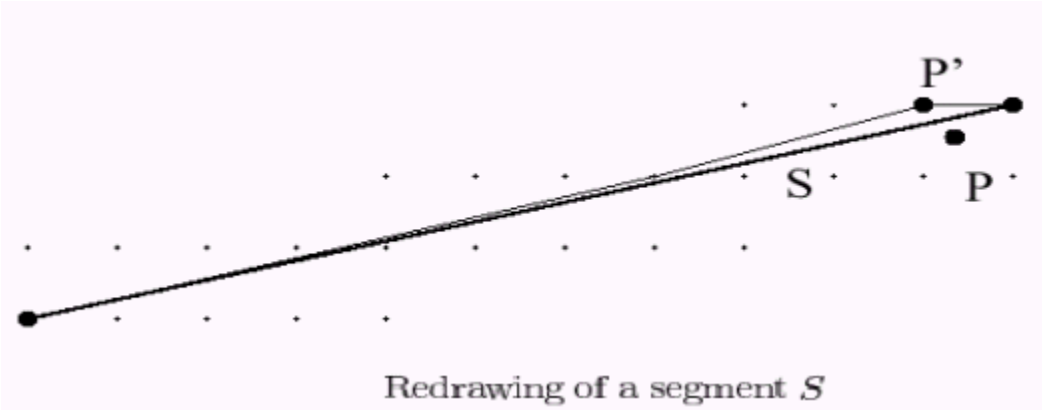

Figure 2.2: Redrawing a segment in the Realm-Based Approach

In other words, rewriting might be needed when inserting elements that intersect existing ones. The reason is that the intersection point that has to be saved might not fall on a grid point, and thus the closest point will be chosen to be the intersection point. This leads to a lot of issues that will be mentioned below. Figure 2.3(a) show elements in a realm (grid) and 2.3(b) shows objects in the realm.

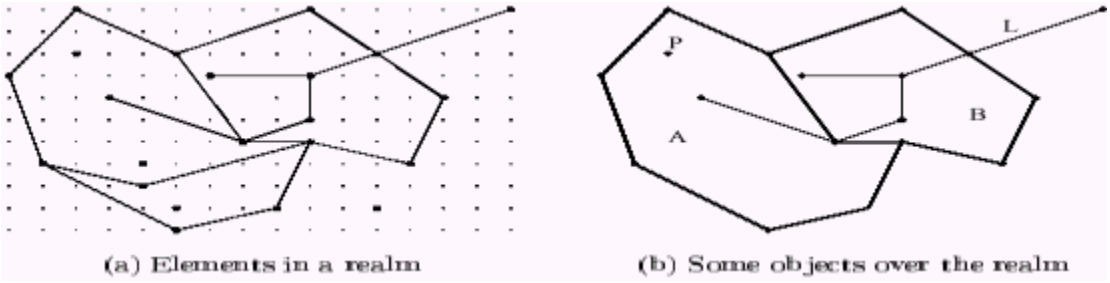

Figure 2.3: Elements in the realm and their object representation

The advantages of the Realm-Based approach are:

- It enforces geometric consistency of spatial objects (for example, a common part of a boundary of two regions will belong to both).
- It guarantees closure properties for the spatial objects (for example, the intersection of two regions can always be represented, because it will always be a new combination of segments of the realm).
- Simplicity and efficiency of spatial operations (with no intersecting points between segments, operations will be simpler and more efficient).
- It insures numerical correctness and robustness of geometric computations.

2.2.2 Temporal Database Models

Proposed approaches for modeling temporal databases fall under these two main categories:

- First Normal Form approach (1NF): Requires fixed length attribute. In this case time instants or intervals represent time stamps.
- Non First Normal Form approach (N1NF): A single tuple tries to capture all the history of the object.

Most first normal form approaches are homogeneous. On the other hand, there are two main forms to model temporal data with N1NF relations. These two are homogeneous data modeling and heterogeneous data modeling, defined as follows:

1. Homogeneous data modeling: The temporal domain of all attributes in a tuple is the same.
2. Heterogeneous data modeling: Different attributes in a single tuple could have different temporal domains.

We present two approaches in this section to have a better understanding of the above definition of temporal database categories.

Heterogeneous Modeling: Tansel's Approach

Tansel [51,52, 53, 54, 55] proposes a N1NF approach for modeling temporal data heterogeneously by using time stamping. In this approach, temporal data or attributes that

are changing with time are attached to a time attribute. A temporal attribute is represented by a pair <t,v> where t is a temporal set or time interval and v is an atomic value of an attribute representing any information changing with time. These N1NF temporal relations should be normalized (flattened) before applying relational algebra operations like projection, intersection, union, difference, and others. In a series of papers [3,4,5], Tansel proposes algebraic operations to flatten (Unnest) and pack back (Nest) the relations. Moreover, Tansel proves the equivalence of the relational algebra and calculus of such N1NF nested relations. Figure 2.4 shows tuples of N1NF relations with Tansel's heterogeneous modeling.

E#	ENAME	SALARY	DEPT
E1	Tom	{<[Jan 85,May 89], 20k>, <May 89, now], 26k>}	{<[Jan 85,May 89], Shoe>, <May 89, now], Toy>}
E2	Anne	{<[Jan 87,May now], 25k>}	{<[Jan 87,May now], Toy>}
E3	Sue	{<[Jan 85,Jan 88], 18k>, <Jan 88, now],22k>}	{<[Jan 85,Jan 88], Toy>, <Jan 88, now], Shoe>}

Figure 2.4: EMP relation in N1NF heterogeneous relation

For this model, Tansel defines many algebraic operations of what he calls the Temporal Relational Algebra. These operations include selection, nesting, unnesting, slicing, temporal atom decomposition, temporal atom formation, dropping time, digitalization and other operations that help manipulate temporal data. The following are definitions of important algebraic operations that Tansel defines (and examples of two of them).

Unnest: This operation flattens the relation over one of its temporal attributes. Multiple tuples of the same object are created where each tuple has information about one time instant of that temporal attribute. Figure 2.5 is shows the relation EMP1 where EMP1= *Unnest* $_{\text{SALARY}}$(EMP).

The Unnest operation over SALARY in this case created multiple tuples of the employee with E#=E1 where each tuple has information about SALARY at one time instant rather than the entire history as in Figure 2.4. The same happens for all tuples that have a history of all values of SALARY at different time. Figure 2.5 shows another example of the Unnest operation, this time over the DEPT temporal attribute.

Nest: the nest operation is the opposite of the unnest operation. It nests together all those tuples of the same key (E# in this case) over one temporal attribute. A history of this temporal attribute will be stored in one tuple in this case.

Example: suppose EMP1 in Figure 2.4 is our relation; then EMP= *Nest* $_{\text{SALARY}}$ EMP1 will result in the tuples in Figure 2.3.

Slice ($S_{\theta,k,p}$): the slice operation replaces the temporal set part of the k^{th} attribute in a relation R by a new temporal set, which is obtained by forming the union, intersection or difference of the temporal set of the kth and pth columns.

DropTime($\pm_i$): discards the temporal set part of the i^{th} attribute. In other words, if the i^{th} attribute is a set of pairs <v,t> where t is the time and v is the value at that time, then all times in all pairs are discarded and the tuple becomes a set of values.

E#	ENAME	SALARY	DEPT
E1	Tom	<[Jan 85,May 89], 20k>	{<[Jan 85,May 89], Shoe>, <May 89, now], Toy>}
E1	Tom	<May 89, now], 26k>	{<[Jan 85,May 89], Shoe>, <May 89, now], Toy>}
E2	Anne	<[Jan 87, now], 25k>	{<[Jan 87,May now], Toy>}
E3	Sue	<[Jan 85,Jan 88], 18k>	{<[Jan 85,jan 88], Toy>, <Jan 88, now], Shoe>}
E3	Sue	<Jan 88, now], 22k>	{<[Jan 85, Jan 88], Toy>, <Jan 88, now], Shoe>}

Figure 2.5: EMP1=*Unnest* $_{\text{SALARY}}$(EMP) (unnesting of the relation on SALARY)

E#	ENAME	SALARY	DEPT
E1	Tom	<[Jan 85,May 89], 20k>	<[Jan 85,May 89], Shoe>
E1	Tom	<[Jan 85,May 89], 20k>	<May 89, now], Toy>
E1	Tom	<[May 89, now], 26k>	<[Jan 85,May 89], Shoe>
E1	Tom	<May 89, now], 26k>	<May 89, now], Toy>
E2	Anne	<[Jan 87,now], 25k>	{<[Jan 87, now], Toy>}
E3	Sue	<[Jan 85,Jan 88], 18k>	<[Jan 85,Jan 88], Toy>
E3	Sue	<[Jan 85,Jan 88], 18k>	<Jan 88, now], Shoe>
E3	Sue	<Jan 88, now], 22k>	<[Jan 85,Jan 88], Toy>
E3	Sue	<Jan 88, now], 22k>	<Jan 88, now], Shoe>

Figure 2.6: EMP2= *Unnest* $_{\text{DEPT}}$(EMP1) (unnesting of the relation on DEPT)

In other words, the main idea in Tansel's [51,52,53,54,55] approach is that the original relation stores a complete history of the temporal attribute in one tuple. Flattening (unnesting) is used before any application of any algebraic operation, and nesting is used to get back to the original form. Other useful operations are also proposed.

Homogenous Modeling: Snodgrass' Approach

Snodgrass [50] proposes a 1NF approach for modeling temporal databases, in which he adds time in two forms: (1) interval relations where time is added implicitly as two time attributes, named Valid-From and Valid-To and (2) event relations where a timestamp in the relation is added as one attribute, named Valid-At. The EMP relation (presented earlier in Tansel's approach) could be viewed as shown in Figure 2.7 when the Snodgrass approach is applied.

E#	ENAME
111	Tom
222	Joe

E#	SALARY	Valid-From	Valid-To
111	20000	9/5/2001	8/6/2002
111	26500	8/7/2002	NOW
222	46700	9/7/2002	NOW

E#	DEPT	Valid-From	Valid-To
111	sales	9/5/2001	8/6/2002
111	Toys	8/7/2002	NOW
222	Marketing	9/7/2002	NOW

Figure 2.7: 1NF relation for EMP relation

Snodgrass also proposes a temporal query language called Tquel by extending the Quel query language. Three new clauses are added, named "When," "Valid," and "As of." The

"When" clause performs a temporal selection by introducing a time condition. It is expressed as follows: ***When τ*** where τ is an expression made up of temporal predicates. The "Valid" clause determines the valid interval of the tuples derived as the result of any query. It is expressed as follows: ***Valid from t1 to t2*** where t1 and t2 are temporal expressions specifying the end points of the interval.

The "As of" clause rolls back the database to a previous time point; this is beyond the scope of this study.

2.2.3 Spatio-Temporal Database Models

In this section, we present one of the most useful works in spatio-temporal database modeling. We briefly discuss Worboys' [60] approach and the definitions used in his work.

Worboys' Approach

Worboys [60] proposes a spatio-temporal model. The following definitions are used in Worboys' proposal of the model:

Database time or transaction time is the time when a transaction actually took place in the information system.

Event time or valid time is the time when the event actually occurs in the application.

A *Bi-Temporal Element* or BTE is defined to be the union of a finite set of Cartesian products of intervals of the form $I_D \times I_E$, where I_D is an interval of database time and I_E is an interval of event time.

A *Simplex* is a point, a finite straight-line segment or a triangular area.

A *Simplicial Complex* is a collection of non-overlapping simplexes.

An *ST-simplex* is a spatial object (simplex) attached to a bi-temporal element <S,T> where S represents the simplex and T represents the *BTE*.

ST-complex: a set of ST-Simplexes subject to some constraints.

An *S-Complex* is a Spatial Complex.

