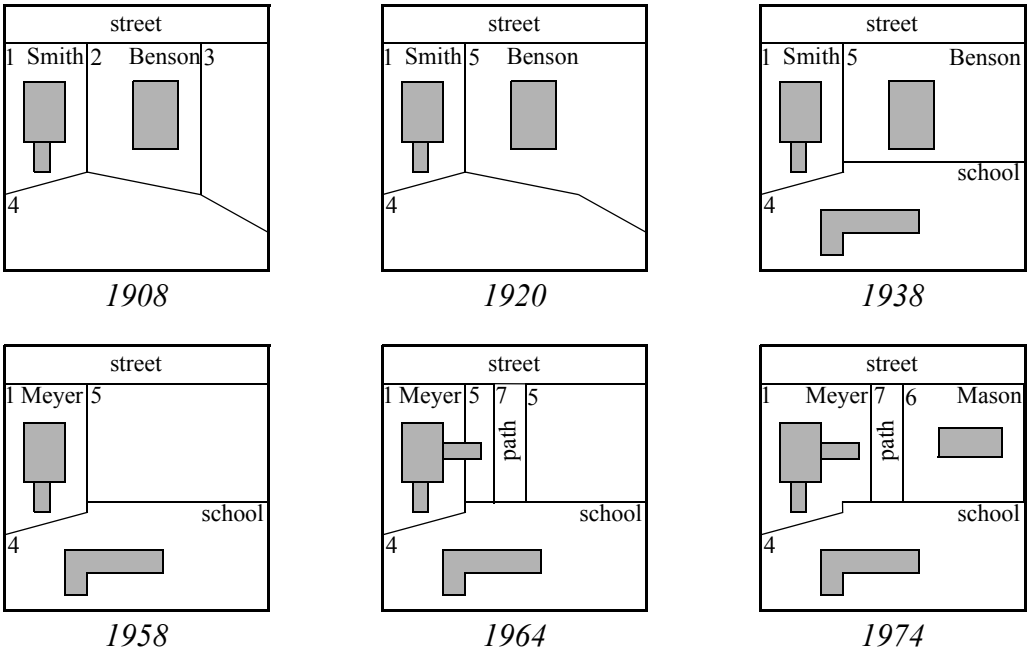
Moving Objects Databases

Unit 1:

Introduction - Spatio-Temporal Databases in the Past

Authors: Ralf Hartmut Güting, Markus Schneider



Preface

Dear student,

welcome to the course on “Moving Objects Databases”. We hope you will enjoy reading about this topic which has come up as a research issue not too long ago, roughly in 1996 or 1997. We, the authors, surely find it exciting, as it has been at the center of our research for the last few years.

The Topic

The general idea of moving objects databases is that we would like to be able to represent moving entities in databases and ask queries about them. Moving entities could be people, animals, all kinds of vehicles such as cars, trucks, air planes, ships, etc. For these examples, usually only the time-dependent position in space is relevant, not the extent, hence we can characterize them as *moving points*. However, there are also moving entities with an extent, for example, hurricanes, forest fires, oil spills, armies, epidemic diseases, and so forth. These we would characterize as *moving regions*.

Extending database technology to deal with such objects means - as for many other non-standard database applications - to provide facilities in a DBMS data model for describing such entities and to extend the query language by constructs for analyzing them, e.g. for formulating predicates about them. Second, it means that the implementation of a DBMS must be extended. The two major strategies for this are (i) to build a layer on top of an existing DBMS and so to map moving object representations and predicates to existing facilities of the DBMS, or (ii) to actually extend the DBMS by providing data structures for moving objects, methods for evaluating operations, specialized indexes and join algorithms, and so forth.

There are two major ways of looking at moving objects in databases: (i) to be interested in maintaining continuously information about the current position and predict near future positions, and (ii) to consider whole histories of movements to be stored in the database and to ask queries for any time in the past or possibly the future (if we allow “histories” to include the future). The course treats both perspectives in depth.

English

It is a bit unusual for a course at Fernuniversität Hagen to be in English, so why is that? As we said above, this is a new research area and when we started to write the course,

there did not yet exist books about it. Hence we planned to publish the course as a book, too. Since this should be accessible to a world-wide audience, it made sense to write it in English.

In the meantime, a book based on the course has indeed been published as follows:

> R.H. Güting and M. Schneider, Moving Objects Databases. Morgan Kaufmann Publishers, 2005.

The book has roughly the same contents as this course. Since you receive the course materials, it does not make a lot of sense for you to buy the book additionally. More details about the book and some additional material (such as slides for instructors) can be found at the book Web site

> http://www.informatik.fernuni-hagen.de/import/pi4/gueting/mod.html

We hope that you as mostly German-speaking students will nevertheless enjoy to participate in a course in English. It should be a good practice for you, as in computer science you need English anyway quite often, which you probably have noticed already.

To say this right away, even though also the assignments ("Einsendeaufgaben") and their solutions are formulated in English, it is fine if you write your solutions in German. If you write in German, they will be corrected in German. However, there is no knowledge of German required to participate in this course; you can also send your solutions in English and will get back English corrections.

Prerequisites

We assume that you are familiar with the general concepts of database systems, as for example given by the course 01665 "Datenbanksysteme" at the Fernuniversität. More detailed knowledge of the implementation of database systems, as presented in the course 01664 "Implementierungskonzepte für Datenbanksysteme" is helpful but not required.

Exercises and Assignments

As usual we recommend that you not only read the material but try to work actively yourself by solving the *Exercises* in the course text (corresponding to "Selbsttestaufgaben" in German) and by working on the *Assignments* ("Einsendeaufgaben").

Literature

The book based on this course is the first on the topic of moving objects databases, but of course, there exist many research articles. Each chapter provides bibliographic notes at the end, and every unit of the course ("Kurseinheit") has its own "Bibliography" section. The bibliographic notes at the end of Chapter 1 provide references to background literature on database systems in general as well as on spatial and on temporal databases.

Structure of the Course

The course 01675 "Moving Objects Databases" consists of seven units. It is possible to study only the first part consisting of the first four units; this is offered as a course 01676 "Moving Objects Databases I". If you have taken that course earlier, you can extend it to the full course by studying course 01677 "Moving Objects Databases II". Obviously, that course consists of the last three units of 01675.

The table of contents for the whole course is given below. Since we might still do minor changes and extensions to the various units (up to the deadline when they need to be sent to printing), it is possible that the table of contents may slightly change, in particular the page numbers. With the last unit we will provide a final version.

Version in the VU ("Virtuelle Universität")

The course will be available in PDF for registered students in the VU. That version is in colour and has active links (for cross references, index terms, table of contents). It may be more suitable for searching through the text, and also the drawings are due to the colours generally more beautiful than in the printed version.

Hagen/Gainesville

Prof. Dr. Ralf Hartmut Güting

Prof. Dr. Markus Schneider

About the Authors

Dr. Ralf Hartmut Güting is (since 1989) a full professor in computer science at the University of Hagen, Germany. He received his Diplom and Dr. rer. nat. degrees from the University of Dortmund in 1980 and 1983, respectively, and became a professor at that university in 1987. From 1981 until 1984 his main research area was Computational Geometry. After a one-year stay at the IBM Almaden Research Center in 1985, extensible and spatial database systems became his major research interests. His group has built a prototype of an extensible spatial DBMS, the Gral-System. He is the author of two (German) text books on data structures/algorithms and on compilers and has published about 50 articles in computational geometry and database systems. He is also an associate editor of the *ACM Transactions on Database Systems* and an editor of *GeoInformatica*. His major current research interests are extensible database architecture and moving objects databases.

Web site: http://www.informatik.fernuni-hagen.de/import/pi4/gueting/home.html

Dr. Markus Schneider received the Diploma degree in computer science from the University of Dortmund, Germany, in 1990, and the Dr. rer. nat. degree in computer science from the University of Hagen, Germany, in 1995. From 1996 to 2001 he worked as a reasearch assistant ("Hochschulassistent") at University of Hagen. Since January 2002 he is an Assistant Professor in the Department of Computer and Information Science and Engineering (CISE) at the University of Florida in Gainesville, Florida. His research interests were first related to the design and implementation of graphical user interfaces for spatial database systems. Since then, he has worked on the design and implementation of spatial data types (geo-relational algebra, ROSE algebra, realms). His current research interests are spatial, spatio-temporal and fuzzy spatial database systems.

Web site: http://www.cise.ufl.edu/~mschneid/

Contents of the Course

Unit 1

Unit 2

Unit 3

Unit 4

Unit 5

Unit 6

Unit 7

Contents of Unit 1

Teaching Goals

After completing your work on this unit, you should

- be able to explain some limitations of database systems with respect to non-standard applications
- be able to explain basic concepts of spatial database systems, in particular the concept of *spatial data types*
- be able to formulate some simple spatial queries, using SQL extended by SDTs
- be able to explain the basic ideas in temporal databases, such as
  - tuple or attribute time-stamping
  - valid time and transaction time
  - different kinds of temporal databases and relations
- be able to explain the five temporal data models described (Sarda, Segev, HRDM, Bhargava, BCDM)
- be able to construct a figure of bitemporal space based on a description of who knew what at what time.
- be able to write some simple temporal queries in TSQL2
- be able to give examples of moving objects and spatio-temporal data and explain how these data can be classified
- be able to describe several approaches to moving objects databases
- be able to explain the model of spatio-bitemporal objects, in particular
  - explain the terms bitemporal element, simplex, simplicial complex
  - explain ST-complexes and their operations
  - be able to formulate some simple queries in this model
- be able to describe the event-based model for spatio-temporal databases

Chapter 1

Introduction

The topic of this course is the extension of database technology to support the representation of moving objects in databases, termed *moving objects databases*. This is an exciting new research area that came up during the second half of the 1990s. Moving objects are basically geometries changing over time; hence this is a specific flavor of spatio-temporal databases which in turn have their roots in spatial databases, dealing with descriptions of geometry in databases, and temporal databases, addressing the development of data over time. The term "moving objects databases" emphasizes the fact that geometries may now change continuously, in contrast to earlier work on spatio-temporal databases that supported only discrete changes.

In this first chapter we provide some overview and background. We first review briefly the role of database management systems. This is followed by short introductions to the fields of spatial and temporal databases. We then describe the topic of this course in more depth, explaining different views of moving objects databases and describing classes of moving objects and applications.

1.1 Database Management Systems

Although we assume the reader to be familiar with the general concepts of database systems, let us briefly review their major aspects.

A *database management system* (*DBMS*) is a piece of software that manages a *database*, a repository of interrelated data items which are often central for the working of some enterprise or institution. A database is generally used by many diverse applications and multiple users each of which may need only a fraction of the data. One role of the database is to provide a single representation to all these applications, avoiding redundancies and possible inconsistencies that would occur if each application managed its data separately.

A DBMS provides to applications a high level *data model* and a related *query and data manipulation language*. The data model is a logical view of how data are organized which is generally very different from the way data are actually laid out on physical storage media. One of the most popular data models is the relational model which provides to users the view that data are organized in simple tables. The query language is based on the concepts offered in the data model. For example, in the relational model it is possible to derive new tables from given tables by selecting rows with certain properties, or a subset of the columns.

The separation between the logical view of data given in the data model and the actual physical representation is called the principle of *data independence*, one of the most fundamental contributions of DBMS technology. In the three level architecture for database systems, a widely accepted architectural model, data independence actually occurs at two different levels (Figure 1.1). Here the physical level describes how data are orga-

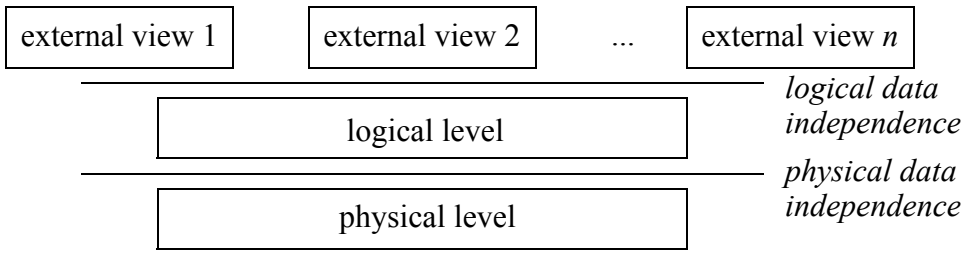


Figure 1.1: The three level architecture

nized on storage media, the logical level defines data in terms of the data model mentioned above, and the top level offers for each application its own view of a part of the data from the logical level, possibly transformed in some way. Physical data independence means that we can reorganize the physical organization of data without affecting the representation at the logical level, and logical data independence allows one to change the logical level to some extent without affecting the view of data of specific applications. It is the task of the DBMS to map efficiently between the levels. In particular, the *query optimizer* component needs to transform queries posed at the logical level into efficient access plans at the physical level.

Data in a database are a valuable resource and one major functionality of the DBMS is to protect data from being corrupted. To this end, changes to a database performed by an application are encapsulated within *transactions*; either all of the changes within a transaction are applied to the database, or none of them is applied, so that a transaction transforms a database from a consistent state to another consistent state. The DBMS manages concurrent access to the database by multiple applications and isolates them from each

other; changes performed within a transaction T become visible to all other applications only after transaction T is completed. The DBMS also keeps track of all physical changes performed during a transaction and is able to recover the consistent state before the transaction in most cases of failure, e.g. if the application software or even the DBMS itself, crashes, and even in many cases of hardware failure.

Other aspects of data protection are facilities in the data model to formulate *integrity constraints*, rules about certain relationships between data items that need to hold, and the management of *access rights* for various user groups.

The classical database management systems were conceived for relatively simple business applications. For example, the data types available for attribute types in the relational model are simple, basically integers or floating point numbers or short text strings. One goal of the database research of the last two decades has been to widen the scope so that as much as possible any kind of data used by any application can be managed within a DBMS, described by a high level data model and accessed by a powerful query language. For example, one would like to store images, geographic maps, music, videos, CAD documents, data from scientific experiments, meteorological measurements, etc. For all these kinds of data, one is interested in appropriate extensions of data model and query language so that any kind of question about these data can be formulated in a manner as simple as possible, and be answered efficiently (i.e. *fast*) by the DBMS. For example, we would like to retrieve images containing shapes similar to a given one ("find the images containing an air plane") or produce a map of the distribution of rainfall over some terrain.

With respect to the topic of this course, moving objects databases, we observe the following limitations of classical databases and the standard relational model.

1. We would like to represent geometric shapes such as the region belonging to a country. There is no reasonable way to do this, except for very simple objects such as points, for which the coordinates can be represented in numeric attributes.
2. We would like to represent the development of entities over time. But the data represented in a database generally reflect the current state of the world, there is no easy way to talk about the past.
3. We would like to represent objects moving around right now or in the past. For currently moving objects this would mean that positions are continuously updated which is not really feasible.

These limitations are addressed in the following three subsections.

1.2 Spatial Databases

The goal of spatial database research has been to extend DBMS data models and query languages to be able to represent and query geometries in a natural way. The implementation of a DBMS needs to be extended by corresponding data structures for geometric shapes, algorithms for performing geometric computions, indexing techniques for multi-dimensional space, and extensions of the optimizer (translation rules, cost functions) to map from the query language to the new geometry-related components.

The major motivation for studying spatial databases are geographic information systems (GIS). Early GIS systems made only limited use of DBMS technology, for example, by storing non-spatial data in a DBMS but managing geometries separately in files. However, spatial database technology has matured so that now all the major DBMS vendors (e.g. Oracle, IBM DB2, Informix) offer spatial extensions. Hence it is easier now to build GIS entirely as a layer on top of a DBMS, i.e., store all the data in the DBMS.

Whereas GIS have been the major driving force, spatial databases have a wider scope. Besides geographic space, there are other spaces of interest that may be represented in a database such as

- the layout of a VLSI design (often a large set of rectangles)
- a 3D model of the human body
- a protein structure studied in molecular biology

An important distinction concerns *image databases* and *spatial databases*. Although geographic space can be represented by images obtained by aerial photographs or satellites, the focus of spatial DBMS is to represent entities in space with a clearly defined location and extent. Image databases manage images as such. Of course, there exist connections. For example, feature extraction techniques may be used to identify within an image spatial entities that can be stored in a spatial database.

1.2.1 Modeling Spatial Concepts

What are the entities to be stored in a spatial database? Considering geographic space, obviously anything qualifies that might appear in a paper map, for example, cities, rivers, highway networks, landmarks, boundaries of countries, hospitals, subway stations, forests, corn fields, and so forth.

To model these diverse entities, one can offer concepts to model *single objects* and *spatially related collections of objects.*

For modeling single objects, three fundamental abstractions are *point*, *line*, and *region*. A *point* represents (the geometric aspect of) an object, for which only its location in space, but not its extent, is relevant. Examples of point objects are cities on a large scale map, landmarks, hospitals, or subway stations. A *line* (in this context always meaning a curve in space) is the basic abstraction for moving through space, or connections in space. Examples of line objects are rivers, highways, or telephone cables. Finally, a *region* is the abstraction for an entity having an extent in the 2D space. A region may in general have holes and consist of several disjoint pieces. Examples of region objects are countries, forests, or lakes. The three basic abstractions are illustrated in Figure 1.2.

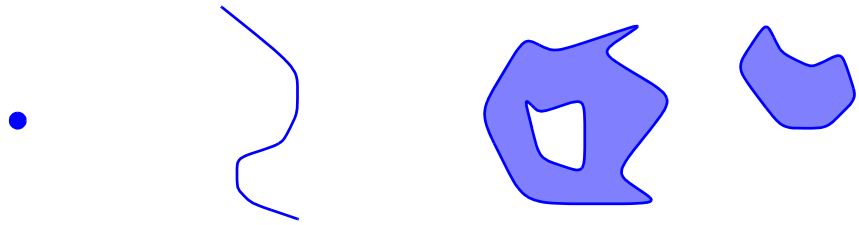


Figure 1.2: The three basic abstractions point, line, and region

The two most important instances of spatially related collections of objects are *partitions* (of the plane) and *networks*. A *partition* (Figure 1.3) can be viewed as a set of region objects that are required to be disjoint. The adjacency relationship is of particular interest, that is, there exist often pairs of region objects with a common boundary. Partitions can be used to represent so-called thematic maps.

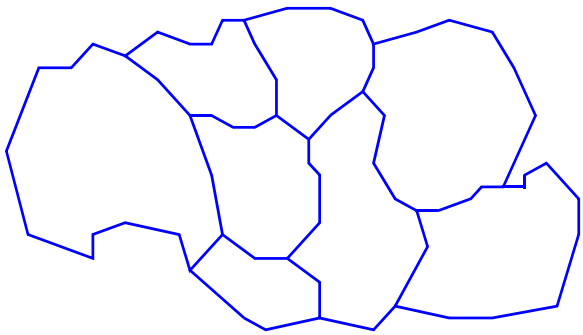


Figure 1.3: A partition

A *network* (Figure 1.4) can be viewed as a graph embedded into the plane, consisting of a set of point objects, forming its nodes, and a set of line objects describing the geometry of the edges. Networks are ubiquitous in geography, for example, highways, rivers, public transport, or power supply lines.

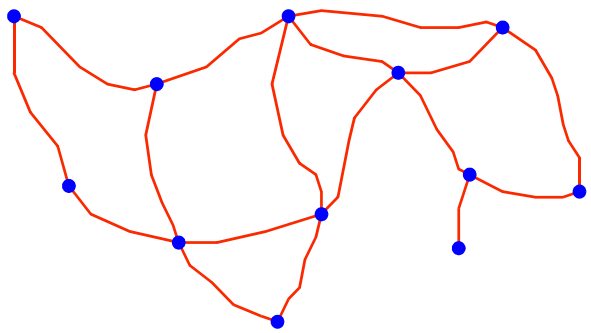


Figure 1.4: A network

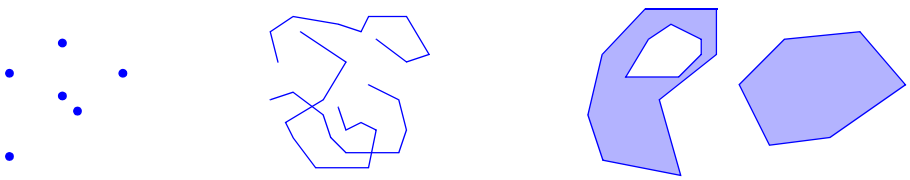
We have mentioned only the most fundamental abstractions to be supported in a spatial DBMS. For example, other interesting spatially related collections of objects are nested partitions (e.g. a country partitioned into provinces partitioned into districts etc.) or a digital terrain (elevation) model.

1.2.2 Extending Data Model and Query Language

We now consider how the basic abstractions can be embedded into a DBMS data model. For the single object abstractions point, line, and region, it is natural to introduce corresponding abstract data types, or *spatial data types* (*SDTs*). An SDT encapsulates the structure, e.g. of a region, with operations. These may be (i) predicates, e.g. testing whether two regions are adjacent or one is enclosed by the other, (ii) operations constructing new SDT values, e.g. forming the difference of two regions or the intersection of a line with a region, (iii) numeric operations such as computing the area of a region or the distance between a point and a line, or (iv) operations on sets of SDT values, e.g. aggregating a collection of regions into a single region, or finding in a collection of points the one closest to a query point.

A collection of spatial data types with related operations forms a *spatial algebra*. Important issues in the design of such algebras are closure under operations and completeness. The data types should be chosen carefully so that closure can be achieved. For example, the intersection of two line values yields in general a set of points[1], and the difference of two regions, even if each argument is a simple region without holes, may yield a region consisting of several disjoint components containing holes. An algebra with nice closure properties, the ROSE algebra, offers data types called *points*, *line*, and *region*[2] whose structure is illustrated in Figure 1.5. Here type *points* offers a set of points, type *line* a set

1. There may also be line values in the intersection, if there are overlapping parts of the argument lines. These will normally be returned by another operation.



a *points* value a *line* value a *region* value

Figure 1.5: The spatial data types *points*, *line*, and *region*

of polylines, and type *region* a set of polygons with holes. So one can offer operations such as

intersection:	*line* × *line*	→ *points*
minus:	*region* × *region*	→ *region*
contour:	*region*	→ *line*
sum:	*set*(*line*)	→ *line*
length:	*line*	→ *real*

Once spatial data types are defined, they can be embedded into a DBMS data model in the role of *attribute types*. Hence in addition to the standard types such as *int*, *real*, *string*, we may have spatial types *points*, *line*, and *region*. These types can be used in any kind of DBMS data model; it does not matter whether it is relational, object-oriented, or something else. In a relational setting we may have relations to represent cities, rivers, and countries, for example:

```
cities (name: string, population: int, location: points)
rivers (name: string, route: line)
highways (name: string, route: line)
states (name: string, area: region)
```

Queries can then be formulated by using SDT operations on spatial attributes within a standard query language such as SQL. Let us assume that predicates are available:

inside:	*points* × *region*	→ *bool*
adjacent:	*region* × *region*	→ *bool*

We can then formulate queries:

"What is the total population of cities in France?"

```
SELECT SUM(c.pop)
FROM cities AS c, states AS s
WHERE c.location inside s.area AND s.name = 'France'
```

2. Actually the names used for the second and third data type in the ROSE algebra are *lines* and *regions*. We rename them here to be consistent with later parts of the course.

"Return the part of the river Rhine that is within Germany."

```
SELECT intersection(r.route, s.area)
FROM rivers AS r, states AS s
WHERE r.name = 'Rhine' AND s.name = 'Germany'
```

"Make a list, showing for each country the number of its neighbour countries."

```
SELECT s.name, COUNT(*)
FROM states AS s, states AS t
WHERE s.area adjacent t.area
GROUP BY s.name
```

Exercise 1.1: Formulate the following queries, using SQL and data type operations. In each case, first define new SDT operations if necessary, and then write the query.

(a) How many people live within ten kilometers from the river Rhine? (Cities are modeled as points, hence if the point is within that distance we count the whole population.)
(b) With which of its neighbour countries does Germany have the longest common border?
(c) Find the locations of all bridges of highways crossing rivers. Return them as a relation with the name of the highway, the name of the river, and the location.

You may use the following notations in formulating queries.

Assignments. The construct `LET <name> = <query>` assigns the result of a query to a new object called *name* which can then be used in further steps of a query.

Multistep Queries. A query can be written as a list of assignments, separated by semicolons, followed by one or more query expressions. The latter are the result of the query.

Defining Derived Values. We assume that arbitrary ADT operations over new and old data types may occur anywhere in a WHERE clause, and can be used in a SELECT clause to produce new attributes, with the notation `<expression> AS <new attrname>`.

□

1.2.3 Implementation Strategy

To implement such a model, obviously one needs data structures for the types and algorithms implementing the operations. Moreover, one needs to support selection and join by spatial criteria. For selection, specialized index structures are needed. One popular candidate is the R-tree which organizes hierarchically a set of rectangles. The actual SDT values (e.g. *region*) are represented in such an index by their minimum bounding rectan-

gle (*MBR*, also called *bounding box*). To support spatial join, there are also specialized algorithms available some of which make use of spatial indexes.

To integrate these components into a DBMS, an extensible DBMS architecture is needed. The DBMS should offer interfaces to register components such as the following:

- data structures for the types
- algorithms for the data type operations
- spatial index structures with appropriate access methods
- spatial join methods
- cost functions for all methods, for use by the query optimizer
- statistics about the distribution of objects in space, needed for selectivity estimation
- extensions of the optimizer, e.g. in the form of translation rules
- registration of types and operations in the query language
- user interface extensions to handle presentation of spatial data, possibly input of spatial values for querying

Such extensible architectures have been investigated in research since about the mid-eighties. In the last years some of these capabilities have become available in commercial systems. In particular, extensibility by attribute data types and operations is well understood; one can add such an algebra as a *data blade*, *cartridge*, or *extender* in the various systems. Extensibility by index structures and extensions of the query optimizer are a much more thorny issue, but limited capabilities of this kind have also been realized.

1.3 Temporal Databases

1.3.1 Managing Time in Standard Databases

The databases managed by standard DBMS normally describe the current state of the world as far as it is known in the database. A change in the current state of the world will be reflected a bit later in some update to the database after which the previous state is lost.

Of course, for many (perhaps most) applications it is not sufficient to maintain just the current state; they need to keep track of some kind of history. In a standard DBMS this is possible if the application manages time itself, by adding explicit time attributes and performing the right kind of computations in queries. For example, suppose a company has an employee table of the form

```
employee (name: string, department: string, salary: int)
```

If the company wishes to keep track of previous departments and salaries for its employees, the table may be extended:

```
employee (name: string, department: string, salary: int, start:
date, end: date)
```

Standard DBMS offer a very limited support for this in the form of data types such as *date* or *time* (see below).

However, dealing with time in this form by the application is difficult, error-prone, leads to complex query formulations and often inefficient query execution. For example, in a join of two tables extended by time attributes as above, it is necessary to make sure that only tuples with overlapping time intervals are joined, by adding explicit conditions in the query. These conditions are several inequalities on the time attributes. Standard DBMS are often not very good at handling inequalities in query optimization (they focus more on equi-joins), hence an inefficient execution may result. In contrast, if true temporal support is built into the DBMS, this can be done automatically; no conditions are needed in the query, and execution will be tuned to perform this kind of join very efficiently.

Hence the goal of temporal database research has been to integrate temporal concepts deeply into the DBMS data model and query language and to extend the system accordingly to achieve efficient execution. We address the basic ideas for this in the sequel.

1.3.2 The Time Domain

First of all, let us consider how time itself can be modeled. Time is generally perceived as a one-dimensional space extending from the past to the future. There are some options:

The time space can be viewed as *bounded* or *infinite*. A bounded model assumes some origin and also an end of time.

Time can be viewed as *discrete*, *dense*, or *continuous*. Discrete models are isomorphic to the natural numbers or integers. Dense models are isomorphic to either the rationals or the reals: between any two instants of time another instant exists. Continuous models are isomorphic to the real numbers. Whereas most people will perceive time as being continuous, for practical reasons temporal database models often use discrete representations of time. In contrast, later in this course continuous models will be used, since this is more appropriate for dealing with moving objects.

In the continuous model, each real number corresponds to a "point in time"; in the discrete model, each natural number corresponds to an "atomic" time interval called a *chronon*. Consecutive chronons can be grouped into larger units called *granules* (e.g. hours, weeks, years).

One can also distinguish between *absolute* and *relative* time (also called *anchored* and *unanchored* time, respectively). For example, "January 22, 2002, 12pm" is an absolute time, and "three weeks" is a relative time.

These concepts of time can be captured in a number of data types:

- *instant*, a particular chronon on the time line in the discrete model, or a point on the time line in a continuous model.
- *period*, an anchored interval on the time line.
- *periods*, a set of disjoint anchored intervals on the time line, usually called a *temporal element* in the literature. We call the type *periods* to be consistent with later parts of the course.
- *interval*, a directed, unanchored duration of time. That is, a time interval of known length with unspecified start and end instants.

Some additional more "practical" data types, present in the SQL-92 standard, are

- *date*, a particular day from a year in the range 1 through 9999 AD
- *time*, a particular second within a range of 24 hours
- *timestamp*, a particular fraction of a second (usually a microsecond) of a particular day

1.3.3 Time Dimensions

We now turn to the semantics of the time domain. Whereas many different semantics can be thought of, the two most important "kinds" of time are the so-called *valid time* and *transaction time*. The *valid time* refers to the time in the real world when an event occurs, or a fact is valid. The *transaction time* refers to the time when a change is recorded in the database, or the time interval during which a particular state of the database exists.

In this context, standard databases are called *snapshot databases,* those dealing with valid time only are called *valid-time* or *historical databases*, those handling only transaction time *transaction-time* or *rollback databases*, and those treating both kinds of time *bitemporal databases*. The term *temporal database* refers to a model or system offering any kind of time support.

The various kinds of databases are illustrated in Figures 1.6 through 1.9. Figure 1.6 shows a simple standard relation with three tuples and three attributes, now called a snapshot relation.

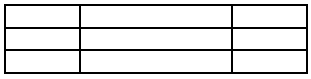


Figure 1.6: A snapshot relation

Figure 1.7 introduces the valid-time dimension. One can see that for each of the three tuples there are different versions for certain valid-time intervals in the past. Indeed, there is a fourth tuple that is not valid at the current time.

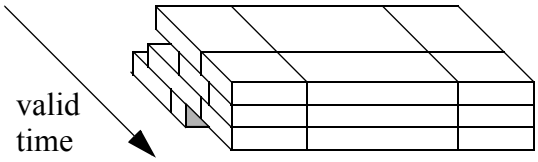


Figure 1.7: A valid-time relation

Figure 1.8 shows the transaction-time dimension. Here a first transaction has inserted three tuples into the relation. A second transaction has added a fourth tuple. Then, the third transaction has deleted the second and inserted yet another tuple.

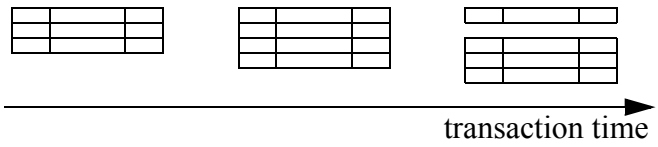


Figure 1.8: A transaction-time relation

Finally, Figure 1.9 shows a bitemporal relation. Here, an initial transaction creates two tuples valid from now on. The second transaction modifies the value of the second tuple and inserts a third one, also valid from now on. The third transaction deletes the second and the third tuple from the database (indicated by the gray shading, so these tuples are no longer valid). In addition it changes the start time of the second tuple (presumably the previous start time was wrong). The first tuple is still valid.

Note that what is represented in the figures is the content of the respective database at the current time. For example, in the transaction-time figures we can access all the previous states of the database.

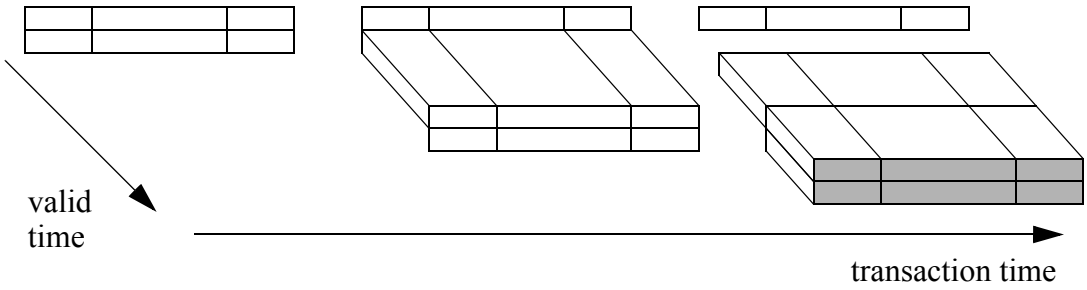


Figure 1.9: A bitemporal relation

1.3.4 Extending the Data Model

The question is now how time can be incorporated into the DBMS data model. The general approach in temporal databases has been to consider elements of the DBMS data model (e.g. tuples) as *facts* and to associate elements of the time domain with them to describe when facts are valid (*timestamps*). There are some choices:

- The data model extended: the most important choices are *relational* and *object-oriented* models.
- The granularity of facts: the most relevant are *tuples/objects* and *attributes*.
- The kind of timestamp used: a single chronon (*instant*), single time interval (*period*), set of time intervals = temporal element (*periods*).
- The time dimension: support of *valid time* or *transaction time* or *bitemporal*.

A vast number of data models has been proposed in the literature (around 40 according to (Zaniolo et al. 1997, Part II: Temporal Databases)) that can be classified along these criteria. We can show only a few of them in Table 1.1. Most of these models are relational. Some of them are mentioned only in one field of the table even though they do address both time dimensions. The name mentioned in the table is either the name of the model or of the author proposing it; details can be found in the bibliographic notes at the end of the chapter.

We now discuss a few representative models using a very simple example. The first model by *Segev* timestamps tuples with the instant when they became valid. The example in Table 1.2 describes the history of two employees Lisa and John working in different departments during a particular month, say, January 2002. On the 1st, Lisa started to work in the toys department. On the 8th, she moved to the books department. She returned to the toys department on January 14, and quit the company on January 16. John started to work on the 11th in the books department and still works there. In this model, a

		instant	*period*	*temporal element*
valid time	*timestamped attribute values*	Lorentzos	Tansel	HRDM
	timestamped tuples	Segev	Sarda	BCDM
transaction time	*timestamped attribute values*	Caruso		Bhargava
	timestamped tuples	Ariav	Postgres	BCDM

Table 1.1: Classification of temporal data models

Name	*Department*	*Time*
Lisa	Toys	1
Lisa	Books	8
Lisa	Toys	14
Lisa	Null	17
John	Books	11

Table 1.2: Model by Segev

separate tuple with null values in all non-key attributes is required to record termination of a valid time interval.