Moreover, Worboys proposes the following table (Table 2.1) of operations applied on the proposed spatio-temporal types:

Operator	Operand	Operand	Resultant
Equals(=)	ST-Complex	ST-Complex	Boolean
Subset (C)	ST-Complex	ST-Complex	Boolean
Boundary (∂)	ST-Complex		ST-Complex
S-project (Π^s)	ST-Complex		S-Complex
T-project (Π^t)	ST-Complex		BTE
ST-β-product(x_β)	ST-Complex	ST-Complex	ST-Complex
ST-Union(U)	ST-Complex	ST-Complex	ST-Complex
ST-Intersection (∩)	ST-Complex	ST-Complex	ST-Complex
ST-difference(\ )	ST-Complex	ST-Complex	ST-Complex
S-select(σ^s_x)	ST-Complex		ST-Complex
T-select(σ^t_Φ)	ST-Complex		ST-Complex

Table 2.1: Worboys' table of operations

For better understanding, we present one of the queries given by Worboys [60] in his paper. This query uses the operations presented in Table 2.1.

Query: does the path currently pass through land that was ever part of Jane's house?

Assumptions: C1 is the path, C2 is Jane's house and now_{DB} is the current database.

Answer using Worboys' operations: $\Pi^{s}(\sigma^{t}_{t \supseteq nowDB}\ (C1)) \cap (C2) = \varnothing$

2.3 A Spatio-Temporal Model for AIRSTD

This section presents the spatio-temporal model that we will use when we present our solution later (AIRSTD). The idea of the model is not new since we combine different spatial and temporal database modeling approaches. We present the spatial data types, operations, and the way we add the time dimension in our model.

2.3.1 Spatial Data Types

Consider the following abstract object definitions of Points, Lines, and Regions where:

Points: is a set of points

Lines: is a set of lines

Regions: is a set of regions.

Definition 1: Let GEO = {Points, Lines, and Regions} be defined as the set of points, lines, and regions that define a geometry.

2.3.2 Operations

These operations are offered in all spatial models, and they are not new. The operations will be divided into two main categories: (1) algebraic and (2) non-algebraic

Table 2.2 (similar to Worboys' operations [12]) shows all possible algebraic operations and predicates that could be applied on spatio-temporal data.

Operation	**Operand**	**Operand**	**Return Type**
Spatial Union (U)	GEO	GEO	GEO
Spatial Intersection (∩)	GEO	GEO	GEO
If Equal (==)	GEO	GEO	Boolean
IfSubset (C)	GEO	GEO	Boolean
Boundary (β)	GEO		Region
Area (Δ)	GEO		Real
Overlaps (θ)	GEO	GEO	Boolean

Table 2.2: Table of operations

Table 2.3 shows some of the non-algebraic operations. There are many kinds of non-algebraic operations such as: topological, non-topological, and directional. These operations exist in almost all spatial approaches.

	Operation	Operand	Operand	Return Type
Topological	Overlap	Region	Region	BOOLEAN
	Touches	Region	Region	BOOLEAN
Non-Topological	Inside	Region	Region	BOOLEAN
	Length	Line		Real
	Perimeter	Region		Real
	Euclidean-Distance	Point	Point	Real
Directional	North	Region	Region	BOOLEAN
	South	Region	Region	BOOLEAN

Table2.3: Non-algebraic operations

2.3.3 Adding the Time Dimension

This section extends the model with a time dimension. As mentioned before, time will be added in two forms: Flat Temporal Relation and Nested Temporal Relation.

First, consider the following definitions:

Valid_From: is the left boundary of the interval of validity of the data or the first time point when the data is valid.

Valid_To: is the right boundary of the interval of validity or the last time point when the data is valid.

T: is an interval of the form [*Valid_From*, *Valid_To*]

Now: represents the current time

2.3.4 Flat Temporal Relation

Definition 2: A Spatio-Temporal relation that has spatial information changing with time is defined as the following relation $\{A_1,\ldots,A_n\}$ with attributes $A_1,..,A_n$ where one of these attribute A_i is of type GEO and any two A_j and A_k represent the interval boundaries *Valid_From* and *Valid_To* respectively.

To illustrate an example of a Spatio-Temporal relation consider the following definition of the CITY relation.

CITY {Name, Geometry, Valid_From, Valid_To}

Figure 2.8 represents some examples of the rows stored in the CITY relation. G1, G2, and G3 represent the geometries of the objects. Details about these geometries could be stored in the table itself or in another table. However, this is beyond the scope of our discussion here.

Name	Geometry	Valid_From	Valid_To
Manhattan	G1	1/2/1967	Now
Manhattan	G2	1/3/1950	1/1/1967
Brooklyn	G3	2/4/1950	Now

Figure 2.8: Some data examples (in a database table).

2.4 Summary

In this chapter, we presented a group of approaches that propose models for spatial, temporal, and spatio-temporal databases. We started with spatial databases and different

proposals for spatial database models (both data types and operations). After that, we discussed temporal databases where we have shown many ways of modeling represented by different approaches. Finally, we presented a model for spatio-temporal databases and our spatio-temporal model that will be used in our thesis. We think this order (spatial, temporal, and then spatio-temporal) is the best way to build an understanding of spatio-temporal models which is a combination of spatial and temporal modeling approaches.

Chapter 3: Indexing Spatio-Temporal Databases

3.1 Introduction

Spatio-temporal databases usually contain huge amounts of data. Entire cities now have detailed spatial databases containing a model of all of their physical infrastructure as well as all political boundaries and land ownership. The sheer volume of data is compounded by the fact that spatial data structures are complex, data can change rapidly where both old and new information needs to be available. Space complexity is always a cause of time complexity, and manipulating large numbers of data is often expensive and needs some techniques like clustering and/or indexing the data.

Indexing databases is basically clustering the data according to some rules and then building an index (or indexes) that lead to these clusters or groups. In this chapter, we will talk about two main kinds of indexing (1) spatial indexing and (2) temporal indexing. The reason for presenting these two kinds is that we will use both of them when we provide our approach and algorithms in Chapter 4.

3.2 Related Work

In this section we will present some previous work in the domains of spatial and temporal indexing. The section is divided into two subsections: spatial indexing and temporal indexing.

3.2.1 Spatial Indexing

There are many kinds of spatial indexing. However, we will present two of the most famous ones (R-tree and Quad-tree). In our approach later we will use the "R-tree" index as part of the global structure that our algorithms depend on.

The R-Tree

An R-tree is a hierarchical, height-balanced external memory data structure proposed by Guttman in [25]. It is a general view of the B-tree for multidimensional spaces. Multidimensional objects are represented by what is called a Minimum Bounding Rectangle (MBR), where an MBR of an object is the smallest rectangle that encloses the object and has its sides parallel to the axes. Figures 3.1 shows an example of a spatial object enclosed in a rectangle (MBR), Figure 3.2 shows a group of MBRs, and Figure 3.3 show how these MBRs are indexed in an R-tree structure.

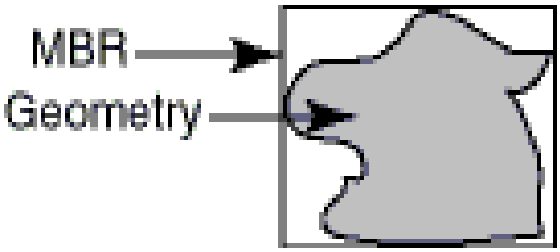

Figure 3.1: An MBR enclosing geometry

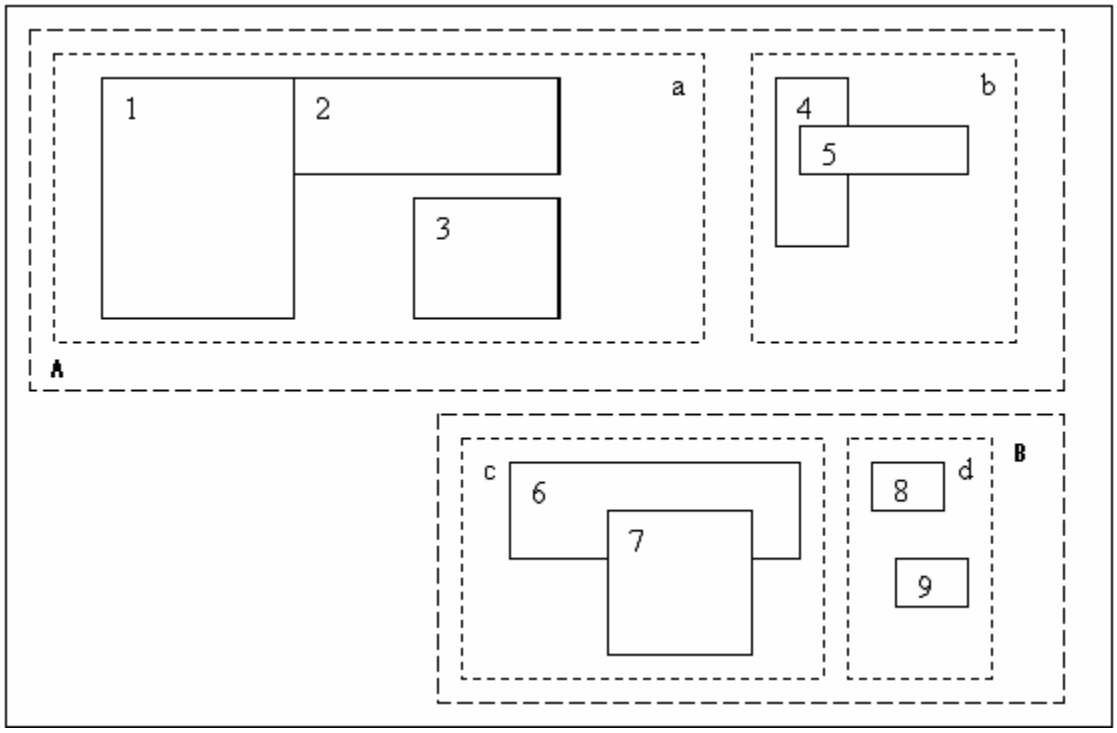

Figure 3.2: A collection of spatial objects and MBRs

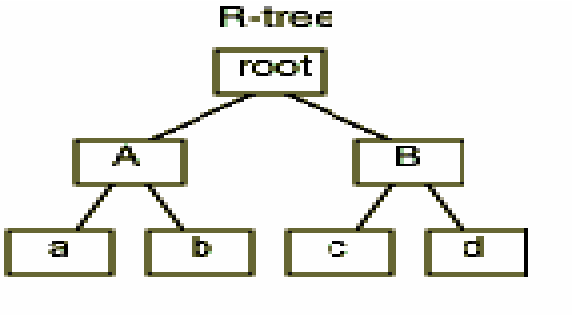

Figure 3.3: R-tree of MBRs

The R-tree [25] consists of directory and leaf (data) nodes, each one corresponding to one disk page (secondary storage). Directory nodes contain entries of the form (container, ptr) where “ptr” j is a pointer to a successor node in the next level of the tree, and “container”

is the MBR of all the entries in the descendant node. Leaf nodes contain entries of the form (container, oid), where "oid" is an object-identifier used as a pointer to the real object, and it is the MBR of the corresponding object. Each page can hold up to "B" entries and all the nodes except the root must have at least "m" records (usually m=B/2). Thus the height of the tree is at most $\log_m N$ where N is the total number of objects.

Searching an R-tree is similar to searching a B-tree. At each directory node, we test all entries against the query, and then we visit all child nodes that satisfy the query. However, MBRs in a node are allowed to overlap. This is a potential problem with the R-tree, since, unlike B-trees, we may have to follow multiple paths when answering a query, although some of the paths may not contribute to the answer at all. In the worst case we may have to visit all leaf nodes.

3.2.2 Temporal Indexing

In this section, we present an example of temporal indexing to provide an understanding of how temporal databases are indexed according to their time validity. We will not use this index in our approach; however there are many similar rules that apply in this and our temporal index.

The Snapshot Index

The snapshot index [58] is a structure that indexes different groups of databases, each of which is called a snapshot database. A snapshot database is a database that is valid at a

specific time or period. Before we present the snapshot index, consider the following related rules:

- Assume for simplicity an initially empty set S. As time proceeds, objects can be added to or deleted from this set.
- When an object is added to S and until (if ever) it is deleted from S, the object is called “alive.”
- This is represented by associating with each object a semi-closed interval, or lifespan, of the form: [start time, end time).
- While an object is alive it cannot be re-added in the set (i.e., the set contains no duplicates).
- Deletions can be applied only to alive objects.
- When an object is added at time T its start time is T (time of addition) but its end time is yet unknown. Thus its lifespan interval is initiated as [t, now), where “now” is a variable representing the always increasing current time.
- If this object is later deleted from S, its end time is updated from “now” to the object’s deletion time.
- Since an object can be added and deleted many times, objects with the same id may exist but with non-intersecting lifespan intervals (i.e., such objects could be alive at different times).
- The state of the set S at a given time t, called S(t), is the collection of all alive objects at time t.

The snapshot index [58] deals with the pure-snapshot problem where a time instant t_i is specified and the state of the set $S(t_i)$ has to be retrieved. It uses three main structures: (1) a balanced tree (time-tree) structure that indexes data pages by time, (2) a pointer structure (called access-forest) among the data pages and (3) an ephemeral hashing scheme. The time-tree and the access-forest enable fast query response, while the hashing scheme is used for fast updates.

Here are some rules that should be applied when manipulating the snapshot index:

- Objects are stored sequentially in data pages in the same order as they are inserted into set S.
- When a new object O with key "id" is added at time t, a new record of the form <id,[t, now)> is created and inserted in a data page.
- At any given instant t, there is only one data page that stores records, called the acceptor (data) page.
- When the current acceptor page becomes full, a new one is created.
- The time when an acceptor page was created and its page address are stored in the time-tree.
- As acceptor pages are created sequentially the time-tree is easily maintained.
- For object insertions, the sequence of all data pages resembles a regular log but with two main differences: (1) deletion updates are handled differently, and (2) additional links (pointers) among the data pages exist.

- Object deletions are not added sequentially; rather they are in-place updates. When object k is deleted at time t', its record is first located and then updated from <k,[t, now)> to <k,[t,t']>.
- When we insert an object into the set S, its id and the address of the page that stores the object's record are inserted in the hashing scheme.
- When an object is deleted its id is removed from the hashing scheme.

Moreover, there are many constraints and rules that apply to the other structures (in addition to the balanced tree) used by the snapshot index. The next few paragraphs describe briefly a few procedures that should be applied to the structures forming the snapshot index.

Storing only one record for each object assumes that the records of the objects in S(t) may be spread over various data pages. Accessing all pages with alive objects at t would require too much I/O (There could be O(a) pages). To achieve good record clustering, a "controlled" copying technique is used that keeps the total space linear with respect to the number of updates. The copying procedure is based on the concept of page usefulness.

Consider the number of "alive" records a page contains after it becomes full. As long as a page contains a number (uB) of "alive" records, the page is called useful. During this time the page contains a big part of the answer for S(t). Answering a pure-snapshot query about some time t requires us only to locate the useful pages at t; each such page will contribute at least "uB" objects to the answer. The usefulness parameter "u" is a constant that tunes the behavior of the snapshot index.

Acceptor pages are special. During the time when a page is the acceptor page it may contain fewer than “uB” alive records. By definition an acceptor page is also called a useful page. Such a page may not give enough answer to justify accessing it, but it must still be accessed. Having only one acceptor page for each time instant does not affect query performance.

Let [u-start-time, u-end-time] denote a page’s usefulness period; u-start-time is the time the page started being the acceptor page. When the page gets full it may continue to be useful (as long as the page has at least “uB alive” records); otherwise, it becomes non-useful. The next step is to cluster the “alive” records for each time t among the useful pages at t. When a page becomes non-useful, an artificial copy occurs that copies the “alive” records of this page to the current acceptor page. The non-useful page behaves as if all its objects are marked as deleted but copies of its “alive” records can still be found from the acceptor page. Copies of the same record contain subsequent non- overlapping intervals of the object’s lifespan. The copying procedure reduces the original problem of finding the “alive” objects at time t into finding the useful pages at t. The solution of the reduced problem is facilitated through the access-forest (the second data structure used by this approach).

Finally, the access-forest is a pointer structure that creates a logical “forest of trees” among the data pages. Each new acceptor page is appended at the end of a doubly linked list and remains on the list for as long as it is useful. When a data page d becomes non-

useful: (a) it is removed from the list and (b) it becomes the next child page under the page c preceding it in the list (i.e., c was the left sibling of d when d became non-useful). As time proceeds, this process will create trees of non-useful data pages rooted under the useful data pages of the list. The doubly linked list and the child lists are implemented by using four pointers per page. As a result, each page has one pointer that points to the next page, one pointer to the previous page, one to the first page of its child list and one to the last page of its child list. The next, previous and last child pointers are updated so that they always point to the current pages in the corresponding positions.

3.3 A Temporal Index Structure for AIRSTD

This section presents an approach to index spatio-temporal databases. The indexing approach can be applied on the model presented in the previous subsection.

3.3.1 Temporal Index for AIRSTD: Changes Temporal Index

The indexing approach is as follows. Multiple versions are used to store the spatio-temporal database. Only the current version of the database contains information about all objects (or pointers to them). All old versions include objects (or pointers to them) that are valid at that version but not in the current versions. In other words, these objects are the changes of the spatial information with respect to the current version. Figure 3.4 explains this idea, where the last version (current) includes the whole database that is available between some time n and now. All previous versions include only the tuples that are different at the time of their interval. We will call our structure the “Changes Temporal Index” (CTI).

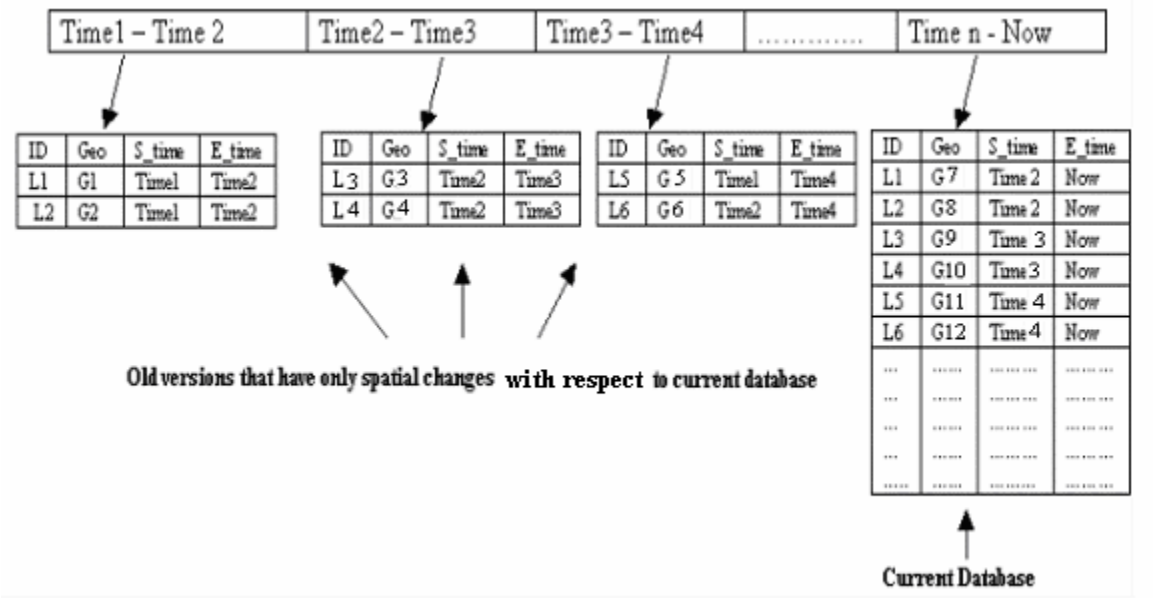

Figure 3.4: Time interval index pointing to spatial changes and the current database.

3.3.2 Rules and Constraints on Data Manipulation

In general, when an object in the current version is changed, all older versions should have spatial changes information relative to the new current version. Adding, updating, and deleting tuples from the current database or accessing old versions using the CTI should follow these rules:

Updating Tuples in the Current Database

If a change of spatial information happens at time T for an object in the current version, the following should be done:

- A new tuple of the same object (with new spatial information) is added to the current version.
- The old tuple stays in the current version.

- The old tuple (or a pointer to the object) will be added to any old version that does not contain a tuple of the same object and satisfies the tuples time interval.
    - The time interval of the tuple satisfies the time interval of the version if the intersection of the two intervals is not empty.
- The right boundary of the interval of the old tuple will take the time of the update and hence become [T1, T-1], assuming that the tuple was created at T1.
- The new tuple (of the same object) will have its time interval as follows: [T,now).

Figure 3.5 shows what happens when the tuple with ID L4 is updated. A new tuple with ID L4 but different time interval is added. The old tuple is added as well to all other versions that do not have any tuple of the same ID.

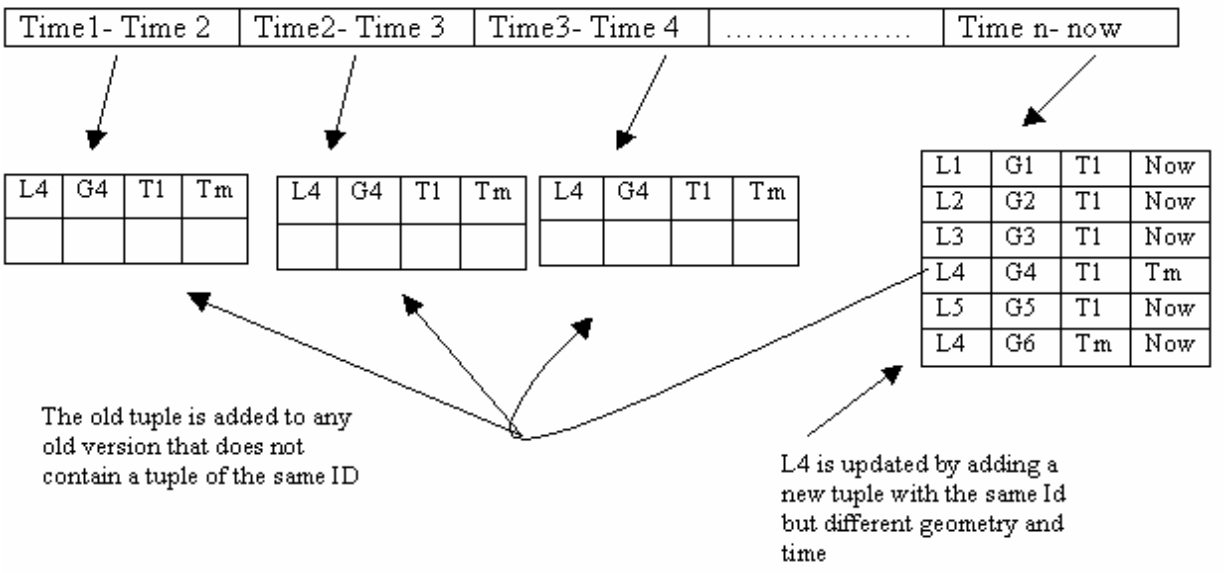

Figure 3.5: Updating tuple with ID l4 in the current database.

Adding a New Tuple to the Current Database

When a new object is added to the current version at time T, nothing happens to the old versions. The new version takes the interval [T, now).

Deleting a Tuple from the Current Database

When a tuple is deleted at time T from the current version, we follow the update procedure. However, we do not add a new tuple. The deleted tuple interval becomes [T1,T] assuming that it was created at T1.

Querying or Accessing Old Versions

When a user want to access any version of the database at time T that belongs to an old version of time Interval INT, the DBMS should do the following:

- Get all tuples in the INT version that have their time intervals satisfy T
    - A time interval [T1, T2] satisfies T if $T1 \leq T \leq T2$
- Get all tuples in the current version with time interval that satisfies T.

Figure 3.6 shows how data is retrieved when the user asks for database information at time T, assuming $Time2 \leq T \leq Time3$.