The next model by *Sarda* uses period time stamps. In this model the same information looks as shown in Table 1.3. Here null values are not needed any more. The symbol "∞" denotes "forever" in valid time, i.e., an end of the valid time period is not yet known.

Instead of tuples, it is also possible to timestamp attribute values. In the historical relational data model HRDM, attribute values are functions from time into some domain (Table 1.4). Here the whole employment history can be represented in two tuples, one for each value of the key attribute.

Exercise 1.2: Mr. Jones takes a trip from London to Edinburgh on the 5th of December where he stays at the Grand Hotel for three nights. On the 8th he decides that the Grand Hotel is too expensive and moves to a cheaper place called Traveler's Inn where he

Name	*Department*	*Time*
Lisa	Toys	[1-7]
Lisa	Books	[8-13]
Lisa	Toys	[14-16]
John	Books	[11-∞]

Table 1.3: Model by Sarda

Name	*Department*
1 → Lisa ... 16 → Lisa	1 → Toys ... 7 → Toys 8 → Books ... 13 → Books 14 → Toys ... 16 → Toys
11 → John 12 → John ...	11 → Books 12 → Books ...

Table 1.4: HRDM

spends a further week. On the 15th, after the business part of his trip is finished, he starts a short skiing vacation in the ski resort of Aviemore where he spends a weekend, staying at the Golf Hotel. On Sunday, the 17th of December, he goes back home.

In the meantime, his wife Anne finds it boring to stay at home alone, so on the 7th she visits her friend Linda in Brighton and stays with her for 5 days. On the 12th she goes back home. On the 16th she visits her parents and stays with them for a while. Today, on the 20th of December, she is still there.

Represent this information in the data models by Segev, Sarda, and the HRDM, starting on the 5th of December. □

These three models have dealt with valid time only. We extend our previous example to a bitemporal one by considering how information about Lisa and John was recorded in the database. This happened in the following transactions:

1. On the 6th of January, the administration was informed that Lisa had started to work in the toys department on the 1st and was going to work there until the 15th.
2. On the 10th it became known and entered into the database that Lisa had moved to the books department on the 8th. She was still expected to work until the 15th.
3. On the 12th it was decided that Lisa would move back to toys on the 14th and would stay there a while longer, until the 20th. Also it became known that a new employee John had started the day before in the books department.
4. On the 20th, it was entered that Lisa had actually quit the company on the 16th.

This is illustrated in a drawing of the bitemporal space in Figure 1.10. Here transaction time is on the horizontal and valid time on the vertical axis. The left part of the figure

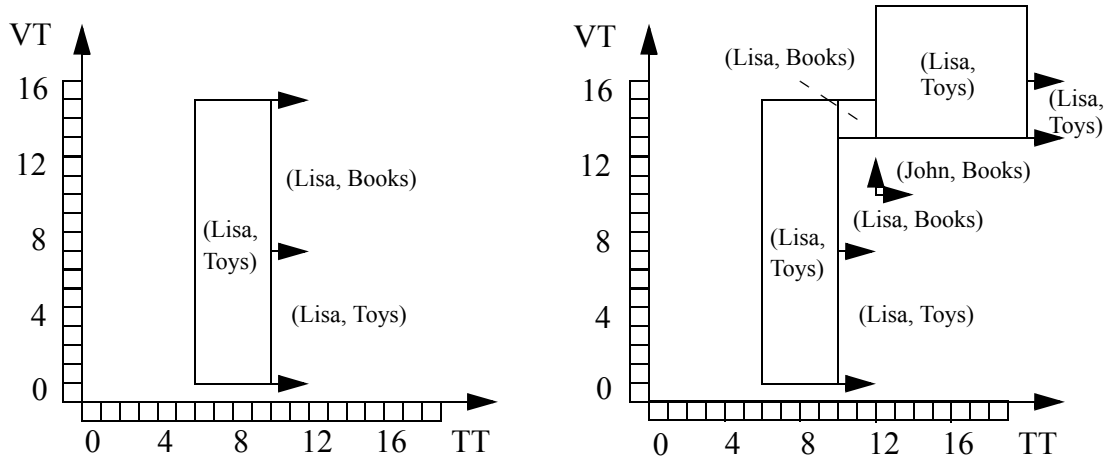


Figure 1.10: Bitemporal space

shows the state of the database after the second transaction, the right side the final state. An arrow to the right indicates that this information is valid w.r.t. transaction time "until changed". An upward arrow indicates an unknown end of interval w.r.t. the valid time. Note that by drawing a vertical line in such a diagram we can see what was known in the database at that particular time. For example, at the current time (say the 20th) we have the same information about employment history as in the valid time tables before.

The model by Bhargava is a bitemporal model using attribute value timestamping. A timestamp is a rectangle in the bitemporal space. Here our example (Figure 1.10 right) looks as shown in Table 1.5. The value *uc* ("until changed") denotes an open-ended interval in transaction time.

Name	*Department*
[6, 9] × [1, 15] Lisa [10, *uc*] × [1, 13] Lisa [10, 11] × [14, 15] Lisa [12, 19] × [14, 20] Lisa [20, *uc*] × [14, 16] Lisa	[6, 9] × [1, 15] Toys [10, *uc*] × [1, 7] Toys [10, *uc*] × [8, 13] Books [10, 11] × [14, 15] Books [12, 19] × [14, 20] Toys [20, *uc*] × [14, 16] Toys
[12, *uc*] × [11, ∞] John	[12, *uc*] × [11, ∞] Books

Table 1.5: Bhargava's model

Exercise 1.3: We extend the example from Exercise 1.2 by considering what Anne's mother Jennifer knew about the locations of her daughter and her son-in-law (of course she calls him by his first name, Keith). We start on December 1. On this day Jennifer assumed both of them to be at home as they had been before. Her knowledge was then changed by the following events:

1. On the 6th, Anne called her on the phone and told her that Keith had yesterday gone on a business trip to Edinburgh. He would stay there for two weeks. She herself was planning to visit Linda for a week, starting tomorrow.
2. On the 13th, Anne called again and told her that she was already back home since yesterday.
3. On the 16th, Anne arrived. What a pleasant surprise!
4. On the 19th she received a postcard by Keith from Aviemore, describing his skiing vacation. He wrote that he had arrived on Friday (yesterday), and would go home tomorrow.

Draw figures of the bitemporal space corresponding to Jennifer's knowledge, as of the 13th and as of the 19th of December. Draw separate figures for Keith and Anne, since otherwise figures get too crowded. □

The last model we mention here is the bitemporal conceptual data model BCDM. This model uses tuple timestamping. Timestamps are bitemporal elements which are finite sets of bitemporal chronons. No two value-equivalent tuples are allowed in a relation instance, hence the complete history of any given fact is represented in a single tuple. In this model, our example bitemporal database looks as shown in Table 1.6.

So the BCDM simply enumerates all the bitemporal chronons forming the bitemporal element of a tuple. This seems like an unnecessarily large representation. However, the purpose of the BCDM is not to determine an efficient representation but rather to have

Name	*Dept.*	*Time*
Lisa	Toys	{(6, 1), ..., (6, 15), ..., (9, 1), ..., (9, 15), (10, 1), ..., (10, 7), ..., (19, 1), ..., (19, 7), (*uc*, 1), ..., (*uc*, 7), (12, 14), ..., (12, 20), ..., (19, 14), ..., (19, 20), (*uc*, 14), ..., (*uc*, 16)}
Lisa	Books	{(10, 8), ..., (10, 13), ..., (19, 8), ..., (19, 13), (*uc*, 8), ..., (*uc*, 13), (10, 14), (10, 15), (11, 14), (11, 15)}
John	Books	{(12, 11), (12, 12), ..., (12, ∞), (13, 11), ..., (13, ∞), ..., (19, 11), ..., (19, ∞), (*uc*, 11), ..., (*uc*, ∞)}

Table 1.6: BCDM (at time 20)

simple semantics. The idea is that this model is then mapped in an implementation to some more space-efficient representation. For example, one can compute a minimal decomposition of the temporal element into rectangles, similar to Bhargava's model.

If you look at the translation from Figure 1.10 to Table 1.6 in detail, some questions come up. How is the translation of open-ended time intervals involving the symbols "∞" (in valid time) and "*uc*" (in transaction time) done? We have stated above that temporal elements are *finite* sets of chronons, so how can this be achieved?

The answer is as follows. The BCDM uses a *bounded* model of time. For valid time this is a set of chronons $\{t_1, ..., t_k\}$ where t_1 is the origin of time and t_k the end of time, assumed to lie in the past and the future, respectively. For transaction time it is the set of chronons $\{t'_1, ..., t'_l\} \cup \{uc\}$. A valid-time interval $[t_j, \infty]$ is therefore interpreted as a set of chronons $\{t_j, ..., t_k\}$. For transaction time, things are slightly more subtle. The value *uc* is assumed to move with the current time. At time $t'_m = now$ a transaction time interval $[t'_j, uc]$ is interpreted as the interval $[t'_j, ..., t'_{m-1}, uc]$. At every tick of the clock, the bitemporal elements in a relation instance are updated by adding new chronons for the current time. Therefore it is important to state in Table 1.6 that we consider the relation instance at time 20. For the tuple (John, Books), at this time the transaction time chronons are {12, 13, ..., 19, *uc*}. At time 21 they will be {12, 13, ..., 19, 20, *uc*}.

1.3.5 Extending the Query Language: TSQL2

As an example of a temporal query language we consider TSQL2 which is based on the BCDM data model. It was designed jointly by a committee of 18 researchers who had proposed temporal models and query languages earlier. TSQL2 is a superset of SQL-92 and has also been incorporated into the SQL3 standard.

In TSQL2 a bitemporal relation can be defined as follows. As a richer example, let us assume we wish to represent prescriptions in a doctor's database, recording for each patient which drugs were prescribed for which period of time. This can be done by a data definition command:

```
CREATE TABLE prescription (
  name char(30),
  drug char(30),
  dosage char(30),
  frequency interval minute)
AS VALID STATE DAY AND TRANSACTION
```

Here *name* is the name of the patient, *frequency* the number of minutes between drug administrations. The clause `as valid state day and transaction` says this is a *bitemporal state relation* where the granularity w.r.t. the valid time is one day. For the transaction time, the granularity is system dependent, something like milliseconds.

There are six different kinds of relations in TSQL2:

- snapshot relations
- valid-time state relations (specified: `as valid state`)
- valid-time event relations (`as valid event`)
- transaction-time relations (`as transaction`)
- bitemporal state relations (`as valid state and transaction`)
- bitemporal event relations (`as valid event and transaction`)

The difference between *state* and *event* relations is that a state relation records facts that are true over certain periods of time whereas an event relation records events that occurred at certain instants of time. Each tuple records a kind of event and is time-stamped with the instants when this event occurred. An event relation might record the days when a patient visited the doctor:

```
CREATE TABLE visit (
  name char(30))
AS VALID EVENT DAY AND TRANSACTION
```

Let us now formulate a few queries. First of all, it is possible to get an ordinary relation from a (bi)temporal relation by using the keyword `snapshot`.

"Who has ever been prescribed any drugs?"

```
SELECT SNAPSHOT name
FROM prescription
```

This returns an ordinary (snapshot) relation containing the names of all patients that ever were prescribed drugs.

In contrast, the normal behaviour of queries is to return the complete history with respect to valid time, assuming a version of the database (transaction time) as of *now*. In other words, the evaluation is based on our current knowledge of the past.

"Which drugs were prescribed to Lisa?"

```
SELECT drug
FROM prescription
WHERE name = 'Lisa'
```

will return a valid-time relation containing one tuple for each drug that Lisa was prescribed, associated with one or more maximal periods when Lisa was taking that drug. Note that in the prescription relation, after selecting for the name Lisa and the current time, there may be several tuple instances for a given drug, with different dosage and frequency values. These are all merged into a single tuple, joining their respective periods of valid time. This is an important operation in temporal databases called *coalescing*.

"Which drugs have been prescribed together with Aspirin?"

```
SELECT p1.name, p2.drug
FROM prescription AS p1, prescription AS p2
WHERE p1.drug = 'Aspirin' AND p2.drug <> 'Aspirin'
  AND p1.name = p2.name
```

Here the correlation variables `p1` and `p2` can be bound to pairs of tuples from `prescription`; it is automatically ensured that the valid time intervals of these tuples overlap. The result is a set of tuples containing the name of a patient and a drug, together with the maximal periods of time when both that drug and Aspirin were prescribed to the patient.

So far, the timestamp of result tuples was determined by the intersection of the timestamps of the argument time-stamp. This default can be overridden by a *valid-clause*:

"Which drugs was Lisa prescribed during 1999?"

```
SELECT p.drug
VALID INTERSECT(VALID(p), PERIOD '[1999]' DAY)
FROM prescription AS p
WHERE p.name = 'Lisa'
```

The `intersect` operation is applied to two intervals, namely the valid-time interval of the tuple and the year 1996, specified as an interval of days. Result tuples will have valid time intervals restricted to the time interval of that intersection.

We can also go back to some earlier state of the database:

"What did the physician believe on September 10, 1998, was Lisa's prescription history?"

```
SELECT drug
FROM prescription AS p
WHERE name = 'Lisa'
  AND TRANSACTION(p) OVERLAPS DATE '1998-09-10'
```

In fact, there is a default predicate on transaction time that was implicitly appended to all the earlier queries:

```
TRANSACTION(p) OVERLAPS CURRENT_TIMESTAMP
```

This may suffice to illustrate a few of the capabilities of a temporal query language like TSQL2. The language is powerful and quite complex queries are possible.

1.4 Moving Objects

The goal of research on moving objects databases is to extend database technology so that any kind of moving entity can be represented in a database and powerful query languages are available to formulate any kind of questions about such movements. In this section we look at the motivation for this research in more detail, and consider examples of moving objects and questions one might ask about them.

There are actually two different approaches leading to the idea of moving objects databases which can be described as the *location management* perspective and the *spatio-temporal data* perspective.

1.4.1 The Location Management Perspective

This approach considers the problem of managing the positions of a set of entities in a database, for example, the positions of all taxi-cabs in a city. At a given instant of time, this is no problem. We might have a relation with a taxi-ID as a key and attributes for x- and y-coordinates to record the position. However, taxis are moving around. To keep the location information up to date, for each taxi-cab the position has to be updated frequently. Here we encounter an unpleasant trade-off. If updates are sent and applied to the database very often, the error in location information in the database is kept small, yet the update load becomes very high. Indeed, for a large set of entities to keep track of, this is not feasible any more. Conversely, if updates are sent less frequently, the errors in the recorded positions relative to the actual positions become large.

This led to the idea of storing in the database for each moving object not the current position, but rather a *motion vector* which amounts to describing the position as a function of time. That is, if we record for an object its position at time t_0 together with its speed and direction at that time, we can derive expected positions for all times after t_0. Of course,

also motion vectors need to be updated from time to time, but much less frequently than positions.

Hence, from the location management perspective, one is interested in maintaining dynamically the locations of a set of currently moving objects, and be able to ask queries about the current positions, the positions in the near future, or any relationships that may develop between the moving entities and static geometries in the next time.

Note that from the point of view of temporal databases, what is stored in such a location management database is not a temporal database at all; it is a snapshot database maintaining the current state of the world. No history of movement is kept. We will consider moving objects databases based on the location management perspective in Chapter 3.

1.4.2 The Spatio-Temporal Data Perspective

Here the approach is to consider the various kinds of data that might be stored in a (static) spatial database and to observe that clearly such data may change over time. We wish to describe in the database not only the current state of the spatial data, but rather the whole history of this development. We would like to be able to go back in time to any particular instant and to retrieve the state at that time. Moreover, we would like to understand how things changed, analyze when certain relationships were fulfilled, and so forth.

Two basic questions come up:

1. What kinds of data are stored in spatial databases?
2. What kinds of change may occur?

For the first question, in Section 1.2.1 we have seen that spatial databases support abstractions for single objects such as *point*, *line*, or *region*, as well as spatially related collections of objects among which *networks* and *partitions* are the most relevant.

Regarding kinds of change, a major distinction concerns *discrete changes* and *continuous changes*.

Classical research on spatio-temporal databases has focused on discrete changes for all the spatial entities mentioned above. In contrast, continuous changes are the topic of this course, and this is what is usually meant by the term “moving object”.

Whereas discrete changes occur on any kind of spatial entity, continuous changes seem most relevant for *point* and *region*.[3] Hence, a *moving point* is the basic abstraction of a physical object moving around in the plane or a higher-dimensional space, for which only the position, but not the extent, is relevant. The *moving region* abstraction describes

an entity in the plane that changes its position as well as its extent and shape, i.e., a moving region may not only move, but also grow and shrink.