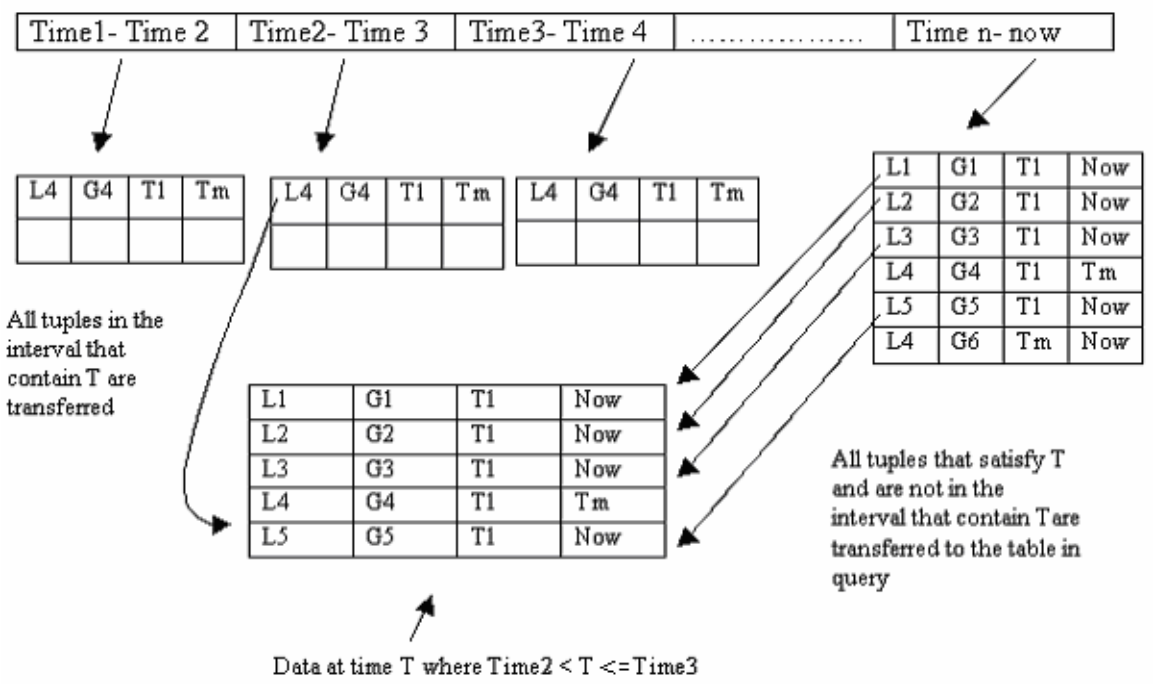

Figure 3.6: Retrieving the database table at time T.

Moving to a New Version

Moving to a new version at time T can be easily done with the following steps:

- Assign the high boundary of the current interval to T-1.
- Create a new interval with the boundaries [T, now) and add it to the temporal index.
- Make the new interval ([T, now)) point to the current version.
- Create a new table.
- Move all tuples in the current version that do not satisfy the new interval (their right boundary ≠ now) to the new table.
- Make the old current version interval point to the new table.

When the time comes to switch to a new interval, all tuples in the current version are transferred to the new current version except for those whose high boundary does not satisfy the low boundary of the new current version interval. In other words, only those with high boundary = “now” are transferred.

3.4 Applying the Binary Search Tree Structure

Since the time index is built with ordered entries (according to time) we can apply the binary search algorithm when searching for any index. Binary search can be applied when an array structure (of fixed size) is used. On the other hand if a dynamic structure is used then efficient pointer-based structures like a binary search tree or B-tree can be used and applied on this index. We will demonstrate the binary search tree as an example; however, other structures could be used if they provide better performance in some applications. We will add minor characteristics to the binary search tree index. First, an additional pointer will be used to point to the last entry, which is the entry pointing to the current version. The reason for adding this pointer is that most of the queries will access the current version, and hence we need a means of direct access to that version. A second issue we have to worry about is tree balance. The tree will be growing with time, and if we use the traditional binary search tree, the new nodes will always be added on the right (according to the defined ADT) and this will result in an unbalanced tree. In this case we have to make sure the tree is balanced. To do this, we can use a tree-balancing algorithm.. Figure 3.7 illustrates the binary search tree index that could be used as our temporal index.

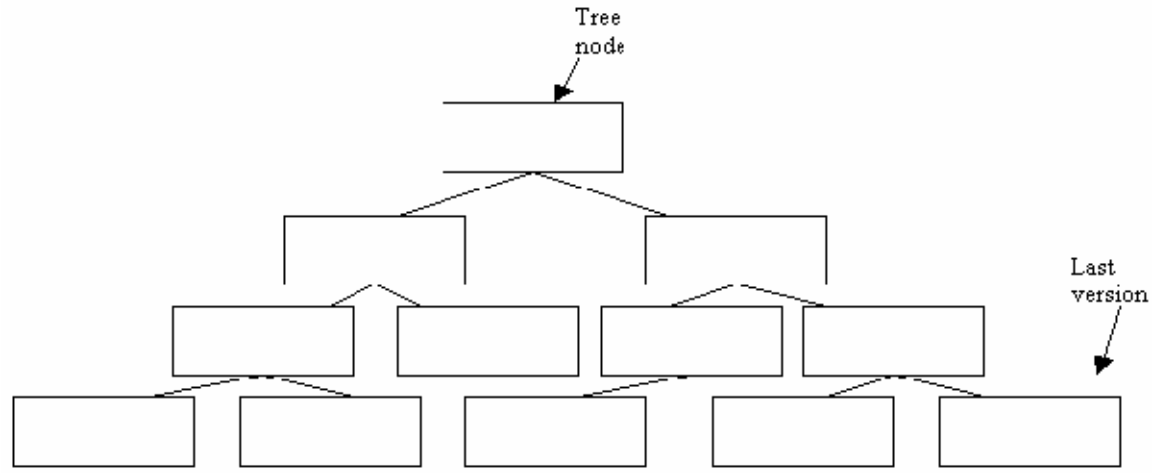

Figure 3.7: Temporal index using an updated binary search tree structure

Note that all the rules and constraints that we used for the array-based index can be applied to the binary search tree index.

3.5 Issues and Conclusions

This section includes some assessments of the approach. We present some positive and negative features of the behavior of the temporal index used in AIRSTD.

3.5.1 Advantages

Here are some positive aspects of the approach:

- This approach saves some space in the sense that data (or pointers to data) in the "Now" version, that have the same time interval are not duplicated in old versions.
- Transfer from one interval to another can be done in one step by changing the pointer in the index.
- The approach saves time as well when searching and querying at the current time or any time in the current interval.

- The approach is efficient when building a snapshot.

All these issues are general in the sense that they have not been compared to other approaches. In Chapter 4, we present a detailed comparison of this indexing approach with other well-known approaches.

3.5.2 Disadvantages

Here are some negative aspects of the approach:

- The approach needs additional time to update or delete a tuple since we need to access some of the old versions.
- The approach wastes some space when tuples in the old versions are duplicated. We do this in ordsr to gain time when building a snapshot.

3.6 Summary

This chapter presents and discusses various ways of organizing spatio-temporal data in order to gain performance while querying and manipulating spatio-temporal databases. The chapter goes over some background and previous work done in the field of indexing spatial and temporal databases. The chapter illustrates the importance of clustering and indexing data in general and spatio-temporal data in particular, and the contribution of indexing to decreasing the time and space complexity. Finally, we present our approach for indexing spatio-temporal databases using what is called “changes versions” where only changes are recorded in old versions rather than entire snapshots. We also discuss the way we deal with this index and the possibility of using efficient structures if needed.

In the next chapter we add a new index (R-tree) to our structure in order to take advantage of both ways of indexing (spatial and temporal) and gain time in both kinds of queries (spatial and temporal).

Chapter 4: Retrieving Spatio-Temporal Data Using Temporal and Spatial Indexing

4.1 Introduction

This chapter includes the main core of our thesis study. In previous chapters we presented our approaches for modeling and indexing spatio-temporal data. Now we will briefly review these approaches and extend our indexing structure in order to present our efficient algorithms for retrieving spatio-temporal data.

This chapter is organized as follows. Section 4.2 includes recent work that is applied to a currently running project. This work is directly related to our study. Sections 4.3 and 4.4 briefly discuss our modeling and indexing approaches and structures that will be used in our algorithms. Section 4.5 presents the proposed algorithm for retrieving spatio-temporal data at time T. Section 4.6 presents the proposed algorithm for building a snapshot at time T. Section 4.7 provides analysis and a performance study of our work and comparisons to other work. Section 4.8 presents some experimental results of the algorithms and a comparison with other algorithms. Finally, Section 4.9 covers a summary of this chapter.

4.2 Related Work: A Recent Study and Application to Spatial and Temporal Information Structuring

This section presents recent work [34,35] in spatial and temporal information structuring for natural (volcano) risk monitoring. The work was presented in "GIS Planet 2005," which is the Second International Conference and Exhibition on Geographic Information (Estoril, Portugal). In the same conference, important material [26] from our dissertation was presented and published.

In their papers [34,35], Noel, Servigne, and Laurini discuss methods and approaches that they designed and applied for volcanic activity monitoring of a Mexican volcano. The project uses heterogeneous types of sensors that are responsible for delivering real-time spatial information. The papers describe the following main points:

- Sensor data properties: multidimensional data (temporal and spatial) delivered by heterogeneous types of sensors.
- Volcanic activity monitoring: the use of sensors to deliver the change of spatial data of a Mexican volcano in real time, and the use of this data in the analysis of volcanic activities.
- An efficient approach for structuring the spatio-temporal data: use of an efficient structure (PoTree) used to store and retrieve the spatio-temporal data.

In this section we will concentrate on the last point (the PoTree), which is directly related to our thesis study. The PoTree is a spatial index with two kinds of structures: a Kd-tree and a B+ tree. The Kd-tree (a well-known structure for efficient access of spatial objects)

is not a traditional tree. Instead, each leaf node in the tree points to a B+ tree. The B+ tree (also a well-known structure) uses time to order its nodes, where each node is a spatial object associated with time. Figure 4.1 represents the PoTree structure.

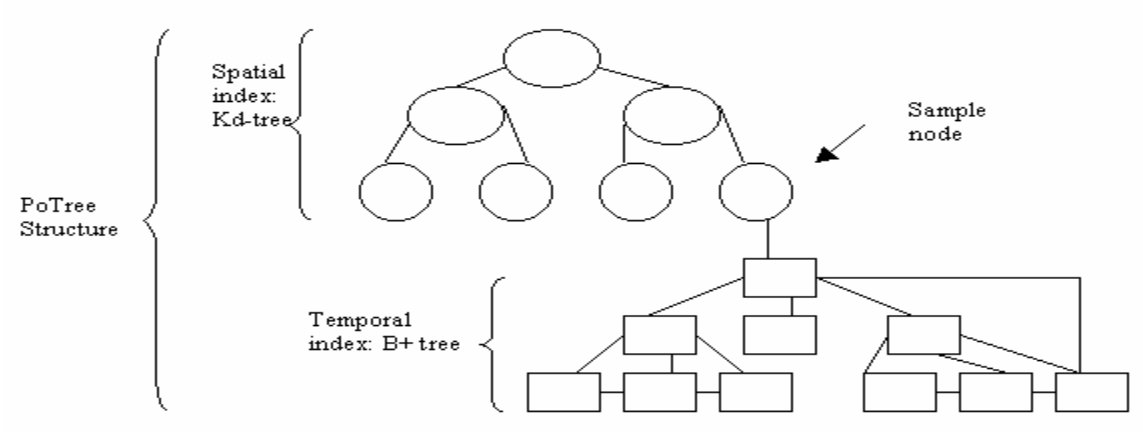

Figure 4.1: The PoTree

We will include analysis of this structure later in our analysis section (Section 4.7.1).

4.3 The Proposed Spatio-Temporal Database Model

In this section we briefly present the spatio-temporal model we designed and built in Chapter 2.

These are the spatial data types we use:

- Points: is a set of points
- Lines: is a set of lines
- Regions: is a set of regions.
- *GEO = {Points, Lines, and Regions} is defined as the set of points, lines, and regions that define a geometry.*

Again we present the way we add the time dimension to our spatial relation to become a spatio-temporal relation. Consider the following definitions:

Valid_From: is the left boundary of the interval of validity of the data, or the first time point when the data is valid.

Valid_To: is the right boundary of the interval of validity, or the last time point when the data is valid.

T: is an interval of the form [*Valid_From*, *Valid_To*]

Now: represents the current time

With these definitions we can define the spatio-temporal relation as follows:

A Spatio-Temporal relation that has spatial information changing with time is defined as the following relation $\{A_1,\ldots,A_n\}$ with attributes $A_1,..,A_n$ where one of these attribute A_i is of type GEO, and any two A_j and A_k represent the interval boundaries *Valid_From* and *Valid_To*, respectively.

Again, this section is a brief repetition of what we presented in Chapter 2. The reason for the repetition is to have a clear view of the structures we are using in our approach.

4.4 The Proposed Spatial and Temporal Indexes

This section is also a brief view of what we presented earlier (Chapter 3). We will include our combination of spatial and temporal indexing; later in this chapter we will propose our algorithms that will be applied to these structures.

4.4.1 Temporal Index: Changes Versions

In this sub-section, we again briefly present our temporal indexing structure (Chapter 3). As shown in Figure 4.2, the temporal indexing structure is a binary search tree of intervals, where each node points to a list of elements that represent (each) spatial information about one object.

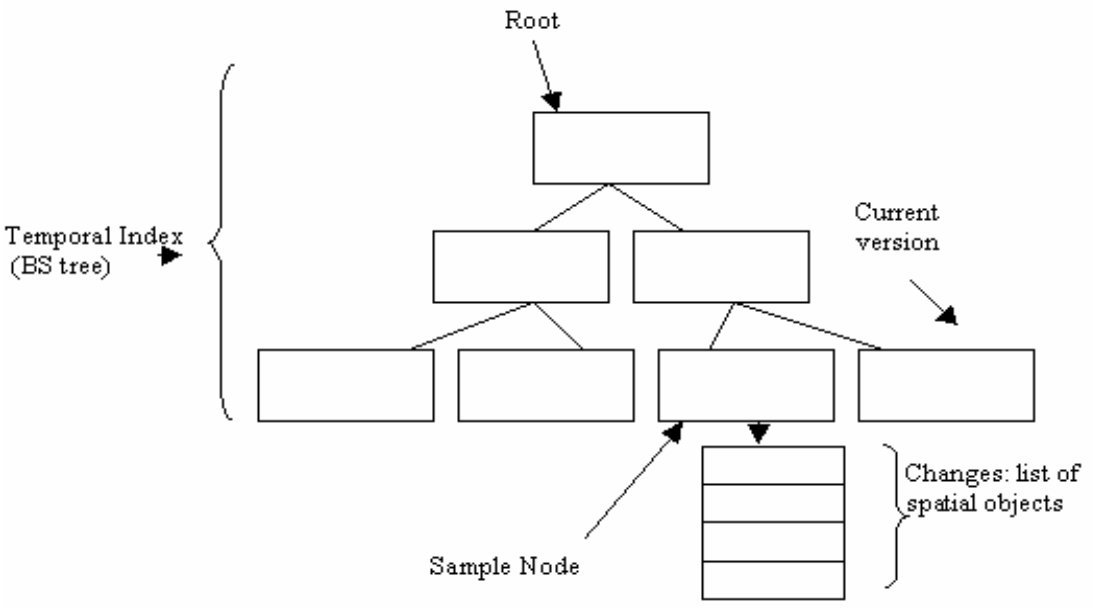

Figure 4.2: Temporal indexing structure

All rules and constraints of this structure were presented in Chapter 3.

4.4.2 Spatial Indexing Extension for the Current Version

This subsection introduces an extension of our indexing approach. We extend our structure by a spatial index (R-tree, see Section 3.2.1) that is only applied to the current version. The reason for applying spatial indexing to the current version only, is that we assume that any other version does not include all the information (snapshots), and hence does not have a large amount of data. This assumption applies to many applications that change discretely with time. A detailed analysis will be provided later (Section 4.7) on the efficient use of our approach. In addition to our structure, we will present the relation between the temporal and spatial indexes.

Figure 4.3 shows the new indexing structure that is a combination of our temporal indexing structure and a new structure (R-tree) that is only applied to the current version. We will call this structure the AIRSTD (Approach for Indexing and Retrieving Spatio-Temporal Data) structure.

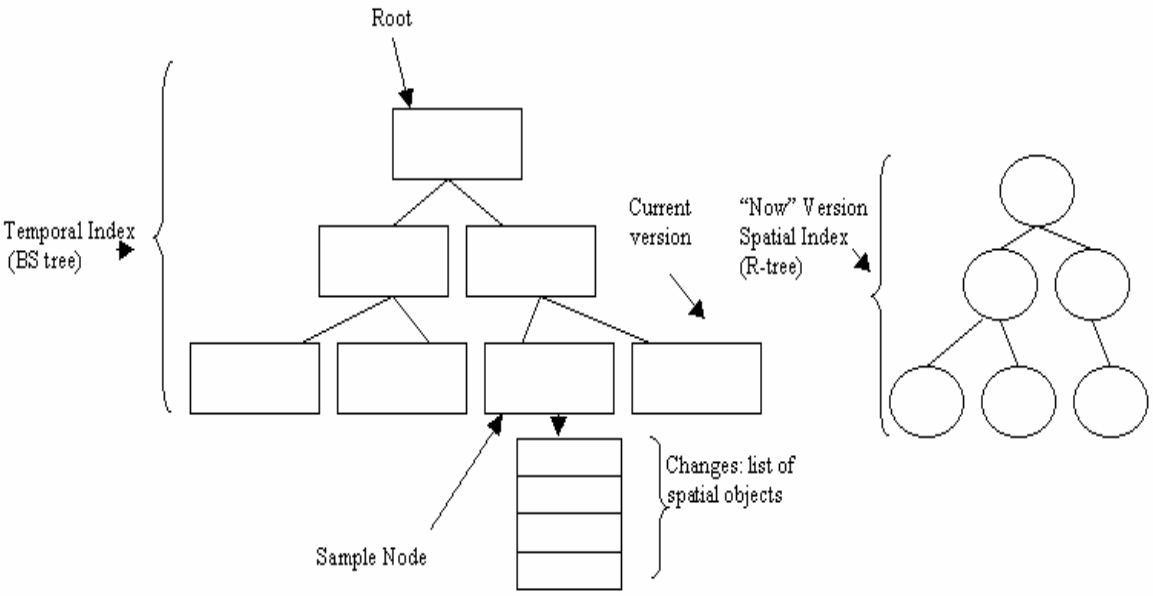

Figure 4.3: AIRSTD Structure

The “Now” Version (R-tree) includes all items that have their right time boundary as “now,” which means still valid at the current time. On the other hand, the current version that exists in the temporal index (BS tree) has all the objects that are valid in the current interval but not at the current time. In other words the objects satisfy the interval of that node but their right boundary is not equal to “now.”

All pre-mentioned rules and constraints still work on the new structure, with one difference. We use the R-tree to access our current objects (valid at “now”). Any need for an old snapshot will be a mix of the “now” version (R-tree) and the list of changes that are valid at that time (we get this from the BS Tree).

4.5 The Proposed Algorithm for Retrieving Spatio-Temporal Data at Time T

In this section we propose our algorithm for retrieving a spatial object that was valid at time T. Using the AIRSTD structure, we first search for that object in old (using the temporal index) and “Now” version (using the R-tree) and check its time before we retrieve it. Later in this section we illustrate an example of a possible execution of the algorithm.

Algorithm: Retrieve Spatio-Temporal Data at a time T

Input: Spatial object “O” (name or id) and Time “T”

Task: Search for O at time T and retrieve all its data

Step 1: Search the temporal index for the right interval that satisfies T

If found go to step 2

Otherwise terminate (invalid time)

Step2: Search the table pointed to by the interval found in step 1 for O

Step 3: If found

Retrieve O and terminate

Else go to step 4

Step 4: Search the R-tree ("Now" version) for that spatial object

Step 5: If found

Check its interval of validity (TI)

If T satisfies TI retrieve O

Else O (at T) does not exist

Else O (at T) does not exist

Discussion and Example

While designing this algorithm, we considered which structure to search first: the temporal index or the spatial index (R-tree). This discussion does not affect the worst-case scenario of the algorithm; however, it has an effect on the best-case scenario, which could be the case most of the time if a study is made on the behavior of data values and how often they change with time.

In cases where spatial data is changing rapidly (but discretely), it is very important that we search the temporal index first and go to an older version, since the probability of finding the object is very high (it could be close to 100%).

Execution Examples

In order to give a better example view of this algorithm, we present two examples of possible executions of the algorithm. Our examples will consider two cases: (1) when the spatial object at time T exists in the temporal index and (2) when it exists in the spatial index.

First case: Spatial Object O exists in the temporal index.

In this case, only steps 1, 2, and 3 of the algorithm will execute. The reason for this is that we first check for the interval [T1, T2] that satisfies T. Then after the interval is found we search for the object that exists in the list pointed to by the interval. Figure 4.4 illustrates this execution.

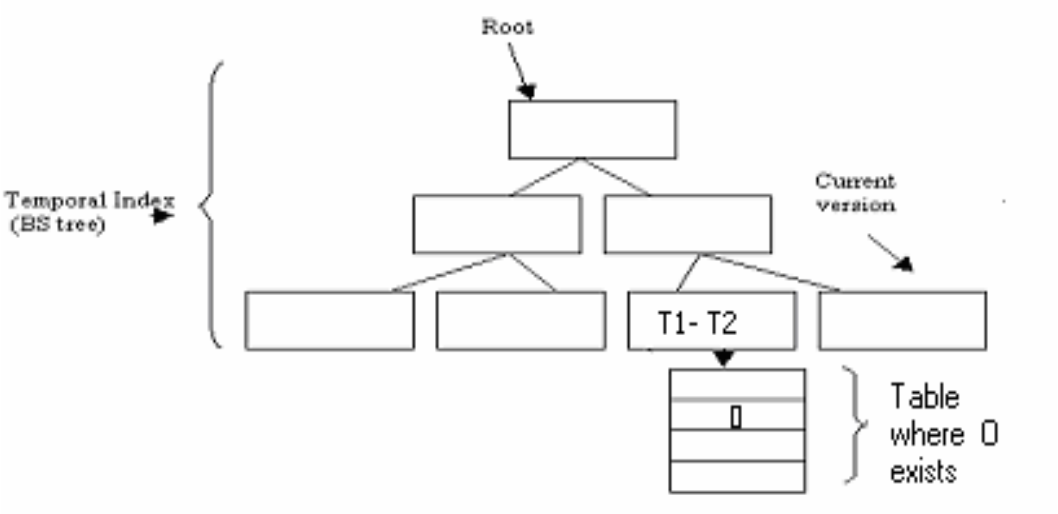

Figure 4.4: An example where a spatial object O exists in the temporal index.

Second case: Spatial Object O exists in the spatial index

In this case, all the steps of the algorithm will execute. We first check the temporal index, and when O is not found we switch to the spatial index where the object exists. Figure 4.5 illustrates this execution.