1.4.3 Moving Objects and Questions About Them

Let us look at some examples of moving entities and possible questions about them. We consider moving points (Table 1.7) and moving regions (Table 1.8). With the exception

Moving Point Entities	*Questions*
People: politicians, terrorists, criminals	• When did Bush meet Arafat? • Show the trajectory of Lee Harvey Oswald on November 22, 1963.
Animals	• Determine trajectories of birds, whales, ... • Which distance do they traverse, at which speed? How often do they stop? • Where are the whales now? • Did their habitats move in the last 20 years?
Satellites, spacecraft, planets	• Which satellites will get close to the route of this spacecraft within the next 4 hours?
Cars: taxi-cabs, trucks	• Which taxi is closest to a passenger request position? • Which routes are used regularly by trucks? • Did the trucks with dangerous goods come close to a high risk facility?
Air planes	• Were any two planes close to a collision? • Are two planes heading towards each other (going to crash)? • Did planes cross the air territory of state X? • At what speed does this plane move? What is its top speed? • Did Iraqi planes pass the 39th degree?
Ships	• Are any ships heading towards shallow areas? • Find “strange” movements of ships indicating illegal dumping of waste
Military vehicles: rockets, missiles, tanks, submarines	• All kinds of military analyses

Table 1.7: Moving Points and Questions

of countries, all of them change continuously. Whether they have been or can be observed continuously is a different issue discussed later.

3. It seems much harder to think of examples of continuously moving lines, networks, or partitions, although such examples can certainly be found.

Moving Region Entities	*Questions*
Countries	• What was the largest extent ever of the Roman empire? • On which occasions did any two states merge? (e.g. reunification) • Which states split into two or more parts? • How did the Serb-occupied areas in former Yugoslavia develop over time? When was the maximal extent reached?
Forests, lakes	• How fast is the Amazon rain forest shrinking? • Is the dead sea shrinking? • What is the minimal and maximal extent of river X during the year?
Glaciers	• Does the polar ice cap grow? Does it move? • Where must glacier X have been at time Y (backward projection)?
Storms	• Where is the tornado heading? When will it reach Florida?
High / low pressure areas	• Where do they go? Where will they be tomorrow?
Scalar functions over space, e.g. temperature	• Where has the 0-degree boundary been last midnight?
People	• Movements of the celts in the second century B.C.
Troops, armies	• Hannibal traversing the Alps. Show his trajectory. When did he pass village X?
Cancer	• Can we find in a series of X-ray images a growing cancer? How fast does it grow? How big was it on June 1, 1995?
Continents	• History of continental shift
Diseases	• Show the area affected by mad cow disease for every month in 1998.
Oil spills	• Which parts of the coast will be touched tomorrow?

Table 1.8: Moving Regions and Questions

Clearly there exist many kinds of interesting moving entities and one can ask questions about them ranging from simple to very complex. The goal of moving object database research is to design models and languages that allow one to formulate these questions in a simple yet precise way.

1.4.4 A Classification of Spatio-Temporal Data

In Tables 1.7 and 1.8 we have emphasized entities capable of continuous movement. Nevertheless, there exist also many applications involving spatial data that change only

in discrete steps. To understand the scope of the more traditional spatio-temporal database research let us introduce a classification of time-dependent point and region data.

Spatio-temporal data can be viewed in a natural way as being embedded in a space that is the cross-product of the original spatial domain and of time. Here we consider 2D space and restrict attention to a single time dimension, namely valid time. Hence data "live" in a 3D space, as illustrated in Figure 1.11.

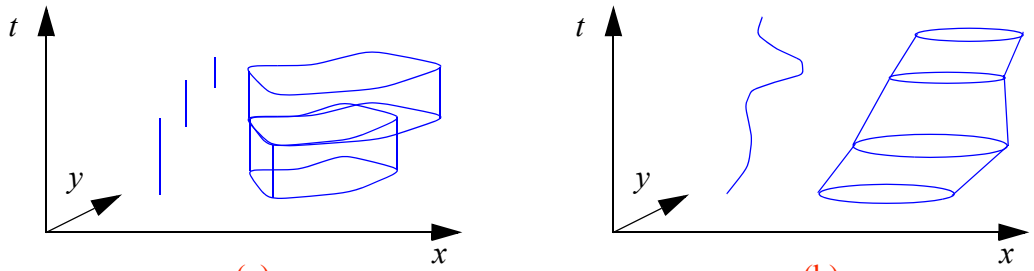


Figure 1.11: (a) Discretely changing point and region (b) Continuously changing point and region

We now characterize application data with respect to their "shape" in this 3D space, obtaining the following categories:

1. *Events in space and time – (point, instant)*. Examples are archeological discoveries, plane crashes, volcano eruptions, earthquakes (at a large scale where the duration is not relevant).
2. *Locations valid for a certain period of time – (point, period)*. Examples are: cities built at some time, still existing or destroyed; construction sites (e.g. of buildings, highways); branches, offices, plants, or stores of a company; coal mines, oil wells, being used for some time; or "immovables", anything that is built at some place and later destroyed.
3. *Set of location events – sequence of (point, instant)*. Entities of class (1) when viewed collectively. For example, the volcano eruptions of the last year.
4. *Stepwise constant locations – sequence of (point, period)*. Examples are: the capital of a country; the headquarter of a company; the accomodations of a traveler during a trip; the trip of an email message (assuming transfer times between nodes are zero).
5. *Moving entities – moving point*. Examples are people, planes, cars, etc., see Table 1.7.
6. *Region events in space and time – (region, instant)*. E.g., a forest fire at large scale.

7. *Regions valid for some period of time – (region, period).* For example, the area closed for a certain time after a traffic accident.
8. *Set of region events – sequence of (region, instant).* For example, the Olympic games viewed collectively, at a large scale.
9. *Stepwise constant regions – sequence of (region, period).* For example, countries, real estate (changes of shape only through legal acts), agricultural land use, etc.
10. *Moving entities with extent – moving region.* For example, forests (growth); forest fires at small scale (i.e. we describe the development); people in history; see Table 1.8.

These classes of data will be useful to characterize the scope of two approaches to spatio-temporal modeling which are described next.

1.4.5 Temporal Databases With Spatial Data Types

A straightforward idea to deal with spatio-temporal applications is the following: Use any temporal DBMS with its system-maintained tuple timestamps and enhance it by spatial data types. For example, assuming the TSQL syntax from Section 1.3.5 and the spatial data types from Section 1.2.2, we might create a table for real estate:

```
CREATE TABLE real_estate (
  owner char(30),
  area region)
AS VALID STATE DAY
```

Such a table would manage discretely changing regions as shown in Figure 1.11 (a). We can ask queries combining the features of the temporal query language with operations on spatial data types:

"Show the properties adjacent to the property of Charles Smith as of March 17, 1977."

```
SELECT r2.area
FROM real_estate AS r1, real_estate AS r2
WHERE r1.owner = 'Charles Smith' AND
  VALID(r1) OVERLAPS DATE '[1977-03-17]' AND
  r1.area ADJACENT r2.area
```

This is something like the cross-product of spatial and temporal databases. Capabilities of spatial and temporal systems are combined without any specific integration effort. This approach is natural and it appears that its technology is already well-understood, since techniques from both fields can be used. Considering the classification of Section 1.4.4, this approach can support classes (1) through (4) and (6) through (9) of spatio-temporal data. However, it cannot deal with moving objects, i.e., classes (5) and (10).

In Chapter 2 we consider this approach in more detail, studying two representative data models.

1.4.6 Spatio-Temporal Data Types

An alternative idea is to extend the strategy used in spatial databases to offer abstract data types with suitable operations: In this case we offer *spatio-temporal data types* such as moving point (type *mpoint* for short) or moving region (*mregion*). A value of such a data type captures the temporal development of a point or a region over time. Hence, a value of type *mpoint* is a continuous function f: *instant* → *point*, and a value of type *mregion* is a continuous function g: *instant* → *region*. Geometrically, such values correspond to the 3D shapes shown in Figure 1.11 (b). As in spatial databases, such types can be embedded into relational or other DBMS data models. Here we can describe the real estate data of the previous subsection as

```
real_estate (owner: char(30), area: mregion)
```

This is possible since, of course, a data type capable of describing continuous movement can also describe discrete changes. However with this approach it is also possible to describe truly continuous changes and have relations describing the movements of air planes or storms:

```
flight (id: string, from: string, to: string, route: mpoint)
weather (id: string, kind: string, area: mregion)
```

The data types include suitable operations such as:

intersection:	*mpoint* × *mregion*	→ *mpoint*
distance:	*mpoint* × *mpoint*	→ *mreal*
trajectory:	*mpoint*	→ *line*
deftime:	*mpoint*	→ *periods*
length:	*line*	→ *real*
min:	*mreal*	→ *real*

One discovers quickly that in addition to the main types of interest, *mpoint* and *mregion*, related spatial and temporal as well as other time-dependent types are needed. For example, the distance between two moving points is a real valued function of time, captured here in a data type *mreal* (“moving real”). The operations above have the following meaning: **Intersection** returns the part of a moving point whenever it lies inside a moving region which is a moving point (*mpoint*) again. **Distance** was mentioned already above. **Trajectory** projects a moving point into the plane, yielding a *line* value. **Deftime** returns the set of time intervals when a moving point is defined, a *periods* value, as intro-

duced in Section 1.3.2. **Length** returns the length of a *line* value, and **min** yields the minimal value assumed over time by a moving real.

Given such operations, we may formulate queries:

"Find all flights from Düsseldorf that are longer than 5000 kms."

```
SELECT id
FROM flights
WHERE from = 'DUS' AND length(trajectory(route)) > 5000
```

"Retrieve any pairs of air planes that during their flight came closer to each other than 500 meters!"

```
SELECT f.id, g.id
FROM flights AS f, flights AS g
WHERE f.id <> g.id AND min(distance(f.route, g.route)) < 0.5
```

"At what times was flight BA488 within the snow storm with id S16?"

```
SELECT deftime(intersection(f.route, w.area))
FROM flights AS f, weather AS w
WHERE f.id = 'BA488' AND w.id = 'S16'
```

Clearly, the approach using spatio-temporal data types can manage continuous as well as discrete changes and support all 10 classes of spatio-temporal data discussed in Section 1.4.4. On the other hand, time is not managed "automatically" by the system as in the temporal databases of Section 1.3. We will consider this approach in more detail in Chapter 4.

1.5 Bibliographic Notes

In Section 1.1 we have briefly discussed database systems in general. Of course, there exist many good books of which we can mention only a few. Some good English books are (Garcia-Molina et al. 2002, Elmasri and Navathe 2003, Kifer et al. 2005). If you wish to read in German, we recommend (Vossen 2000, Kemper and Eickler 1999) and especially regarding the implementation of database systems (Härder and Rahm 1999).

A very good book on spatial databases has appeared recently (Rigaux et al. 2002); other good books include (Shekhar and Chawla 2003, Worboys and Duckham 2004, Laurini and Thompson 1992). Section 1.2 is based on the survey article (Güting 1994). The ROSE algebra and its implementation is described in (Güting and Schneider 1995, Güting et al. 1995). Spatial database technology as available in the Oracle System is described in (Kothuri et al. 2004).

A good book that summarizes many of the research results in temporal databases up to 1993 is (Tansel et al. 1993). A nice, shorter introduction to temporal databases, on which Section 1.3 is based, can be found in (Zaniolo et al. 1997, Part II: Temporal Databases). The language TSQL2 is described in detail in (Snodgrass 1995). There exists also a survey article on temporal databases (Özsoyoglu and Snodgrass 1995).

A classical paper that established the distinction between valid time and transaction time is (Snodgrass and Ahn 1986). The data models by Segev and Sarda are described in (Segev and Shoshani 1987) and (Sarda 1990), respectively. The HRDM model is presented in (Clifford and Croker 1987). Bhargava's model can be found in (Bhargava and Gadia 1993). The bitemporal conceptual data model BCDM is the model underlying TSQL2; it is described in detail in (Snodgrass 1995, Chapter 10). Details on the other models mentioned in Table 1.1 can be found also in (Snodgrass 1995, Chapter 10) or in (Özsoyoglu and Snodgrass 1995).

Section 1.4, introducing the basic ideas of moving objects databases, is based on papers by Wolfson and colleagues, e.g. (Wolfson et al. 1998) for the location management perspective and by Güting and colleagues, e.g. (Erwig et al. 1999) for the spatio-temporal data type perspective. Later chapters will examine these approaches in more depth and provide further references.

There exists an edited book covering many aspects of moving objects databases (Koubarakis et al. 2003). It summarizes the results of the CHOROCHRONOS project, in which also the authors participated.

Another book related to moving objects is (Schiller and Voisard 2004) which focuses on location-based services, that is, services depending on the current location of a user. Some chapters of the book do also address database issues; others provide case studies of applications, or the technology of capturing position data, e.g. by GPS.

Chapter 2

Spatio-Temporal Databases in the Past

Classical research on spatio-temporal databases has focused on *discrete* changes of spatial objects over time. Applications here in mind predominantly have a "man-made" nature. For example, cadastral applications deal with the management of land parcels whose boundaries can change from time to time due to specific legal actions like splitting, merging, or land consolidation. Political boundaries suddenly disappear as the reunification of West and East Germany shows. Road networks are extended by new streets.

In the following, we give two representative examples of early spatio-temporal models dealing with this kind of application. In the sense of Section 1.3.5, the first model leads to bitemporal state relations while the second model supports bitemporal event relations.

2.1 Spatio-Bitemporal Objects

The basic idea of the first model to be presented is to provide a unified approach for spatial and temporal information. This is achieved by combining concepts of purely spatial modeling based on so-called two-dimensional simplicial complexes with concepts of purely temporal modeling incorporating the two orthogonal time dimensions of transaction and valid time (Section 1.3.3).

2.1.1 An Application Scenario

As a motivating example, let us consider a highly simplified and fictitious scenario of land ownership.[1] Information related to land ownership usually comprises spatial, temporal, legal, and other aspects. Spatial aspects refer to the geometry of land parcels. Temporal aspects relate to the duration of ownership. Legal and other aspects of ownership are affected by contracts, death and inheritance, legal proceedings, fire, etc. Figure 2.1

1. The scenario is taken from a very similar description in (Worboys 1994, Section 2.3). Used with permission of the author.

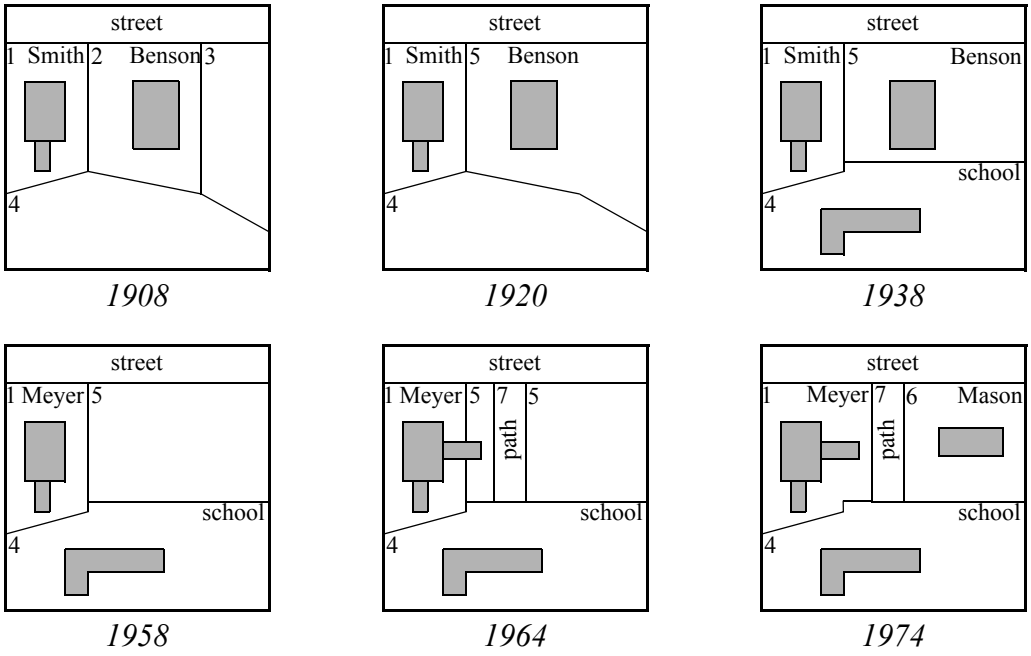


Figure 2.1: Spatio-temporal change of land ownership
[Worboys 1994 Fig. 3, used with permission.]

shows the spatial and ownership variation of a land area through some decades of the century. The chronology of these changes is as follows:

- *1908*: The land area under consideration consists of a street, a land parcel owned by Bill Smith (parcel 1), a land parcel owned by Jane Benson (parcel 2), and further parcels.
- *1920*: Jane Benson has bought parcel 3, which together with her old parcel 2 is now named parcel 5.
- *1938*: Jane Benson has sold a part of her parcel in favor of the construction of a new school on parcel 4.
- *1958*: Jane Benson's house has been destroyed by fire, and she has died. Jack Meyer now owns the land and the buildings of parcel 1.
- *1960*: The council announces to build a path through parcel 5 in 1962 in order to give better access to the school.
- *1962*: The construction of the path on the new parcel 7 is postponed until 1964.
- *1964*: Jack Meyer has built an extension which partly trespasses on parcel 5. The council has built the path through parcel 5.
- *1974*: Jack Meyer has included a part of parcel 5 by illegal possession into his ownership. Jill Mason has bought the remaining part of parcel 5, which is now named parcel 6, and built a house on it.

This scenario incorporates both transaction time and valid time. For example, in 1962 (transaction time) the information is available that the path, originally forecast in 1960 (transaction time) to be built in 1962 (valid time), is postponed until 1964 (valid time).