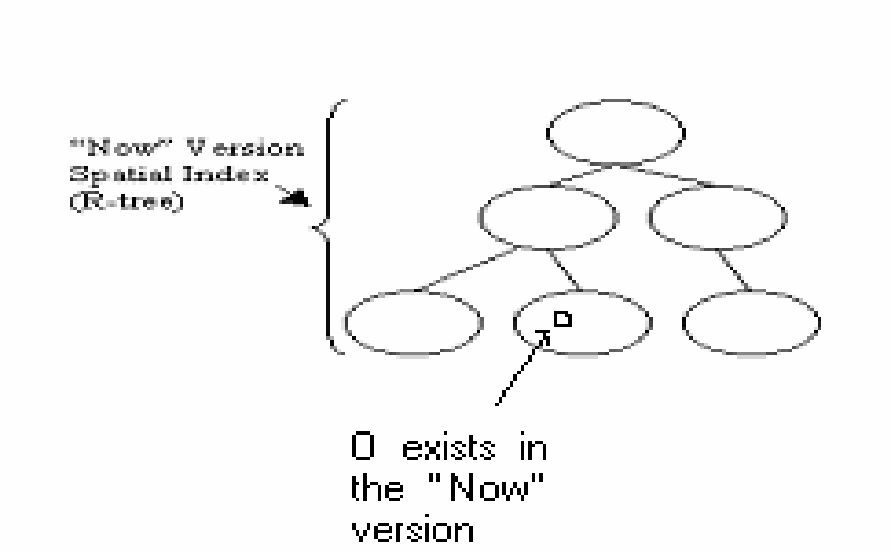

Figure 4.5: An example where a spatial object O exists in the spatial index.

4.6 The Proposed Algorithm for Building a Snapshot at Time T

Building a snapshot of the database is an important task that is needed in many applications. Our approach for organizing and indexing the spatio-temporal database (AIRSTD structure) does not have full snapshots of the database. Instead, we have versions where only changes are recorded. There are two main reasons for following this approach:

1- Unlike snapshots, our approach records changes during the version intervals.

2- We save a lot of space by not duplicating those data values that do not change over time.

We will discuss these issues in detail later in the analysis section. In this section we present our algorithm for building a snapshot from versions where only changes are recorded. Moreover we illustrate an example of the execution of the algorithm.

Algorithm: Build a snapshot at time T

Input: Time "T"

Task: Return the database snapshot available at that time.

Step1: Traverse the "Now" version (R-tree), return all objects (with their time intervals) that satisfy T, and store these objects in a temporary structure (snapshot).

Step 2: Search the temporal index for the interval (TI) that satisfies T.

Step 3: Go to the table pointed to by TI and store all its contents in the temporary table (snapshot).

Step 4: Return the temporary table that contains the snapshot at T.

Figure 4.6 illustrates an example of the execution of the algorithm. First, we check the time validity of each object in the spatial index; if its interval satisfies T then we consider the object otherwise it is discarded. Later, the temporal index is checked for the interval (T1-T2) that satisfies T where the entire objects pointed to by that interval are retrieved.

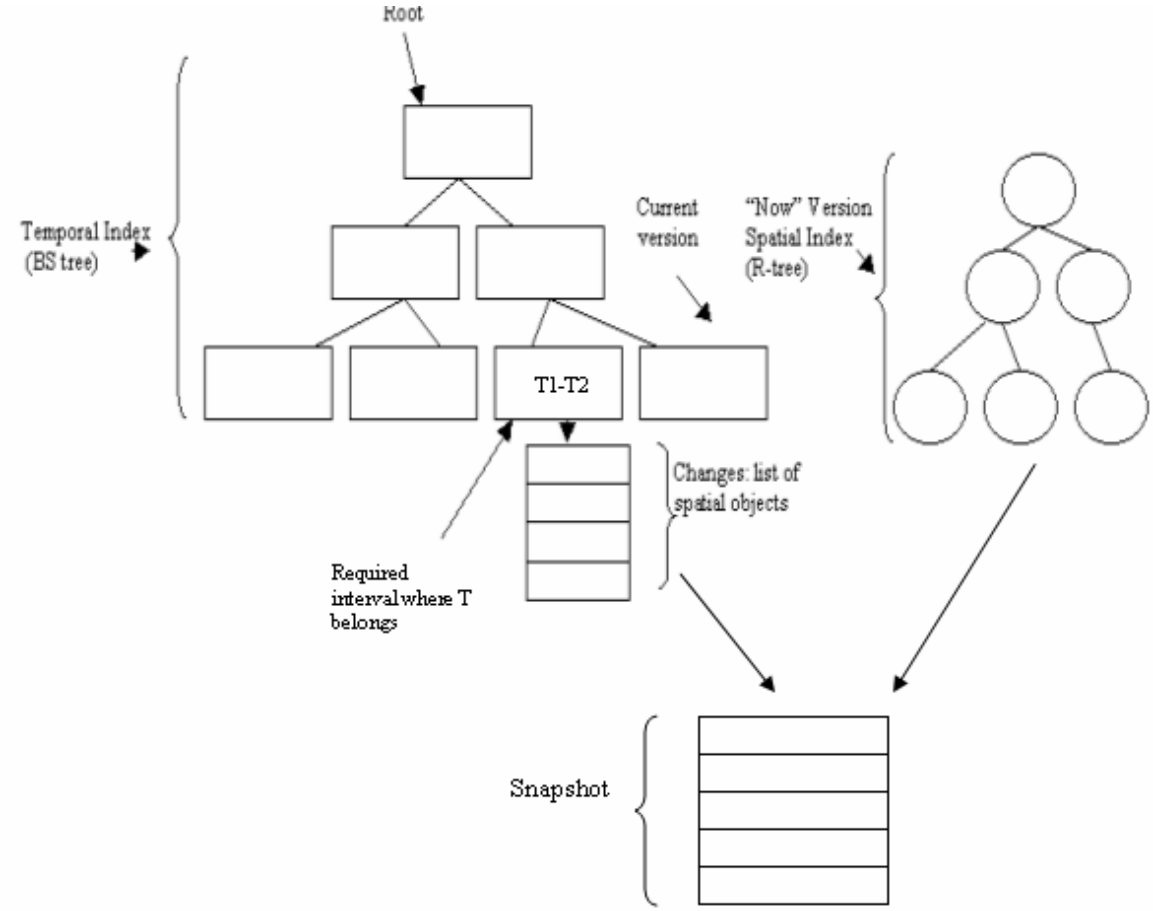

Figure 4.6: Building a snapshot at time T

4.7 Analysis and a Performance Study

This section includes a discussion of the following three main points:

- Analysis of the execution time of our algorithms
- A comparison of our temporal index with the snapshot index.
- A comparison of our spatial and temporal indexing structures with the PoTree.

4.7.1 Analysis of the Algorithms

This section will analyze the two algorithms that were proposed earlier in this chapter.

Analysis of Finding an Object at Time T

Finding an object at time T will require the following main tasks:

- Search temporal index (BS tree)
- If not found, search spatial index (R tree)

The time complexity of these tasks is O(log(i) + x + log(n)) where i is the number of intervals, x is the number of objects in the required interval and n is the number of objects in the spatial index (R-tree). Moreover, in discretely changing geographical systems, x could be considered as a constant. In this case the time complexity is O(log(i)+log(n)).

Analysis of Building a Snapshot at Time T

Building a snapshot at time T (different from “now”) will require the following tasks:

1. Get all objects in the spatial tree
2. Search the temporal index for the interval that satisfies T.
3. Update any object from task 1 which does not satisfy T using objects from task 2.

In the worst case, task 1 requires log(n) steps, task 2 requires log(i) steps, and task 3 requires x steps (where i is the number of intervals, x is the number of objects in the required interval, and n is the number of objects in the spatial index (R-tree)). As a result,

the time complexity of the whole process is O(log(i)+log(n)+x). In discretely changing applications x is usually a constant; hence, the complexity becomes O(log(i)+log(n)).

4.7.2 Time and Space Complexity Comparison with the Snapshot Index

Although the snapshot index deals only with temporal databases, we think it could be used to index spatial databases as well. In this section, we will focus on comparing our temporal index to the snapshot index. The indexes have a lot of similarities and have almost the same execution time. Using ordered balanced trees leads to a search time of log(n) (n = number of nodes). But we want to focus on two issues that could introduce some time latency or more space complexity.

First, the snapshot index has the procedure of copying what is called “alive” object when an acceptor page becomes “not useful.” This procedure could happen at any time, even at run time. On the other hand, the temporal index for AIRSTD has this process only when we are moving from one interval to another; and these tasks could be handled at some selected times and would not lead to run-time delays.

In addition, in the snapshot index there is a need to copy objects from one page to another page and keep the old objects. This issue does not really exist in the AIRSTD temporal index, and hence AIRSTD is better in terms of space complexity.

In conclusion, both indexing approaches are efficient and have the same time complexity; however, there is a slight difference when the complexity is considered.

4.7.3 Time and Space Complexity Comparison with the PoTree

This section presents a comparison analysis of the AIRSTD structure and the PoTree. We first go over the time complexities of searching the two structures, and later we provide a discussion of when each structure is better and why. We also give an idea about the space complexities in both approaches where the PoTree has a slight advantage.

Regarding searching a spatial object, we claim the following:

The two main tasks in the worst-case scenario of searching for a spatial object using the PoTree structure are:

- Search all objects in the spatial index (Kd tree)
- If the object is found, search the temporal index (B+ tree)

The time complexity of these tasks is O(log(n)+log(m)) where n is the number of objects in the spatial index (Kd tree) and m is the average number of objects in each node in the Kd tree.

On the other hand, the tasks using the AIRSTD structure are:

- Search temporal index (BS tree)
- If not found, search spatial index (R tree)

As analyzed before, the time complexity of these tasks is O(log(i)+ x +log(n)).

In best and average case scenarios, the PoTree still requires searching the spatial index, but now the object can be found directly by going to the current node. In this case, the time complexity stays O(log(n)).

On the other hand, using the AIRSTD structure requires that we search the temporal index only. This will require a time complexity of O(log(i)).

In the case of building a snapshot, the AIRSTD structure requires O(n) steps to get n objects from the R-tree, O(log(i)) to search the B-tree, and O(x) to retrieve the x objects in that interval. The total time will be O(n+ log(i) +x). Figure 4.6 illustrates this process.

On the other hand, the PoTree structure requires O(n * log(m)) to go over the n elements in the Kd tree and find the required object in each node with log(m) steps. Figure 4.7 illustrates the idea of how snapshots are built using the PoTree structure.

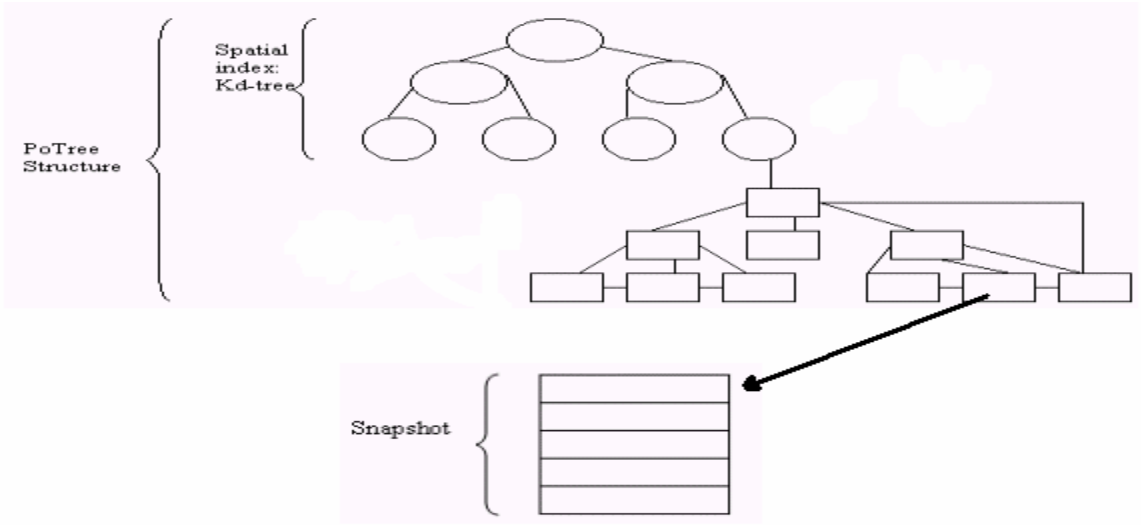

Figure 4.7: Building a snapshot using the PoTree

In order to analyze all these time complexities, we will consider two possible kinds of applications:

1- Data values are changing slowly and discretely (as in road networks or land parcels).
2- Data values are changing rapidly (as in continuous movements or volcanic real-time data)

Considering these two kinds of applications, we can summarize the time complexities that we have analyzed in this section in Table 4.1.

Structure	**Finding an object Best Case**	**Finding an object Worst Case**	**Updating or deleting an object**	**Building a snapshot**
PoTree	O(log(n))	O(log(n)+log(m))	O(log(n))	O(n*log(m))
AIRSTD Structure	O(log(i) + x)	O(log(i)+ x +log(n)).	O(log(n) + i)	O(n+ log(i) +x)

Table 4.1 shows the time complexities for PoTree and AIRSTD structure

From Table 4.1 we can conclude the following:

- In the first kind of applications:

- The AIRSTD Structure is better in the best-case scenario when finding a spatial object, since i is much less than n.
- The PoTree structure is better in the worst-case scenario, since we have the x component even though x is a constant.

- The PoTree is more efficient when updating or deleting tuples.
- The AIRSTD structure is better in building a snapshot.

-In the second kind of applications:

- In the best-case scenario, we can not conclude anything because both i and n could be large.
- The PoTree structure is better in the worst case scenario since the AIRSTD has the x component and also i>m
- The PoTree is more efficient when updating or deleting objects.
- The AIRSTD structure is better in building a snapshot.

The following are the space complexities of the two structures:

The PoTree needs O(m*n) where n is the original number of objects and m is the average number of changes of each object (as defined before).

The AIRSTD Structure needs O(n + x*i) where n is the original number of objects, i is the number of intervals and x is the number of objects in each interval.

Analyzing the space complexities leads us to conclude that the AIRSTD approach takes more space. The main reason for this is that when a tuple is updated, it has to be added to all old versions in which it was valid. However, the AIRSTD structure can use pointers to objects in order to reduce the increase of space complexity coming from old data duplication.

Moreover, the PoTree is more efficient when updating or deleting an object. The reason for that is because updating an object using the PoTree requires the following steps: (1) visit the node that has the object to be updated and (2) add a new tuple that contains the updated information in the temporal index. On the other hand, using the AIRSTD approach requires the following steps: (1) visit the node that has the object to be updated, (2) add a new tuple that contains the updated information and (3) record this update in all old versions that satisfy the time interval of the node before the update. Step 3 requires some time especially if we have a lot of intervals.

It is obvious that the AIRSTD structure is more efficient when building a snapshot at time T. The reason for that is when using the PoTree structure, we have to visit each node of the Kd-tree and get the object with the time interval that satisfies T. On the other hand, the AIRSTD structures need only to get all the objects in the R-tree and update those objects that do not satisfy T by others that exist in the version that satisfy T, which can be accessed using the temporal index.

As a result, we can see that the extra time we spend on the update process makes us save much more in the snapshot building process. This issue will lead to the following:

- AIRSTD is better when using discretely changing applications where snapshots are usually needed.
- PoTree is better in real-time applications where there are a lot of data values and spatial queries are usually needed.

4.8 Experimental Results

This section presents some experimental results of the algorithms. To achieve these results we have implemented a testing tool (using Java) that includes all the structures (temporal and spatial indexes), the rules and the algorithms we presented earlier.

The algorithms were tested with two groups (25000 and 50000 data) of data and two kinds of applications:

1. Discretely changing application with the following information:
    a. Interval = 3 months
    b. Total time = 25 years
    c. Rate of change = 0.5% per interval
2. Real time applications with the following information:
    a. Interval = 1 day
    b. Total time = 2 years
    c. Rate of change = 20% per interval

Moreover, we also tested the application with a constant number of changes (average of 10 changes per 1 month) with the 25000 data values. The reason for executing this test is to show that with constant changes (not a factor of n), the AIRSTD structure is better at some places as shown in Table 4.2.

The execution of these data has considered the following:

- Each recorded value is the average of 10 runs of the required method.

- The testing values or objects have been selected randomly except of the boundary values (the best and worst case scenarios).
- The updates of objects have been done randomly using a random function that selects randomly an object from the database.

Table 4.2 summarizes the results of testing a group of 25000 data values:

Structure/Case	**Execution Time Units Rate of change = 10 objects per 1 month Total Period = 25 years**	**Execution Time Units Rate of change = 0.5% per 3 months Total period = 25 years**	**Execution Time Units Rate of change = 20% per 1 day Total period = 2 years**
Searching PoTree/Best Case	15	12	14
Searching AIRSTD/Best Case	7	134	354
Searching PoTree/Worst Case	18	13	20
Searching AIRSTD/Worst Case	20	301	4201
PoTree/Snapshot	53,201	74,003	220,012
AIRSTD/Snapshot	25,008	25,031	29,007
PoTree/Update	7	8	9
AIRSTD/Update	173	92	233

Table 4.2: The execution times of 25000 discretely changing data values

Considering the second case and using 50000 data values we got the results summarized in Table 4.3:

Structure/Case	Execution Time Units Rate of change = 0.5% per 3 months Total period = 25 years	Execution Time Units Rate of change = 20% per 1 day Total period = 2 years
Searching PoTree/Best Case	13	14
Searching AIRSTD/Best Case	217	631
Searching PoTree/Worst Case	19	63
Searching AIRSTD/Worst Case	704	9391
PoTree/Snapshot	280,113	420,132
AIRSTD/Snapshot	50,332	53,104
PoTree/Update	14	12
AIRSTD/Update	103	414

Table 4.3: The execution times of 50000 rapidly changing data values

Our experimental results reflect the time complexities presented in Table 4.1 and the analysis provided earlier in this section. Table 4.4 presents predicted versus actual execution times.

Structure/Case	25000 data Interval = 3 months Total time = 25 years Rate of change = 0.5% per interval		25000 data Interval = 1day Total time = 2 years Rate of change = 20% per interval		50000 data Interval = 3 months Total time = 25 years Rate of change = 0.5% per interval		50000 data Interval = 1day Total time = 2 years Rate of change = 20% per interval	
	Predicted	***Actual***	***Predicted***	***Actual***	***Predicted***	***Actual***	***Predicted***	***Actual***
Searching PoTree/Best Case	15	12	15	14	16	13	16	14
Searching AIRSTD/Best Case	140	134	512	354	255	217	1013	631
Searching PoTree/Worst Case	17	13	25	20	19	19	26	63
Searching AIRSTD/Worst Case	518	301	5025	4201	1023	704	10026	9391
PoTree/ Snapshot	75,000	74,003	300,000	2,200,12	350,000	280,113	500,000	420,132
AIRSTD/ Snapshot	25,053	25,031	30,000	29,007	51,000	50,332	60,000	53,104
PoTree/Update	15	8	15	9	16	14	16	12
AIRSTD/Update	115	92	730	233	116	103	730	414

Table 4.4: Execution times: Predicted vs. Actual

4.9 Conclusions

In Chapter 4, we presented our main approach and algorithms for indexing and retrieving spatio-temporal data. We first went over the material that was discussed in Chapters 2 and 3, where we introduced our spatio-temporal model and our temporal index (that is, a time-ordered binary search tree). Second, we extended our indexing structure by adding an R-tree that is used for indexing and retrieving spatial objects that are valid "Now."

Using this combined structure (BS tree and R-Tree), called the AIRSTD structure, we provided two algorithms, which do the following:

- Retrieve Spatio-Temporal Data at Time T
- Build a Snapshot at Time T

Later in this chapter, we provided analysis of the time and space complexities of the proposed algorithms in addition to a performance study and comparison with other efficient structures. The main issues concluded from the analysis can be listed as follows:

- The AIRSTD structure relies on the idea of having a current full version, where old versions include only the changes of spatial objects with respect to the "Now" version. This way of handling spatial changes will lead to the following:
    - We can use the "Now" version directly without updating.
    - We can build a snapshot by having the "Now" version and the list of changes that we can get from the spatial index.
- On the other hand, the price of having the above positive points will be an increased amount of time to do an update.

These issues lead to two main conclusions:

1. The AIRSTD is efficient when most of the times queries require building snapshots and using the “Now” versions. This is the case for discrete spatio-temporal databases like road networks, lands, parcels, etc.
2. The AIRSTD is not as efficient as other structures when the application requires a lot of update queries. This case happens with real-time applications like moving objects, real-time remote sensing, etc.

Chapter 5: Summary and Conclusions

5.1 Summary

In this thesis, an approach for indexing and retrieving spatio-temporal databases, called AIRSTD, has been developed. Moreover, two algorithms were presented: (1) searching spatial objects at time T, and (2) building a snapshot at time T. In order to achieve our goals, we have followed these steps:

- Build a spatio-temporal database model.
- Design a temporal index called Changes Temporal Index (CTI).
- Design the AIRSTD structure using CTI and a well-known spatial indexing structure called an R-tree.
- Design the two algorithms that work using the AIRSTD structure.

The first step was achieved by using previous work in modeling spatio-temporal databases. We used geometrical abstract data types and added time attributes which give us the ability to store time, and hence to record changes of the spatial information.

In the second step, we considered a BS-tree structure (we can use B-tree's if needed) where nodes are time intervals. Each node (time interval) points to a list of spatial objects (or pointer to the spatial objects). These objects were valid during the time interval of the

node, but not in the current interval. Only the current interval includes the full database information.

In the third step, we considered spatial indexing, where we used the R-tree for this purpose. At this step, the current information is stored in the R-tree, and all the changes with respect to the current time (referred to by "Now") are stored in the temporal index. We called this combination the AIRSTD structure.

In the last step, we presented the two algorithms:

1. An algorithm for searching a spatial object at a given time T
2. An algorithm for building a snapshot at a given time T.

These algorithms use the AIRSTD structure.

In Chapter 4, we gave a performance analysis study of our approach. We first analyzed the time and space complexities of our algorithms, and later we discussed other approaches and compared them to our approach. We presented a chart showing the time complexities of some tasks using AIRSTD and the PoTree, which is another well-known approach used for similar purposes. Moreover we discussed some experimental results that are the result of testing two groups of data (25000 and 50000 spatio-temporal data values). Each was tested using a testing tool with different intervals, rates of change, and total periods of time. We have implemented (using Java) the testing tool in order to analyze the execution times using the AIRSTD and PoTree structures.

5.2 Conclusions

In this section we will present our point of view of our approach and highlight the positive and negative issues. Before listing these issues, we will provide some of the main conclusions we have discussed in Chapter 4:

- The AIRSTD structure is an indexing structure that uses two substructures: (1) a temporal index and (2) a spatial index.
- The AIRSTD structure uses the idea of having a "Now" version stored in the spatial structure and changes with respect to the "Now" versions that are stored in the temporal index.
- At each update of a current object, the old information about the object should be recorded in all old versions that satisfy the object's time interval.
- As a result, building a snapshot at a given time T will require two steps: (1) retrieve all the objects in the "Now" version and (2) exclude those objects that do not satisfy T, and put those changes recorded in the old version with a time interval satisfying T.

From our analysis of the work done in this thesis, we can conclude the following positive points about AIRSTD:

1. It is efficient in applications where spatial objects are changing discretely like road networks, lands, parcels, etc. The reason for this conclusion is that our approach is efficient when building snapshots is required, which is what discretely changing applications need most of the times.

2. AIRSTD is efficient when the current interval is considered. The main reason behind this conclusion is that we always keep our "Now" version ready and record the changes in the old versions.

Moreover, we can conclude the following negative point:

- It is not as efficient as other structures (for example: PoTree) with applications where data is rapidly changing, as in the case of moving objects and real time data. This conclusion comes from the extra time that we spend on updating which is the main process in real-time applications.