2.1.2 Bitemporal Elements

Formally, the model is based on bitemporal elements (Section 1.3.4) and simplicial complexes. With bitemporal elements, transaction times and valid times are measured along two orthogonal time axes. We assume that the domain T_V of valid time contains the special elements -∞ and ∞ indicating the indefinite past (or initial state) and indefinite future (or final state), respectively. T_T shall denote transaction time.

Definition 2.1: (bitemporal element). A *bitemporal element* (*BTE*) is defined as the union of a finite set of disjoint Cartesian products of periods of the form $I_T \times I_V$, where I_T is a period of T_T and I_V is a period of T_V.

The notion of *period* is here used in the sense of Section 1.3.2. Using a geometric interpretation, this definition specifies a BTE as a point set in the plane rather than a finite set of rectangles, where each rectangle corresponds to the point set formed by the cross product of a transaction time period and a valid time period. The semantics expressed by a BTE T is that $(t_T, t_V) \in T$ if, and only if, at transaction time t_T the database contains the information that the object bitemporally referenced by T exists at valid time t_V.

As an example, Figure 2.2 shows the graphical representation of a BTE T for a fictitious boundary part b of a parcel. The horizontal axis denotes transaction time and the vertical axis denotes valid time. In transaction time 1990, there is no valid time in which b exists. In transaction time 1991, b exists from valid time 1991 into the indefinite future, since a transaction has taken place providing information about the determination of b. The validity up to the indefinite future (∞) is not specially marked in the graphical representation of a BTE. In transaction time 1992, it turns out that the knowledge at transaction time 1991 about b was erroneous and that b exists from valid time 1992 into the definite future. That is, the determination of the boundary part b was postponed to 1992. Altogether, $T = (1991 \times 1991) \cup (1991 \times 1992) \cup (1992 \times 1992)$, where a year is regarded, for example, as a period of days, weeks, or months.

2.1.3 Spatial Objects Modeled as Simplicial Complexes

The geometric part of the model is based on algebraic topology. *Algebraic topology*, also called *combinatorial topology*, plays an important role in the mathematical field of algebra. It investigates topological structures for classifying and formally describing point

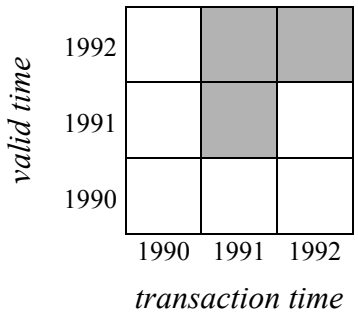


Figure 2.2: Example of a bitemporal element (BTE)
[Worboys 1994 Fig. 4, used with permission.]

sets by using algebraic means. Its techniques rest on problem translation and algebraic manipulation of symbols that represent spatial configurations and their relationships. The basic method consists of three steps: (1) conversion of the problem from a spatial environment to an algebraic environment, (2) solving the algebraic form of the problem, and (3) conversion of the algebraic solution back to the spatial environment. The intersection of two lines, for example, becomes the search for common nodes. The coincidence of two line objects, the neighborhood of two region objects, and the neighborhood of a line and a region object result in a search for common edges or common nodes (depending on the definition of neighborhood) of the lines and the boundaries of regions.

Spatial objects, which are assumed to be embedded in two-dimensional Euclidean space, are represented as *simplicial complexes*, which themselves are composed of *simplexes*.

Definition 2.2: (simplex). Given $k + 1$ points $v_0, ..., v_k \in \mathbb{R}^n$ where the k vectors $\{v_1 - v_0, v_2 - v_0, ..., v_k - v_0\}$ are linearly independent. Then the set $\{v_0, ..., v_k\}$ is called *geometrically independent*, and the point set

$$\sigma = \sigma_k = \{p \in \mathbb{R}^n \mid p = \sum_{i=0}^{k} \lambda_i v_i \text{ with } \sum_{i=0}^{k} \lambda_i = 1, \lambda_i \in \mathbb{R}, \lambda_0, ..., \lambda_k \geq 0\} \subset \mathbb{R}^n$$

is called the (closed) *simplex* of dimension k (or the *k-simplex*) with the vertices $v_0, ..., v_k$, or the *k-simplex spanned* by $\{v_0, ..., v_k\}$.

The above formula collects all points that together form σ. The condition that the sum of all λ_i's has to be equal to 1 ensures that only those points are captured for σ which are located on the boundary of σ or within the interior of σ.

For a given dimension k, a k-simplex is the minimal and elementary spatial object, i.e., a building block from which all more complex spatial objects of this dimension can be constructed. In three-dimensional space, a 0-simplex is a single point or a node, a

1-simplex is a straight line or an edge between two distinct points including the end points, a 2-simplex is a filled triangle connecting three non-collinear points, and a 3-simplex is a solid tetrahedron connecting four non-coplanar points (Figure 2.3).

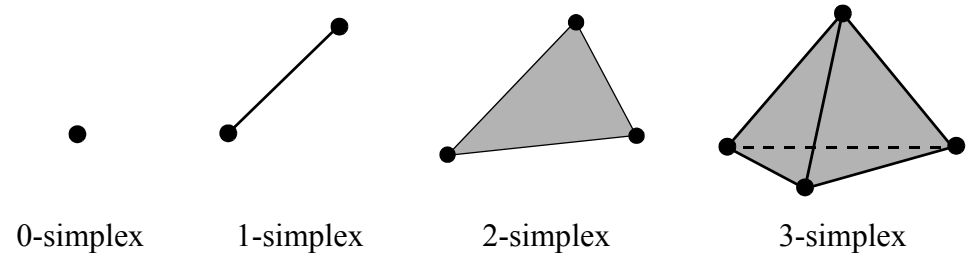


Figure 2.3: Simplex structures of different dimensions

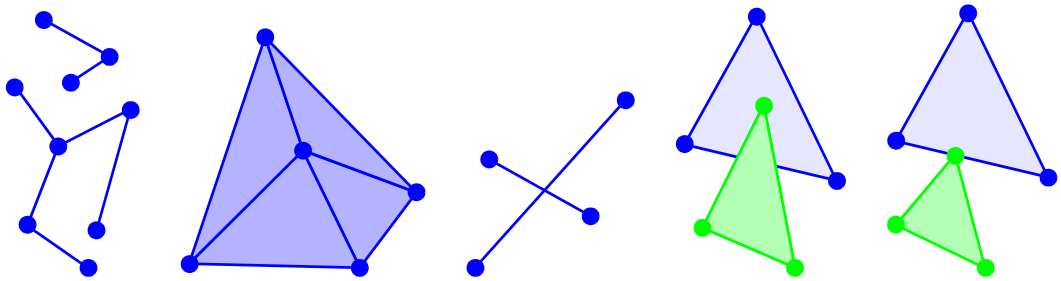
Any k-simplex is composed of $k + 1$ geometrically independent simplexes of dimension $k - 1$. For example, a triangle, a 2-simplex, is composed of three 1-simplexes which are geometrically independent if no two edges are parallel and no edge has length 0. A *face* of a simplex is any simplex that contributes to the composition of the simplex. For example, a node of a bounding edge of a triangle and a bounding edge itself are faces.

Definition 2.3: (simplicial complex). A (*simplicial*) *k-complex* C is a finite set of simplexes so that each face of a simplex in C is also in C, and the intersection of two simplexes in C is either empty or a face of both simplexes. The *dimension* k of C is the largest dimension of the simplexes in C.

This definition is the first of two so-called *completeness principles*. The principle it describes is called *completeness of incidence*. It implies that no two distinct simplexes in C exist which (partially) occupy the same space and that the intersection of lines at points which are neither start nor end points of lines is excluded. Figure 2.4 shows some examples of allowed and forbidden spatial configurations of simplicial complexes.

The second principle, which we will not assume here, is called *completeness of inclusion*. It requires that every l-simplex in C with $l < k$ is a face (boundary simplex) of an $(l+1)$-simplex in C. This avoids geometric anomalies. For example, for $k = 2$, every point is then a start or end point of a line (no isolated points exist), and every line is part of the boundary of a triangle (no dangling lines exist).

Definition 2.4: (oriented simplex, oriented complex). An *oriented k-simplex* is obtained from a k-simplex σ with vertices $v_0, ..., v_k$ by choosing an order for the vertices. We write an oriented k-simplex as an ordered sequence $\sigma = \langle v_0 v_1 v_2 ... v_k \rangle$. An *oriented simplicial complex* is obtained from a simplicial complex by assigning an orientation to each of its simplexes.



(a) (b) (c) (d) (e)

Figure 2.4: Examples of a 1-complex (a), a 2-complex (b), and three configurations which are not simplicial complexes, because the intersection of some of their simplexes is either not a face ((c), (d)), or not a simplex (e).

We now distinguish different parts of a simplex and complex, respectively, and specify what their *boundary* and *interior* is.

Definition 2.5: (boundary, interior). The *boundary* of a k-simplex σ_k, denoted by $\partial\sigma_k$, is the union of all its k+1 (k–1)-simplexes. The *boundary* of a k-complex C, denoted by ∂C, is the smallest complex that contains the symmetric difference of the boundaries of its constituent k-simplexes. The *interior* of a k-complex C, denoted by $C°$, is the union of all (k–1)-simplexes which are not part of the boundary of C.

For instance, the boundary of a 2-simplex (triangle) $\sigma = \langle v_0v_1v_2\rangle$ is $\partial\sigma = \{\langle v_0v_1\rangle, \langle v_1v_2\rangle, \langle v_2v_0\rangle\}$ which is a set of three 1-simplexes (edges). The boundary of a 2-complex $C = \{\langle x_1y_1z_1\rangle, ..., \langle x_ny_nz_n\rangle\}$ is $\partial C = \Delta(\partial\langle x_1y_1z_1\rangle, ..., \partial\langle x_ny_nz_n\rangle)$ where Δ denotes the symmetric difference operation with $\Delta(A, B) = (A \setminus B) \cup (B \setminus A)$ for two sets A and B. The common edges of the 2-dimensional simplexes of C form the interior of C.

As an example, which also demonstrates how algebraic computation works in algebraic topology, the computation of the boundary of a 2-complex C_2 consisting of two adjacent oriented 2-simplexes $\sigma_2 = \langle v_1v_2v_4\rangle$ and $\tau_2 = \langle v_2v_3v_4\rangle$ is illustrated in Figure 2.5. We obtain $\partial\sigma_2 = \langle v_2v_4\rangle - \langle v_1v_4\rangle + \langle v_1v_2\rangle$ and $\partial\tau_2 = \langle v_3v_4\rangle - \langle v_2v_4\rangle + \langle v_2v_3\rangle$. Then $\partial C_2 = \partial\sigma_2 + \partial\tau_2 = \langle v_2v_4\rangle - \langle v_1v_4\rangle + \langle v_1v_2\rangle + \langle v_3v_4\rangle - \langle v_2v_4\rangle + \langle v_2v_3\rangle = \langle v_1v_2\rangle + \langle v_2v_3\rangle + \langle v_3v_4\rangle - \langle v_1v_4\rangle = \langle v_1v_2\rangle + \langle v_2v_3\rangle + \langle v_3v_4\rangle + \langle v_4v_1\rangle$.

For later operation definitions, we will need the concept of *common refinement* of two simplicial complexes, which corresponds to a special kind of geometric union and identifies common parts, i.e., two simplicial complexes become acquainted with each other. Its definition makes use of the notion of *planar embedding*, which we define first.

Definition 2.6: (planar embedding). For a given simplicial complex $C = \{s_1, ..., s_n\}$, its *planar embedding* is given as $emb(C) = \bigcup_{i=1}^{n} s_i$.

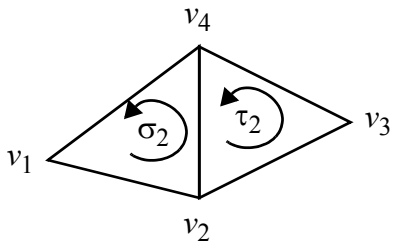
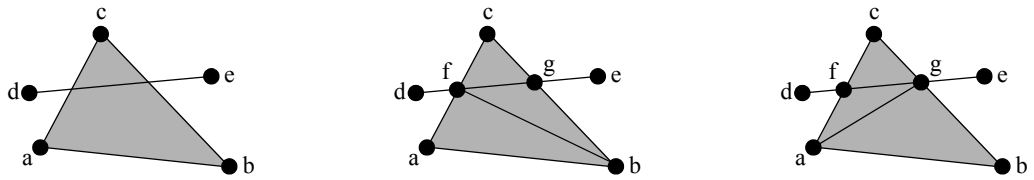


Figure 2.5: Example of a complex consisting of two simplexes

Definition 2.7: (common refinement). A *common refinement* of two simplicial complexes C_1 and C_2, denoted by *refine*(C_1, C_2), is a simplicial complex which has the same planar embedding as the union of the embeddings of C_1 and C_2, i.e., *emb*(*refine*(C_1, C_2)) = *emb*(C_1) $\cup$ *emb*(C_2).

This construction is in general not unique due to possible different decompositions into simplex structures (see Figure 2.6).



(a) (b) (c)

Figure 2.6: Complexes *abc* and *de* (a) and two possible common refinements ((b), (c)). [Worboys 1994, Fig. 7, used with permission.]

2.1.4 Spatio-Bitemporal Objects

The combination of bitemporal elements and simplicial complexes leads us to *spatio-bitemporal objects*. The main idea is to attach BTEs (Definition 2.1) as labels to components of simplicial complexes (Definition 2.3).

Definition 2.8: (ST-simplex). An *ST-simplex* is an ordered pair (S, T) where S is a simplex and T is a BTE. For an ST-simplex R = (S, T) let $\pi_s(R) = S$ and $\pi_t(R) = T$.

An ST-simplex R indicates that a spatial configuration S exists over a given range T of transaction and valid times. π_s and π_t are spatial and bitemporal projection functions.

The concept of *ST-complex* is introduced now for describing the structure of a general, bitemporally referenced spatial configuration.

Definition 2.9: (ST-complex). An *ST-complex C* is a finite set of ST-simplexes fulfilling the following properties:

1. The spatial projections of ST-simplexes in *C* are pairwise disjoint.
2. Taken together, these spatial projections form a simplicial complex.
3. $\forall R, R' \in C : \pi_s(R)$ is a face of $\pi_s(R') \Rightarrow \pi_t(R) \supseteq \pi_t(R')$.

Condition 1 implies in particular that the same simplex may not occur with different BTEs in the same ST-complex. Condition 2 requires that the underlying geometric structure is a simplicial complex. The reason for condition 3 is that a simplex cannot exist without its faces.

As an example, in the left part of Figure 2.7 we consider the temporal development of the boundary of parcel 4 in Figure 2.1 as an ST-complex. Areal properties are not represented here; they would require a representation of the parcel by triangular simplexes. The right part of Figure 2.7 shows which BTEs are associated with which boundary segments and vertices. Remember that in the graphical representation of each BTE, the horizontal axis denotes transaction time and the vertical axis valid time.

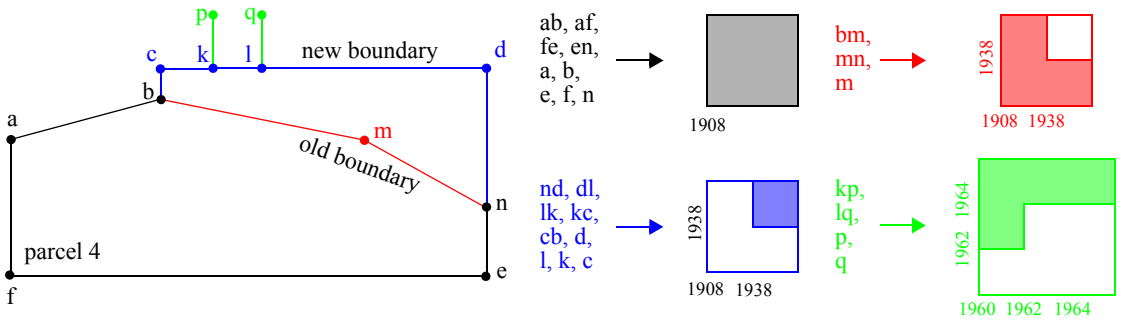


Figure 2.7: ST-complex example: boundary and face annotation with BTEs of parcel 4 [Worboys 1994 Fig. 6, used with permission.]

2.1.5 Spatio-Bitemporal Operations

Besides purely spatial and purely temporal operations, which we will not mention here, spatio-bitemporal operations can be defined on ST-complexes. Some of them will be introduced and explained in the following. We first list their signatures (Table 2.1):

For the predicate definitions of the subset-relationship and the equality of two ST-complexes *C* and *C*', we consider *C* and *C*' as embedded in four-dimensional space comprising two spatial and two temporal dimensions.