5.3 Future Work

The study and analysis done in this thesis lead to many conclusions that can be a source of future work in the field of retrieving and indexing spatio-temporal databases. The following are some issues that could be discussed in the future:

1. Find a way to index (spatially) the objects in old versions. One of the solutions could be an update of the R-Tree that can hold duplicates. This leads to efficient searching when these versions (stored in arrays) are considered.
2. Develop a smart structure, where we can record the most used intervals and keep their snapshots ready for use rather than building them every time.
3. Parts of the processes of some kinds of updates and deletions could be delayed to a later time when the user is not using the system. The reason for doing this is that we can save some time (avoid some delays) in the execution times.

4. Develop a structure that can work on a parallel machine network. Using parallel processing, it could be the case that machines on the network hold only some old versions in addition of the “Now” version.

References

[1] Abdelguerfi, M. and Givaudan, J. " Advances in Spatio-Temporal R-tree Based Structures", Technical report TR012-02, Computer Science Department, University of New Orleans, 2002.

[2] Abdelguerfi, Mahdi; Julie, Givaudan; Kevin, Shaw; Ladner, Roy. Spatio-Temporal Data Handling: The 2-3TR-tree, a Trajectory-oriented Index Structure for Fully Evolving Valid-time Spatio-Temporal Datasets. Proceedings of the 10th ACM International Symposium on Advances in Geographic Information Systems. November 2002.

[3] Al-Taha, K., et al. Bibliography on Spatio-Tempoal Databases. ACM SIGMOND Record, Vol. 22, pp. 59-67, 1994.

[4] Antonio, J. C.; and Güting, R. H. Dual Grid: A New Approach for Robust Spatial Algebra Implementation. CHOROCHRONOS Project, 2000.

[5] Berman, Lex , et al. Chaina GIS Project in Harvard University. www.people.fas.harvard.edu/~chgis, 2001.

[6] Bodolay, M. and Escobar-Molano, M. A Schema-less Spatio-Temporal Database System. SAC00 ACM, 2000.

[7] Cheng, T. S.and Gadia, S. A Pattern Matching for Spatio-Temporal Databases. ACM, 1994.

[8] Choi, Y. and Chung, C. Selectivity Estimation for Spatio-Temporal Queries to Moving Objects. ACM SIGMOD, 2002.

[9] Chomicki, J. and Revez, P. A Geometric Framework for Specifying Spatio-Temporal Objects. In Proc. 6th Intl. Workshop on Temporal Representation and Reasoning (TIME), pp. 41-46, 1999.

[10] Davis Jr., et al. Multiple Representations in GIS: Materialization through Map Generalization, Geometric and Spatial Analysis Operations. ACM GIS'99, pp. 60-65, Kansas City, MO, 1999.

[11] Dayal, G. and Wuu, T.J. A Uniform Approach to Processing Temporal Queries. In Proc. of the ACM 18th International Conference on Very Large Data Bases, 1992.

[12] Djafri, N. et al. Spatio-Temporal Evolution: Querying Patterns of Change in Databases. Proceedings of ACM GIS 2002 Conference, 2002.

[13] Egenhofer, J. Spatial SQL: A Query and Presentation Language. IEEE Transactions on Knowledge and Data Engineering, Vol. 6, pp. 86-95, 1994.

[14] Erwing, M. R., et al. Abstract and Discrete Modeling of Spatio-Temporal Data Types. ACM, 1998.

[15] Erwing, M. R., et al. Spatio-Temporal Types: An Approach to Modeling and Querying Moving Objects in Databases. GeoInformatica Vol. 3, No. 3, 1999.

[16] Gadia, S. K. A Seamless Generic Extension of SQL for Querying Temporal Data. Technical Report, Iowa State University, 1982.

[17] Garnett and Tansel, A. U. Equivalence of the Relational Algebra and Calculus for Nested Relations. Computers Math. Applic. Vol. 23 No. 10 pp. 3-25, 1992.

[18] Gotelo, J. A. and Güting, R. H. Dual Grid: A New Approach for Robust Spatial Algebra Implementation. GeoInformatica 6:1, 2000.

[19] Griffiths, T., et al. Triod: A Comprehensive System for the Management of Spatial and Aspatial Historical Objects. Proceeding of ACM Conference GIS'01. Atlanta, 2001.

[20] Griffiths, Tony, et al. Spatio-Temporal Databases: Tripod: A Comprehensive System for The Management of Spatial and Aspatial Historical Objects. Proceedings of the 9th ACM International Symposium on Advances in Geographic Information Systems. November 2001.

[21] Güting, R. H and Schneider, M. Realm-Based Spatial Data Types: ROSE Algebra. VLDB Journal, 4(2):100143, 1995.

[22] Güting, R. H. An Introduction to Spatial Databases. VLDB Journal Vol. 3, No. 4, 1994.

[23] Güting, R. H. et al. Implementation of the ROSE Algebra: Efficient Algorithms for Realm-Based Spatial Data Types. In Proc. of the 14th Intl. Symposium on Large Databases, pp. 216-239, Portland, Maine, August 1995.

[24] Güting, R. H. Gral: An Extensible Relational Database System For Geometric Applications. Proceeding of the 15th Conference on Very Large Databases, 1989.

[25] Guttman, A. R-tree: A Dynamic Index Structure for Spatial Searching. In Proc. of the 1984 ACM SIGMOND International Conference on Management Data, pp. 47-57, Boston, MA, 1984.

[26] Halaoui, Hatem. Spatio-Temporal Data Model: Data Model and Temporal Index Using Versions for Spatio-Temporal Databases. Proceedings of the GIS Planet 2005, Estoril, Portugal, May 2005.

[27] Homsby, K., Egenhofer, M.J., and Hayes, P. Modeling Cyclic Change. NCGIA and Department of Spatial Information Science and Engineering. University of Maine, 1999.

[28] Kollios, George N. Dissertation: Indexing Problems in Spatio-Temporal Databases. Polytechnic University (Brooklyn-Long Island-Westchester), 2000.

[29] Koubarakis, Manolis. Spatio-Temporal Databases: The CHOROCHRONOS Approach. Springer. Berlin, 2003.

[30] Lander, Roy; Kevin, Shaw; and Abdelguerfi, Mahdi (editors). Minig Spatio-Temporal Information Systems. Kluwer Academic Press. Berlin, 2002.

[31] Liou, Jaeik. Temporal Support for Land Information Systems in Object-Oriented Modeling. Dissertation at the School of Architecture, Surveying and Civil Engineering in the Royal Institute of Technology. Stockholm, Sweden, December 1999.

[32] Lorentoz, N.A. and Johnson, R.G. Extending Relational Algebra to Manipulate Temporal Data. ACM Transactions on Database Systems, Vol. 12, Issue 4, pp. 566 – 592, December 1987.

[33] Nascimento, M. Silva, and J. Towards Historical R-trees. In Proc. of ACM Symposium on Applied Computing, pp. 235-240, 1998.

[34] Noel, G. et al. The Po-Tree, a Real-Time Spatio-Temporal Data Indexing Structure. In Proc. of Development in Spatial Data Handling SDH2004, pp. 259-270, Leicester, August 2004.

[35] Noel, Guillaume, Servigne, Sylvie, and Laurini, Robert. Spatial and Temporal Information Structuring for Natural Risk Monitoring. Proceedings of the GIS Planet 2005, Estoril, Portugal, May 2005.

[36] Oracle Spatial Documentations. www.oracle.com.technology/products/spatial/htdocs. 2005.

[37] Papadimitriou, H., Suciu, D., and Vianu, V. Topological Queries in Spatial Databases, ACM PODS, pp. 81-92, 1996.

[38] Parent, C.; Spaccapietra, S., and Zimanyi, E. Spatio-Temporal, Conceptual Models: Data Structure + Space + Time. ACM GIS99, Kansas City, MO, 1999.

[39] Pfoser , D. and Jensen, C.S. Incremental Join of Time-Oriented Data. In Proc. of the 11th Intl. Conf. on SSDM, Cleveland, Ohio, July 28-30, 1999.

[40] Pfoser, D. and Tryfona, N. Requirements, Definitions and Notations for Spatial-Temporal Application Environments, ACM GIS, 1998.

[41] Pfoser, D., et al. Novel Approaches to the Indexing of Moving Object Trajectories. VLDB Journal, 2000.

[42] Procopiuc, C., Agarwal, P., and Har-Peled, S. Star-tree: An Efficient Self-Adjusting Index for Moving Points. ALENEX, 2000.

[43] Ravikanth, V., et al. Pro Oracle Spatial. Apress. Berkeley, 2004.

[44] Raza, A. and Kainz, W. Cell Tuple Based Spatio-Temporal Data Model: An Object Oriented Approach. Proceedings of the Seventh ACM International Symposium on Advances in Geographic Information Systems, 1999.

[45] Saltenis, S. and Jensen, C. Indexing of Moving Objects For Location-Based Services. In Proc. of ICDE, 2002.

[46] Saltenis, S., et al. Indexing the Positions of Continuously Moving Objects. In Proc. of ACM SIGMOD, 2000.

[47] Salzberg, B. and Tsotras, V. A Comparison of Access Methods for Temporal Data. ACM Computing Surveys 31(2), pp. 158-221, 1999.

[48] Schreck, Thomas and Chen, Zhengxin. R-Tree Implementation Using Branch-Grafting Method. In Proc. of 2000 ACM Symposium, March 2000.

[49] Shekhar, S. and Chawla, S. Spatial Databases: A Tour. Prentice Hall. Upper Saddle River, NJ, 2003.

[50] Snodgrass, R.T. The Temporal Query Language Tquel. ACM Trans. Database Systems, Vol. 12, No. 2, pp. 247-298, 1987.

[51] Tansel, A.U. A Historical Query Language. Information Sciences Journal. Vol. 53, pp.101-133, 1991.

[52] Tansel, A.U. Adding Time Dimension to Relational Model and Extending Relational Algebra, Information Systems Vol. 11, No. 4, pp. 343-355, 1986.

[53] Tansel, A.U. and Tin, E. The Expressive Power of Temporal Relational Query Languages. IEEE Transactions on Knowledge and Data Engineering, Vol. 9 No. 3, 1997.

[54] Tansel, A.U. Temporal Data Modeling and Integrity Constraints in Relational Databases. ISCIS 2004, pp. 459-469, 1993.

[55] Tansel, A.U. Temporal Relational Data Model. IEEE Transactions on Knowledge and Data Engineering, Vol. 9 No. 3, 1997.

[56] Theodoridis, Y. and Sellis, T. A Model for the Prediction of R-tree Performance. In Proc. of the15th Symposium on Principles of Database Systems (PODS), pp. 161-171, 1996.

[57] Theodoridis, Y., Sellis, Papadopoulos, T A., and Manolopoulos, Y. Specifications for Efficient Indexing in Spatio-Temporal Databases. In Proc. SSDBM, pp. 123-132, 1998.

[58] Tsotras ,V.J. and Kangelaris, N. The Snapshot Index, an I/O-Optimal Access Method for Time-slice Queries. Information Systems, 20(3), 1995.

[59] Tsotras, V.J., Jensen, C.S., and Snodgrass, R.T. An Extensible Notation for Spatio-Temporal Index Queries. ACM SIGMOND Record, pp. 47-53, March 1998.

[60] Worboys, M. A Unified Model for Spatial and Temporal Information. The Computer Journal, Vol. 37 No. 1, pp. 27-34, 1994.

[61] Worboys, M. F. A Generic Model for Planar Geographic Objects. International Journal of Geographical Information Systems, 6(5), pp. 353-372, 1992.

[62] Worboys, M.F. An Approach to Object Modeling of a “Navigable” Database. Proceedings of the NCGIA/CALTRANS Conference on Navigable Databases. NCGIA, UCSB, CA, 1996.

[63] Worboys, M.F. Object-oriented Approaches to Geo-Referenced Information. International Journal of Geographical Information Systems, 8(4): 385-399, 1994.

[64] Zeiler, Michael. Modeling our World. ESRI Press, New York, 1999.

[65] Zhang, J., et al. Location-Based Spatial Queries. In Proc. of ACM SIGMOD 2003.