$=$:	*ST-complex* × *ST-complex*	→ *boolean* (*ST-equal*)
$\subset_{ST}$:	*ST-complex* × *ST-complex*	→ *boolean*(*ST-subset*)
∂:	*ST-complex*	→ *ST-complex*(*ST-boundary*)
π_s:	*ST-complex*	→ *S-complex*(*S-project*)
π_t:	*ST-complex*	→ *BTE*(*T-project*)
$\times_\beta$:	*ST-complex* × *ST-complex*	→ *ST-complex*(*ST-β-product*)
$\cup_{ST}$:	*ST-complex* × *ST-complex*	→ *ST-complex*(*ST-union*)
$\cap_{ST}$:	*ST-complex* × *ST-complex*	→ *ST-complex*(*ST-intersection*)
$\setminus_{ST}$:	*ST-complex* × *ST-complex*	→ *ST-complex*(*ST-difference*)
$\sigma^s{}_X$:	*ST-complex*	→ *ST-complex*(*S-select*)
$\sigma^t{}_\phi$:	*ST-complex*	→ *ST-complex*(*T-select*)

Table 2.1: A collection of spatio-bitemporal operations

Definition 2.10: (ST-subset, ST-equal). Let C and C' be two ST-complexes. Then

1. $C \subset_{ST} C' \Leftrightarrow \forall\ (x, y, w, z) \in (S, T) \in C\ \exists\ (S', T') \in C' : (x, y, w, z) \in (S', T')$
2. $C =_{ST} C' \Leftrightarrow C \subset_{ST} C' \wedge C' \subset_{ST} C$

For defining the boundary of an ST-complex, we make use of the purely spatial boundary operation ∂ (Definition 2.5).

Definition 2.11: (ST-boundary). For an ST-complex C let $\partial C = \{(S, T) \mid S \in \partial(\pi_s(C))\}$.

Projection operators are needed to map either to the spatial or to the temporal domain and then to continue computation purely in that domain. Intuitively, the spatial projection of an ST-complex is a complex representing the whole knowledge one has ever had about the spatial extent of the ST-complex over all transaction and valid times. The bitemporal projection of an ST-complex is a BTE gathering all transaction and valid times at which parts of the ST-complex have existed. The projection operators extend the ones on simplexes.

Definition 2.12: (spatial projection, bitemporal projection). Let ST-complex $C = \{(S_1, T_1), ..., (S_n, T_n)\}$. Then

1. $\pi_s(C) = \{S_1, ..., S_n\}$
2. $\pi_t(C) = \bigcup_{1 \le i \le n} T_i$

The next operation we discuss is the so-called *spatio-bitemporal* β-*product*. It allows the composition of two ST-complexes which is parameterized by a so-called β-operation. A β-*operation* is a binary operation with the signature β: *BTE* × *BTE* → *BTE*.

Definition 2.13: (ST-β-product). Let us assume two ST-complexes C_1 and C_2 and a β-operation on BTEs. Let simplicial complex R be a common refinement of $\pi_s(C_1)$ and $\pi_s(C_2)$. We define $C_1 \times_\beta C_2$ to be the smallest ST-complex (with respect to the ST-subset relationship introduced in Definition 2.10) which contains the set of ST-simplexes $\{(S, \beta({T_1}^S, {T_2}^S)) \mid S \in R\}$ where ${T_1}^S$ and ${T_2}^S$ are the BTEs associated with the smallest faces of $\pi_s(C_1)$ and $\pi_s(C_2)$, respectively, which contain S.

This "dense" definition can be explained by giving an algorithm computing $C_1 \times_\beta C_2$. In a first step, we compute the spatial part of the result which is given as a common refinement R of the spatial projections of C_1 and C_2. In a second step, for each constituent face (simplex) S of R, we determine the pertaining BTE as follows: We know that S must explicitly exist either in the spatial projection of C_1, or in the spatial projection of C_2, or in both. Otherwise, it would not be part of R. If this is the case, then we can directly take the pertaining BTE ${T_1}^S$, or ${T_2}^S$, or both. If S does not explicitly exist in one of the spatial projections of C_1 or C_2, the BTE of the smallest face containing S is taken as ${T_1}^S$ or ${T_2}^S$, respectively. We then apply β to ${T_1}^S$ and ${T_2}^S$. In a third step, we remove all those faces S from the result with an empty BTE. The definition will also become clearer below where we look at some examples for β.

The result of $C_1 \times_\beta C_2$ depends on the choice of the common refinement R, which in general is not unique. But because all possible common refinements have the same planar embedding, i.e., comprise the same point set, the different results will all be ST-equal[2]. The reason is that different common refinements of $\pi_s(C_1)$ and $\pi_s(C_2)$ only produce different subdivisions (for example, triangulations, splitting of edges) of $\pi_s(C_1)$ and $\pi_s(C_2)$ with the same planar embedding. This fact does not affect the BTE attached to the same point of each possible common refinement. In other words, a point can belong to different faces due to different common refinements, but the pertaining BTE does not change.

We now consider a few important applications of the ST-β-product. With $\beta \in \{\cup_{ST}, \cap_{ST}, \setminus_{ST}\}$, the ST-β-product can be used to define set-theoretic *union*, *intersection*, and *difference* operations on two ST-complexes.

Definition 2.14: (ST-union, ST-intersection, ST-difference). Let C_1 and C_2 be two ST-complexes. Then

1. $C_1 \cup_{ST} C_2 = C_1 \times_\cup C_2$
2. $C_1 \cap_{ST} C_2 = C_1 \times_\cap C_2$
3. $C_1 \setminus_{ST} C_2 = C_1 \times_\setminus C_2$

2. More precisely, these operations act on equivalence classes of ST-equal ST-complexes and not on single ST-complexes. For notational simplicity, we allow operations to act on single ST-complexes.

Figure 2.8 illustrates these definitions for union (d) and intersection (e) on the basis of

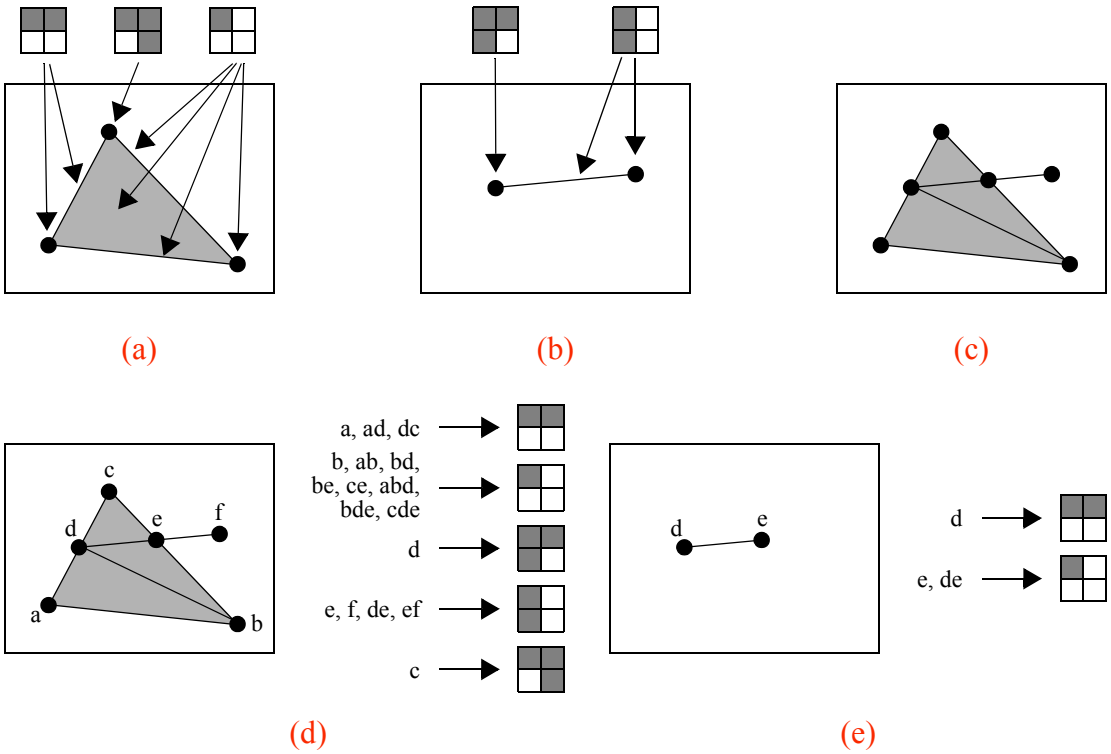


Figure 2.8: ST-complex C_1 (a), ST-complex C_2 (b), a common refinement of $\pi_s(C_1)$ and $\pi_s(C_2)$ (c), the ST-union of C_1 and C_2 (d), and the ST-intersection of C_1 and C_2 (e). [Worboys 1994, Figures 8-11, used with permission.]

two ST-complexes C_1 (a) and C_2 (b) with the common refinement of their spatial projections (c) and the BTEs attached to their single spatial simplexes. The union contains all the elements of the refined spatial projections of C_1 and C_2 where each spatial simplex has associated with it the union of the BTEs associated with that element in C_1 and C_2. The intersection yields an ST-complex whose spatial simplexes are shared by the refined spatial projections of C_1 and C_2 with a non-empty intersection of corresponding BTEs in C_1 and C_2. If an empty BTE is calculated for a common spatial simplex, the spatial simplex is omitted from the resulting complex.

The set of ST-simplexes $\{(S, \beta(T_1^S, T_2^S)) \mid S \in R\}$ does not necessarily form an ST-complex. However, there is always a unique minimum ST-complex which contains this set. An example is the difference of the two ST-complexes C_1 and C_2 in Figure 2.8a and b. The faces of interest are the 1-simplex *de* and the two 0-simplexes *d* and *e* which (the reader should check this) all are associated with the empty BTE. This means that all three

simplexes should be omitted from the resulting complex. Whereas this is correct for *de* and *e*, this is incorrect for *d*, since *d* is also the bounding node of the simplex *de* which becomes incomplete. Consequently, *d* has to be maintained and obtains the BTE  as the maximum BTE of all its incident edges.

The *spatial selection* $\sigma^s_X(C)$ is another operation that can be expressed using the ST-β-product. It allows one to choose from an ST-complex C all those ST-simplexes whose spatial projection is contained in a given simplicial complex X.

Definition 2.15: (S-select). Let $X = \{S_1, ..., S_n\}$ be a simplicial complex, and let $D_X = \{(S_1, T_T \times T_V), ..., (S_n, T_T \times T_V)\}$ be the ST-complex where each simplex of X is associated with the universal BTE $T_T \times T_V$. The *spatial selection* on an ST-complex C with respect to X is then defined as $\sigma^s_X(C) = C \cap_{ST} D_X$.

For example, the spatial selection on C_1 in Figure 2.8a with respect to the spatial projection of C_2 in Figure 2.8b is again shown in Figure 2.8e.

For the definition of the next and last operation considered here we need the two auxiliary notions of *ST-comparable* and *ST-minimum*.

Definition 2.16: (ST-comparable). The elements of a set $\{C_1, ..., C_n\}$ of ST-complexes are *comparable*, if and only if $\forall\ 1 \le i < j \le n : C_i \subset_{ST} C_j \vee C_j \subset_{ST} C_i$.

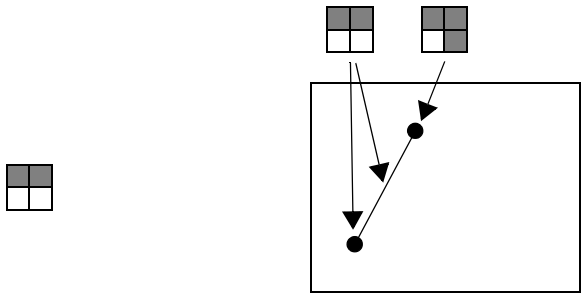
Definition 2.17: (ST-minimum). The *minimum* of a set $\{C_1, ..., C_n\}$ of comparable ST-complexes is defined as $min_{ST}(\{C_1, ..., C_n\}) = C$ such that

1. $C \in \{C_1, ..., C_n\}$
2. $\forall\ 1 \le i \le n : C =_{ST} C_i \vee C \subset_{ST} C_i$

We are now able to define the *temporal selection* $\sigma^t_\phi(C)$ which selects from an ST-complex C the smallest ST-complex, each of whose simplicial components fulfils a temporal condition specified by a formula ϕ.

Definition 2.18: (T-select). Let $\phi(t)$ be a first-order formula which may contain BTEs as constants, β-operations for functions, and a single free variable t. The *temporal selection* on an ST-complex C with respect to ϕ is then defined as $\sigma^t_\phi(C) = min_{ST}(\{C' \mid C' = \{(S, T) \in C \mid \phi(T)\}\})$.

For example, if we take the ST-complex C_1 in Figure 2.8a and the BTE B in Figure 2.9a, the result of the temporal selection $\sigma^t_{t \supseteq B}(C_1)$ is shown in Figure 2.9b.



(a) (b)

Figure 2.9: BTE B (a) and the temporal selection $\sigma^{t}{}_{t \supseteq B}(C_1)$ (b).
[Worboys 1994, Figures 12-13, used with permission.]

2.1.6 Querying

An essential feature of a (spatio-temporal) database system is that it provides a language for posing queries. The spatio-temporal data model described so far can be taken as the basis of a *query algebra*. We will illustrate this by integrating the spatio-bitemporal data type *ST-complex* into the relational setting and by incorporating the spatio-bitemporal operations (Table 2.1) into the well-known standard database query language SQL. For reasons of user-friendliness, readability, and compatibility with SQL, we will not employ the mathematical notations for the operations but replace them by the purely textual terms given in parentheses in Table 2.1.

Again we take the land owner scenario and assume that the land parcel data are stored in a relation with the scheme

```
parcels (parcel-id: integer, owner: string,
         area: ST-complex, building: ST-complex)
```

This schema stores information about the parcel identifier, the owner of a parcel, and the development of the parcel area as well as the building on the parcel over space and time. Parcel identifier and owner together form the key of the schema. It is striking that the non-standard spatio-bitemporal data type *ST-complex* is used in the same way as an attribute type as the standard data types *integer* and *string*. This is possible, because the type *ST-complex* is modeled and used as an *abstract data type*, that is, only its name is visible outwards but its internal, complex structure is hidden from the user in the data type representation.

We are now able to formulate some queries:

“How did the building on parcel 1 look like in 1958?”

```
SELECT S-project(T-select((_, 1958), building))
FROM parcels
WHERE parcel-id = 1 AND (_, 1958) IN T-project(building)
```

In the *where*-clause, the first part selects all those tuples belonging to parcel 1. The second part identifies that tuple with a *building* attribute value whose BTE contains 1958 as a valid time. The underscore "_" serves as a wildcard for transaction time here. In the *select*-clause we first determine the spatio-bitemporal complex for the year 1958 by temporal selection and then compute the spatial projection. Note that we have slightly changed the signature of *T-select* and positioned the temporal selection condition as the first operand.

"How has the building on parcel 1 been extended during 1958 and 1974?"

```
SELECT S-project(ST-difference(T-select((_, 1974), building),
                               T-select((_, 1958), building)))
FROM parcels
WHERE parcel-id = 1
```

The *where*-clause selects parcel 1. By temporal selection, the *select*-clause first determines the snapshots of the building on parcel 1 in the years 1974 and 1958 as an ST-complex each, then computes their difference yielding an ST-complex again, and finally calculates the spatial projection of the result which just describes the geometric difference of the building.

"Does the path currently pass through land that was ever part of Jane Benson's house?"

```
SELECT NOT isempty(S-intersection(
                     S-project(T-select((now_DB, _), p.area)),
                     S-project(q.building))) AS is_part
FROM parcels AS p, parcels AS q
WHERE p.parcel-id = 7 AND p.owner = 'public' AND
      q.owner = 'Jane Benson'
```

The *where*-clause joins parcel 7, which belongs to the path, with all properties Jane Benson ever had. The *select*-clause computes a boolean value for the new attribute *is_part* as follows: First, the ST-complex belonging to the area of the path is temporally restricted to now_{DB}, which indicates a BTE respresenting all bitemporal times with database time *now*, and afterwards its spatial projection is determined. Then, the spatial projection of the ST-complex describing the buildings owned by Jane Benson is computed. The resulting two spatial complexes are intersected by the operation *S-intersection*. Finally, the assumed predicate *isempty* checks whether the intersection result is empty. If the result is not empty, the path and the building under consideration have shared a part in the past.

"Has Jack Meyer's house ever shared a common boundary with the path?"

```
SELECT NOT isempty(ST-intersection(
                      ST-boundary(p.area),
                      ST-boundary(q.building))) AS is_part
FROM parcels AS p, parcels AS q
WHERE p.parcel-id = 7 AND p.owner = 'public' AND
      q.owner = 'Jack Meyer'
```

The *select*-clause determines the boundaries of the two ST-complexes that describe the path area and Jack Meyer's building and computes their intersection.

Exercise 2.1: Where exactly becomes the "discrete" nature of this approach visible? □

Exercise 2.2: Formulate the following queries in the SQL-like style presented above and design extensions to the language if necessary:

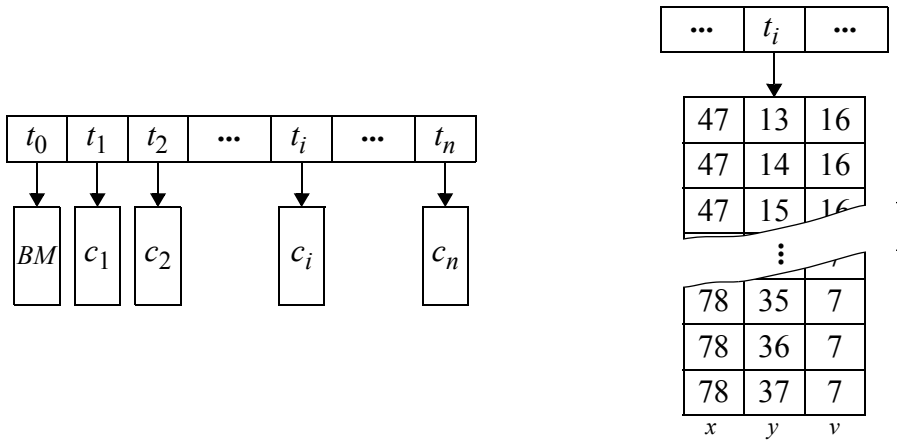
(a) When was a parcel owned by a specific person not a developed real estate?
(b) When was the school (as the first building) constructed on parcel 4? □

2.2 An Event-Based Approach

The second model to be presented propagates the event-based approach. In this time-based approach, location in time becomes the primary organizational basis for recording change. An event represents a change in state at some instant like the change of a value associated with a location in space at some time t. In this model, only valid time is of importance. The treatment of transaction time is not included.

2.2.1 The Model

The time associated with each change (i.e., each event) is stored in increasing order from an initial time t_0 (for example, March 1, 1963) to the latest recorded change at time t_n. This means that the representation of change is organized as a function of a *discrete* time domain to some codomain of interest. Mostly, the length of any temporal period, i.e., the temporal distance between $t_{i\text{-}1}$ and t_i, and any other such period will vary. The idea now is to associate with each timestamp t_i only the changes (the "deltas") that occurred between $t_{i\text{-}1}$ and t_i but not the complete changed snapshot. All changes between $t_{i\text{-}1}$ and t_i are collected in a set c_i. The only exception is at time t_0 when the initial scenario or base map BM has to be recorded once. This is illustrated in the event list in Figure 2.10a. Figure 2.10b gives more insight how changes are recorded at time t_i. In this approach, the geometry is stored as a bitmap or image so that changes at time t_i relate to a set of individual locations (x_{ik}, y_{ik}) with altered values v_{ik}.



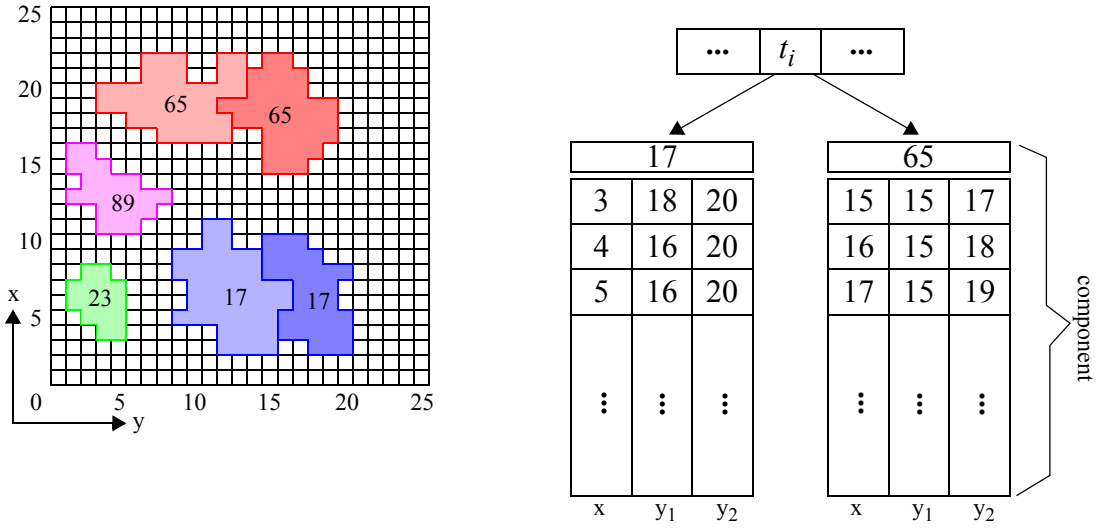
(a) (b)

Figure 2.10: Example of an event list where t_i contains a time value (for example, day, month, and year), *BM* is the base map, and c_i contains all changes that occurred between the times t_{i-1} and t_i (a), structure of the changes in c_i which comprises a collection of triplets (x_{ik}, y_{ik}, v_{ik}), each of which describes the new value v_{ik} at t_i for location (x_{ik}, y_{ik}) (b).

From a representation point of view, a main drawback of this method is that the number of triplets (x, y, v) to be stored in c_i depends directly on the total number of discrete locations whose values changed between t_{i-1} and t_i. An improvement is to group together all individual locations that share the same common value v. Such a value-specific group is called a *component* and the shared value v is called its *descriptor*. Instead of once for every location per event, each new value is stored only once per event with all locations that map to this value. This leads to a separate component for each new value.

A further improvement is to apply raster run-length encoding within each component in order to reduce the amount of storage space required for recording locations. This means that, if in a row of the image with the same x-coordinate several consecutive y-coordinates share the same value v, these y-coordinates can be summarized by an interval which usually leads to a much shorter representation. Figure 2.11 gives an example. In Figure 2.11a a simplified map consists of light shaded and dark shaded locations. Each location is labeled with a numerical value. The light shaded locations visualize the development of the base map *BM* after the changes at the times t_1, ..., t_{i-1}. The dark shaded locations represent the changes at time t_i. Figure 2.11b shows the event components for the changes at time t_i including their descriptors.

The temporal ordering of events in the event list enables search and retrieval of particular time periods, but also of change to specific values in these time periods. Such an ordering also facilitates the comparison of different temporal sequences for different thematic



(a) (b)

Figure 2.11: Status of a simplified map (light shaded) after applying the "deltas" at times t_1, ..., t_{i-1} to a base map *BM* and the changed (dark shaded) locations at time t_i (a), and the corresponding event components (b).

domains, or of the same thematic domain in different geographic areas. A comparison of only the times at which events occur (i.e., looking for temporal patterns) can be performed by retrieving the times alone directly from the event list; it is not necessary to retrieve associated values and locations. Due to the grouping according to values, the spatial locations carrying this new value can be easily determined. In particular, we can pose the following queries dealing with a single temporal sequence:

1. Retrieve all locations which changed at a given time t_i.
2. Retrieve all locations which changed at a given time t_i to a given value.
3. Retrieve all locations which changed at a given time t_i with their new values.
4. Retrieve all locations which changed between t_i and t_j.
5. Retrieve all locations which changed between t_i and t_j to a given value.
6. Retrieve all locations which changed between t_i and t_j with their new values.
7. Calculate the total change in an area to a given value between t_i and t_j.
8. Did location (x, y) change to a new value at a given time t_i?
9. Did location (x, y) change to a new value between t_i and t_j?
10. When did location (x, y) change to a new value?

2.2.2 Query Processing Algorithms

This time, we will not show how these queries can be formulated in a database query language like in Section 2.1.6 but demonstrate how algorithms can be designed for execut-

ing them during the query processing stage. We will also determine the run time complexity of these algorithms.

We begin with the design of an algorithm for evaluating query 2 from above which requires a two-stage search. The first stage of the search is to find the event with the desired time-stamp t within the event list *el* (see Figure 2.10), which is arranged in increasing temporal order. If the entire event list occurred after the desired time, i.e., $t_0 > t$, the search returns the empty list. Otherwise, the search continues until the time t_e associated with an event e is larger than or equal to the desired time t, i.e., $t_e \geq t$. Here, the assumption is that t need not necessarily match any time-stamp stored in the event list. If $t_e \neq t$ for all events e, we use the simple rule of closest temporal distance as to whether t_e or $t_{e\text{-}1}$ is selected. This decision can, of course, change depending on the application. The second stage is to determine the component c associated with this event whose descriptor matches the given value *gv*. All *xy*-locations within that component are then returned by a function *xylist*. This leads us to the following algorithm:

```
algorithm GetChangedLocsAtInstantForValue(el, t, gv)
input: event list el, time t, value gv;
output: list xylist of locations changing to value gv at time t;
begin
  if t0 > t then return null fi;
  foreach event e in el do
    if te >= t then
      if t <= (te + te-1)/2 then ev = e-1 else ev = e fi;
      foreach component c of event ev do
        if descriptor(c) == gv then return xylist(c) fi
      od
    fi
  od
end GetChangedLocsAtInstantForValue.
```

Both the search of the event list and the search of the component descriptors within the desired event once found are linear searches. Hence, the worst time complexity is $O(n_e + c_e + k_c)$, where n_e is the total number of events in the event list, c_e is the maximum number of components for any given event, and k_c is the maximum number of changes stored in any component c. Note that n_e and c_e are input parameters while k_c is an output parameter. This result can be improved to $O((\log n_e) + c_e + k_c)$ by using any $O(\log n)$ search, where n denotes the total number of elements to be searched.

Exercise 2.3: A snapshot model would store the complete map including the changes for each event at a time t_i. Describe roughly (without algorithmic notation) an algorithm performing the same task as above and utilizing the snapshot model. What are the differences? What is the runtime behavior of the algorithm? □

Next, we deal with an algorithm for query 5 asking for changes to a given value in a given temporal interval.

```
algorithm GetChangedLocsInIntervalForValue(el, t1, t2, gv)
input: event list el, start time t1, end time t2, value gv;
output: list xylist of locations changing to value gv in [t1, t2];
begin
  if t0 > t2 then return null fi;
  foreach event e in el do
      if t1 <= te <= t2 then
        foreach component c of event do
          if descriptor(c) == gv then return xylist(c) fi
        od
      fi
  od
end GetChangedLocsInIntervalForValue.
```

This algorithm is a slight variation of the preceding one that uses a range of temporal values at the first stage of search and that retrieves all locations stored in components with a descriptor *gv* for all events from a starting time t_1 to a finishing time t_2. For the sake of simplicity, we assume that the temporal interval $[t_1, t_2]$ is wide enough so that at least one event will be found in between.

The runtime complexity of this algorithm is $O((\log n_e) + n_f \cdot (c_e + k_c))$, where $\log n_e$ is the amount of time needed to search the event list for the starting event, n_f is the number of events retrieved sequentially up to the finishing event so that $t_1 \le t_e \le t_2$, c_e is the maximum number of components for any event e, and k_c is the maximum number of changes stored in any component c. Note that n_e, n_f, and c_e are input parameters while k_c is an output parameter. The worst-case scenario in terms of efficiency is the case where the starting time coincides with the first event in the event list and the finishing time coincides with the last event in the event list. In this case, the run time complexity is $O(n_e \cdot (c_e + k_c))$.

Finally, we discuss an algorithm evaluating query 7 asking for the total change to a given value *gv* in an area during a given temporal interval. The amount of areal change for a given value at some time t_i is determined by the total number of areal units represented within the corresponding component or within an *xylist*(c) returned, e.g., by either of the algorithms above. Since run-length coding is employed for both of these, counting the total number of changed areal units is rather simple as the following algorithm shows. This algorithm assumes an input *xylist* as a run-length-encoded list of (x, y)-locations associated with our desired value *gv* of interest.

```
algorithm Area(xylist)
input: run-length-encoded list xylist of (x, y)-locations;
output: area as an integer value
begin
  area = 0;
  foreach entry (x, y1, y2) in xylist do
    area = area + (y2 - y1 + 1)
  od;
  return area
end Area.
```

Obviously, the run time complexity is a linear function of the number of entries in the list *xylist*.

Exercise 2.4: Discuss algorithms for the evaluation of the remaining queries listed above. Determine their runtime complexity. □

2.3 Bibliographic Notes

The presentation of the spatio-temporal model in Section 2.1 is based on (Worboys 1994). We have especially added a more detailed discussion of spatial complexes as one of the two main underlying pillars of this model and the description of an embedding of the query algebra into an SQL-like query language context. Literature about algebraic topology can be found in (Armstrong 1983, Croom 1978). Here, only the basic concepts are of interest.

A number of papers exploit algebraic topology for spatial modeling. Frank and Kuhn (1986) propose a topological model which is based on simplicial complexes (cell complexes). An algebra for complexes is introduced which provides a variety of operations like creating an initial complex, adding a point or line to a complex, connecting two points in a complex with a line, deleting a point, or deleting a line. These operations can, e.g., be used for computing a common refinement or the spatial part of a spatio-bitemporal operation. In a continuation of this work, Egenhofer et al. (1989) show the simplicity of an implementation of simplicial structures. They present a simplicial algebra with only a small set of operations that fulfill closure properties, i.e., an operation manipulating one or more simplicial complexes can produce only a simplicial complex. Update operations are consistent, and the completeness principles are ensured after each modification. Algorithms proposed relate to the insertion of a node, a line, and a polygon into a simplicial decomposition. Finally, in (Egenhofer 1989, Egenhofer 1991) simplicial topology is applied for the definition of topological relationships.

The presentation of the spatio-temporal model in Section 2.2 is based on (Peuquet and Duan 1995). The authors denote their approach as an event-based spatio-temporal data model (ESTDM).

Two survey articles summarizing the earlier work on spatio-temporal databases like the models presented in this chapter are (Abraham and Roddick 1999, Peuquet 2001). The latter already includes some of the work on moving objects.

Solutions to Exercises

Solution 1.1:

(a)

How many people live within ten kilometers from the river Rhine? (Cities are modeled as points, hence if the point is within that distance we count the whole population.)

Obviously we need to be able to measure the distance between a point and a line, hence need an operation:

distance: *points* × *line* → *real*

This will return the minimal distance between any of the points in the first argument and the second argument line. The query is:

```
SELECT SUM(c.pop)
FROM rivers AS r, cities AS c
WHERE r.name = 'Rhine' and distance(c.location, r.route) < 10
```

(b)

With which of its neighbour countries does Germany have the longest common border?

We use an operation

common_border:*region* × *region* → *line*

which returns the intersection of two *region* boundaries. The query is:

```
LET border_lengths =
SELECT t.name AS name, length(common_border(s.area, t.area))
  AS length
FROM states AS s, states AS t
WHERE s.name = 'Germany' AND s.area adjacent t.area;

SELECT name, length
FROM border_lengths
WHERE length =
  SELECT MAX(length)
  FROM border_lengths
```

(c)

Find the locations of all bridges of highways crossing rivers. Return them as a relation with the name of the highway, the name of the river, and the location.

We introduce a predicate

intersects: *line* × *line* → *bool*

The query is:

```
SELECT r.name, h.name, intersection(r.route, h.route)
FROM rivers AS r, highways AS h
WHERE r.route intersects h.route
```

Solution 1.2:

Name	*Location*	*Time*
Mr. Jones	Edinburgh, Grand Hotel	5
Mr. Jones	Edinburgh, Traveler's Inn	8
Mr. Jones	Aviemore, Golf Hotel	15
Mr. Jones	Home	17
Anne	Home	5
Anne	Brighton, Linda	7
Anne	Home	12
Anne	Parents	16

Table 1: Model by Segev

Name	*Location*	*Time*
Mr. Jones	Edinburgh, Grand Hotel	[5-7]
Mr. Jones	Edinburgh, Traveler's Inn	[8-14]
Mr. Jones	Aviemore, Golf Hotel	[15-16]
Mr. Jones	Home	[17-∞]

Table 2: Model by Sarda

Name	*Location*	*Time*
Anne	Home	[5-6]
Anne	Brighton, Linda	[7-11]
Anne	Home	[12-15]
Anne	Parents	[16-∞]

Table 2: Model by Sarda

Name	*Location*
5 → Mr. Jones ...	5 → Edinburgh, Grand Hotel ... 7 → Edinburgh, Grand Hotel 8 → Edinburgh, Traveler's Inn ... 14 → Edinburgh, Traveler's Inn 15 → Aviemore, Golf Hotel 16 → Aviemore, Golf Hotel 17 → Home ...
5 → Anne ...	5 → Home 6 → Home 7 → Brighton, Linda ... 11 → Brighton, Linda 12 → Home ... 15 → Home 16 → Parents ...

Table 3: HRDM

Solution 1.3:

We also show all intermediate steps to make it easier to follow.

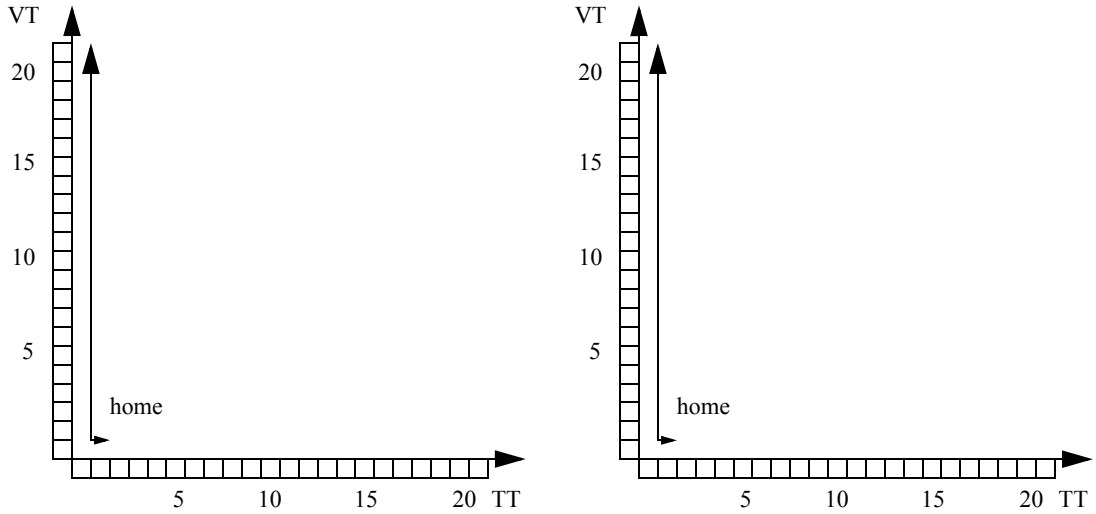


Figure 1: Jennifer's knowledge, December 1st

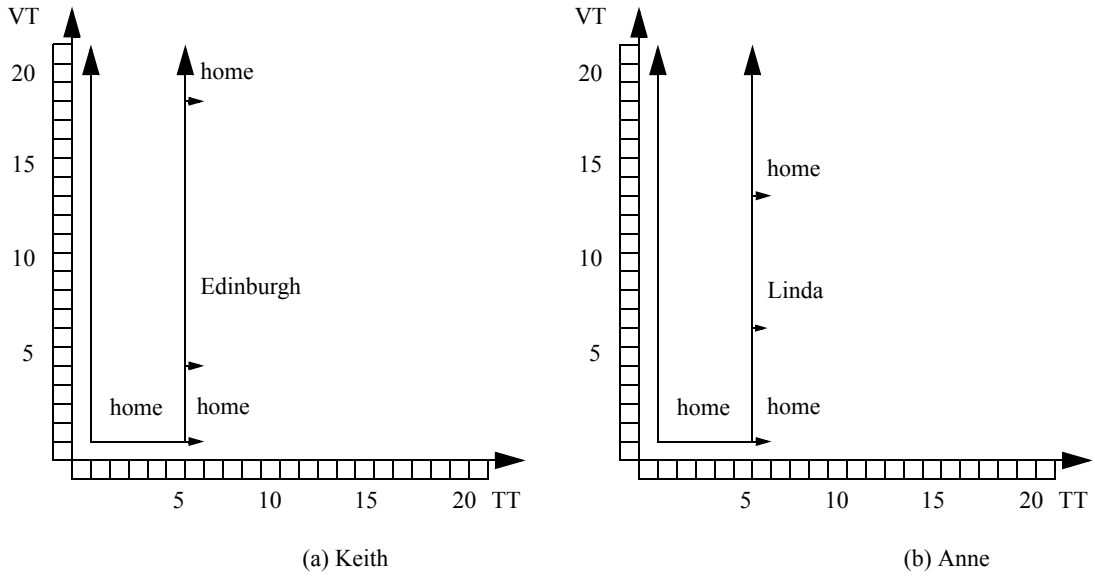


Figure 2: Jennifer's knowledge, December 6

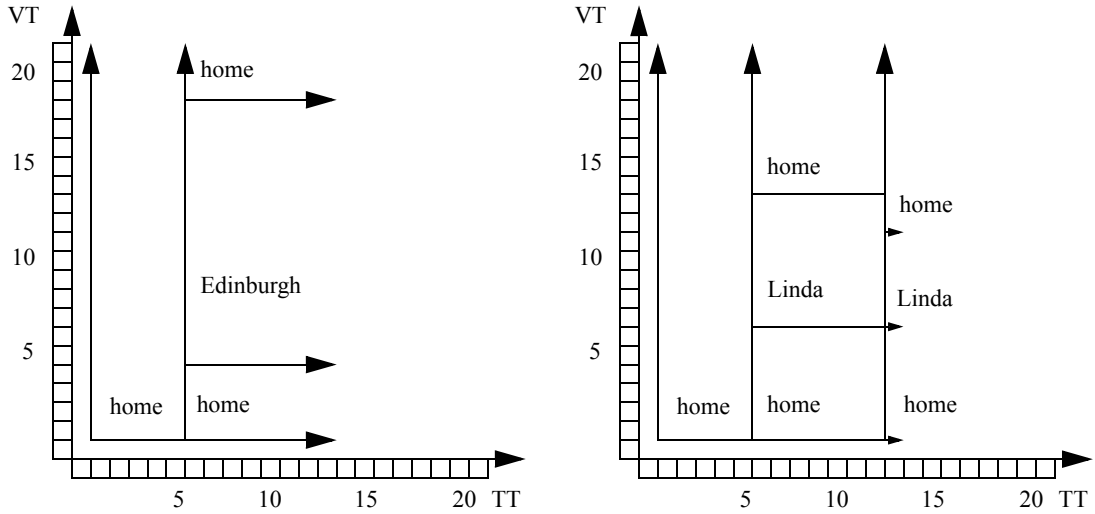


Figure 3: Jennifer's knowledge, December 13

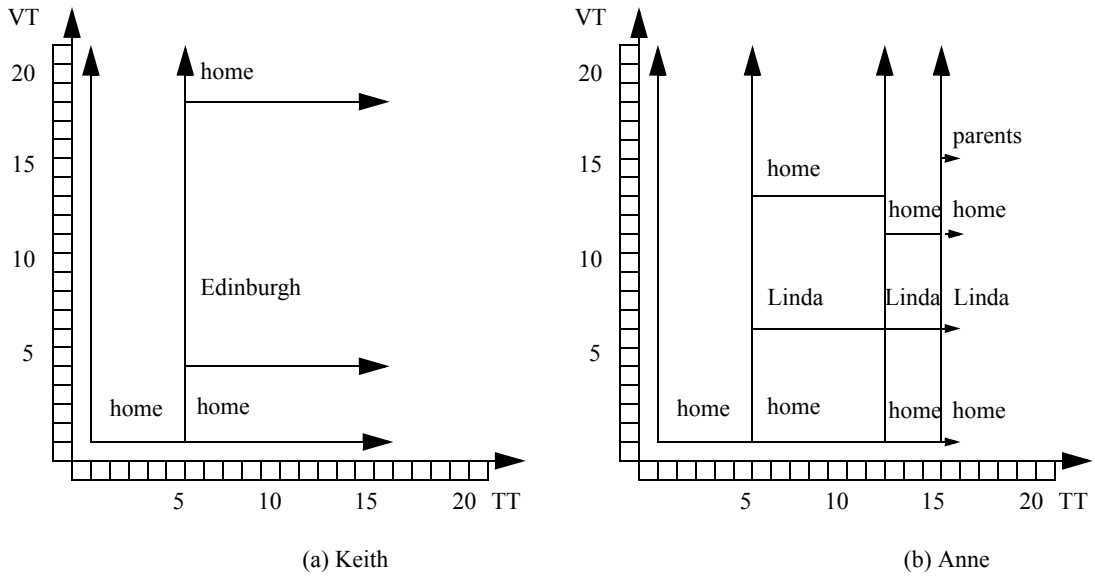


Figure 4: Jennifer's knowledge, December 16

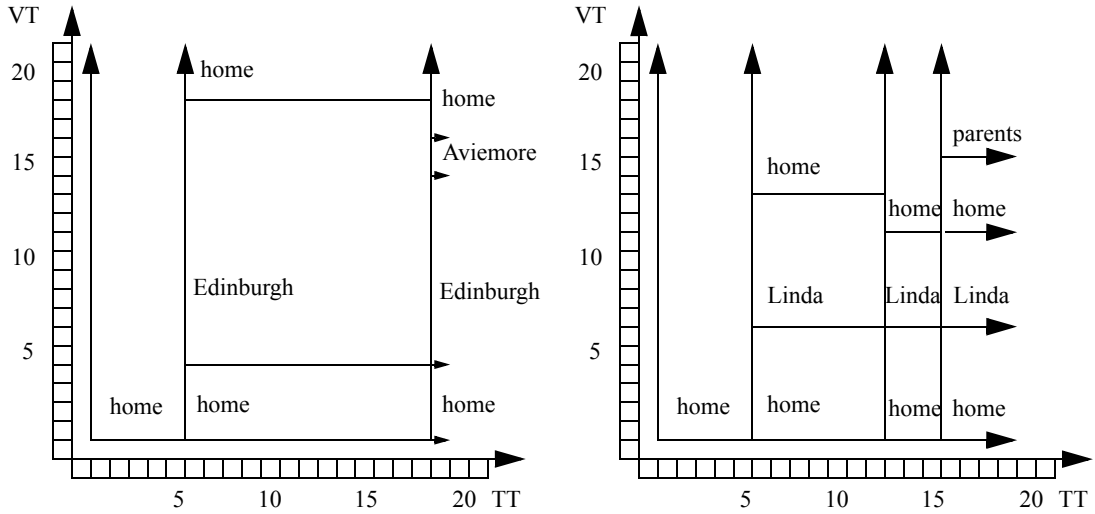


Figure 5: Jennifer's knowledge, December 19

Solution 2.1:

The "discrete" nature of this approach becomes visible in Definition 2.8, which defines an ST-simplex as an ordered pair (S, T) where S is a simplex and T is a BTE. In other words, a specific spatial configuration S exists and is *constant* over a given range T of transaction and valid times. There is no change within the duration of T.

Solution 2.2:

(a)

When was a parcel owned by a specific person not a developed real estate?

We introduce an operation

$$\textit{T-difference}: \quad BTE \times BTE \quad \rightarrow BTE$$

which returns the temporal difference of two BTE values. The query is:

```
SELECT parcel-id, owner,
       T-difference(T-project(area), T-project(building))
         AS undeveloped-area
FROM parcels
```

(b)

When was the school (as the first building) constructed on parcel 4?

We use an operation

$$min: \ BTE \qquad \rightarrow BTE$$

which according to some order on the Cartesian product of periods of a BTE returns the single-valued, minimum BTE of a BTE value. The query is:

```
SELECT min(T-project(building))
FROM parcels
WHERE parcel-id = 4
```

Solution 2.3:

The same task utilizing the snapshot model requires the following three steps:

1. Find the map with the right timestamp in the map sequence.
2. Create a difference map between that map and the preceding map. This difference map represents the "deltas", i.e., it contains the new values in all cells whose values have changed from the preceding map and zero or null in all cells whose values have not changed. During this computation, for each cell check if its contents matches the given value.

This algorithm necessarily requires $n = n_x \cdot n_y$ cell-by-cell comparisons between two adjacent snapshots where n is the total number of cells. This means that the entire task is performed in $O(n)$ time for a complete snapshot image.

Solution 2.4:

The algorithms for queries 1 and 3 are similar to the algorithm for query 2. The algorithm for query 1 outputs all locations of components of the event found for time t_i. The worst time complexity is $O(n_e + c_e \cdot k_c)$ resp. $O((\log n_e) + c_e \cdot k_c)$. The algorithm for query 3 in addition yields the new values of the changed cells. Analogously, the algorithms for queries 4 and 6 are slight variations of query 5 with the worst time complexity $O((\log n_e) + n_f \cdot c_e \cdot k_c)$.

Query 8 asks for a lookup operation and can be answered in time $O(n_e + c_e \cdot k_c)$ resp. $O((\log n_e) + c_e \cdot k_c)$. If the right event has been found, we check all its corresponding components for location (x, y). Similarly, query 9 does the same for a time interval and

has the time complexity $O((\log n_e) + n_f \cdot c_e \cdot k_c)$. The algorithm for query 10 is even more exhaustive. In the worst case, we have to check all n_e events in the event list and for each event all maximum c_e components for location (x, y). Hence, the worst time complexity is $O(n_e \cdot c_e \cdot k_c)$.

Bibliography

Abraham, T., and Roddick, J.F. (1999). Survey of Spatio-Temporal Databases. *GeoInformatica*, 3, 61-99.

Armstrong, M.A. (1983). *Basic Topology*. Springer-Verlag, Berlin.

Bhargava, G. and Gadia, S.K. (1993). Relational Database Systems with Zero Information Loss. *IEEE Transactions on Knowledge and Data Engineering 5* (1), 76-87.

Clifford, J. and Croker, A. (1987). The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans. *Proc. Int. Conf. on Data Engineering* (ICDE, Los Angeles, CA), 528-537.

Croom, F.H. (1978). *Basic Concepts of Algebraic Topology*. Springer-Verlag, Berlin.

Egenhofer, M.J. (1989). A Formal Definition of Binary Topological Relationships. *3rd Int. Conf. on Foundations of Data Organization and Algorithms* (FODO, Paris, France), 457-472.

Egenhofer, M.J. (1991). Reasoning about Binary Topological Relations. *2nd Int. Symp. on Advances in Spatial Databases* (SSD, Zürich, Switzerland), 143-160.

Egenhofer, M.J., Frank, A.U., and Jackson J.P. (1989). A Topological Data Model for Spatial Databases. *1st Int. Symp. on the Design and Implementation of Large Spatial Databases* (SSD, Santa Barbara, CA), 271-286.

Elmasri, R. and Navathe, S.B. (2003). *Fundamentals of Database Systems*. 4th Ed., Addison-Wesley Publ. Co., Reading, MA.

Erwig, M., Güting, R.H., Schneider, M., and Vazirgiannis, M. (1999). Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica 3*, 265-291.

Frank, A.U. and Kuhn, W. (1986). Cell Graphs: A Provably Correct Method for the Storage of Geometry. *3rd Int. Symp. on Spatial Data Handling*, 411-436.

Garcia-Molina, H., Ullman, J.D., and Widom, J. (2002). *Database Systems: The Complete Book*. Prentice-Hall, Upper Saddle River, NJ.

Güting, R.H. (1994). An Introduction to Spatial Database Systems. *VLDB Journal 4* (3), 357-399.

Güting, R.H. and Schneider, M. (1995). Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal 4*, 100-143.

Güting, R.H., de Ridder, T., and Schneider, M. (1995). Implementation of the ROSE Algebra: Efficient Algorithms for Realm-Based Spatial Data Types. *Proc. of the 4th Intl. Symposium on Large Spatial Databases* (Portland, Maine), 216-239.

Härder, T. and Rahm, E. (1999). *Datenbanksysteme: Konzepte und Techniken der Implementierung.* Oldenbourg Verlag, München.

Kemper, A. and Eickler, A. (1999). *Datenbanksysteme*. 3rd ed., Oldenbourg Verlag, München.

Kifer, M., Bernstein, A., and Lewis, P.M. (2005). *Database Systems: An Application-oriented Approach.* Introductory Version, 2nd Ed., Addison-Wesley Publ. Co, Boston.

Kothuri, R., Godfrind, A., and Beinat, E. (2004). *Pro Oracle Spatial: An Essential Guide to Developing Spatially-Enabled Business Applications*. Apress L.P., Berkeley, CA.

Koubarakis, M., Sellis, T.K., Frank, A.U., Grumbach, S., Güting, R.H., Jensen, C.S., Lorentzos, N.A., Manolopoulos, Y., Nardelli, E., Pernici, B., Schek, H.-J., Scholl, M., Theodoulidis, B., and Tryfona, N., eds. (2003). *Spatio-Temporal Databases: The CHOROCHRONOS Approach.* Lecture Notes in Computer Science 2520. Springer-Verlag, Berlin.

Laurini, R. and Thompson, D. (1992). *Fundamentals of Spatial Information Systems*. Academic Press, London.

Özsoyoglu, G. and Snodgrass, R.T. (1995). Temporal and Real-Time Databases: A Survey. *IEEE Transactions on Knowledge and Data Engineering 7* (4), 513-532.

Peuquet, D. (2001). Making Space for Time: Issues in Space-Time Data Representation. *GeoInformatica*, 5*,* 11-32.

Peuquet, D. and Duan, N. (1995). An Event-Based Spatiotemporal Data Model (ESTDM) for Temporal Analysis of Geographic Data. *Int. Journal of Geographical Information Systems* 9 (1), 7-24.

Rigaux, P., Scholl, M., and Voisard, A. (2002). *Spatial Databases: With Application to GIS.* Morgan Kaufmann Publishers, San Francisco.

Sarda, N. (1990). Extensions to SQL for Historical Databases. *IEEE Transactions on Knowledge and Data Engineering 2* (2), 220-230.

Schiller, J. and Voisard, A. (2004). *Location-Based Services*. Morgan Kaufmann Publishers, San Francisco.

Segev, A. and Shoshani, A. (1987). Logical Modeling of Temporal Data. *Proc. ACM SIGMOD Conf.* (San Francisco, CA), 454-466.

Shekhar, S. and Chawla, S. (2003). *Spatial Databases: A Tour.* Prentice-Hall, Upper Saddle River, NJ.

Snodgrass, R.T. and Ahn, I. (1986). Temporal Databases. *IEEE Computer 19* (9), 35-42.

Snodgrass, R.T., ed. (1995). *The TSQL2 Temporal Query Language.* Kluwer Academic Publishers, Boston.

Tansel, A.U., Clifford, J., Gadia, S., Jajodia, S., Segev, A., and Snodgrass, R.T., eds. (1993). *Temporal Databases: Theory, Design, and Implementation*. Benjamin/ Cummings Publ. Co., Redwood City, CA.

Vossen, G. (2000). *Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme*. 4th ed., Oldenbourg Verlag, München.

Wolfson, O., Xu, B., Chamberlain, S., and Jiang, L. (1998). Moving Objects Databases: Issues and Solutions. *Proc. 10th Int. Conf. on Scientific and Statistical Database Management* (Capri, Italy), 111-122.

Worboys, M.F. (1994). A Unified Model for Spatial and Temporal Information. *The Computer Journal* 37 (1), 26-34.

Worboys, M.F. and Duckham, M. (2004). *GIS: A Computing Perspective.* 2nd Ed., CRC Press, Boca Raton, FL.

Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R.T., Subrahmanian, V.S., and Zicari, R. (1997). *Advanced Database Systems*. Morgan Kaufmann Publishers, San Francisco.

Index

R

S

T

U

V

W

Z