A DEDUCTIVE AND OBJECT-ORIENTED APPROACH FOR SPATIAL DATABASES

by

Wei Lu

B.Sc., Zhejiang University, China, 1982

M.Sc., Zhejiang University, China, 1985

M.Sc., Cornell University, 1988

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

© Wei Lu 1993
SIMON FRASER UNIVERSITY
August 1993

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without the permission of the author.

APPROVAL

Name: Wei Lu

Degree: Doctor of Philosophy

Title of thesis: A Deductive and Object-Oriented Approach for Spatial Databases

Examining Committee: Dr. Woshun Luk
Chair

Dr. Jiawei Han, Senior Supervisor

Dr. Tom K. Poiker, Supervisor

Dr. Tom Calvert, Supervisor

Dr. Raymond Ng, External Examiner

Dr. Peter Triantafillou, S.F.U. Examiner

Date Approved: June 16, 1993

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

A DEDUCTIVE AND OBJECT-ORIENTED APPROACH FOR SPATIAL DATABASES.

Author: ______________________

(signature)

Wei Lu

(name)

Aug 6. 1993.

(date)

Abstract

With the rapid development of deductive and object-oriented database technology, it is promising to explore the application of deductive and object-oriented techniques in the development of spatial databases. This thesis investigates the design and implementation of deductive and object-oriented spatial databases (DOOSDB). Several important issues on such spatial databases are studied, including modeling complex spatial objects, spatial data manipulation functionality, a spatial deductive query language, and extensibility of the system. This thesis contributes to the studies on spatial query optimization and processing in DOOSDB in the following aspects: (1) a method for compilation of deduction spatial rules and expressions is proposed with simplification of compiled queries using relational and geo-relational algebra. (2) an algorithm for spatial query plan generation and selection using a dynamic connection graph analysis; (3) techniques for set-oriented optimization and processing of computationally-intensive spatial operators and methods; and (4) a spatial join indexing technique using information associated with frequently used spatial join operations.

This thesis presents an integrated view of a deductive and object-oriented spatial database system and provides an effective mechanism for spatial data handling and efficient algorithms for spatial query processing.

Acknowledgements

I would like to thank my senior supervisor, professor Jiawei Han, for directing my research and for many inspiring discussions. I am also grateful to my supervisor, professor Tom Poiker, for his direction in GIS and for many interesting discussions. Thanks also go to my supervisor, professor Tom Calvert, for his help and discussions about my research.

I also thank professor Beng Chin Ooi of the National University of Singapore and professor Peter Triantafillou for their comments and suggestions that helped improve this thesis.

I would like to thank my fellow students Tong Lu, Ling Liu and Jinshi Xia for their friendship and help. My gratitude goes also to my friends, Goodwin Wang, Karla Cemen, Kay Smedley, Jack Snowden, Dan Fass and Rick Frisent for their understanding and help.

I would like to thank Kersti Jaager for her heartwarming talks and a lot of help.

Thanks go especially to Patrice Belleville for his extremely careful proofreading and many constructive suggestions.

Last but not least, I want to thank my family for their love, encouragement and support.

Contents

Abstract iii

Acknowledgements iv

1 Introduction 1

1.1 Motivations 1

1.2 Problem Specification 3

1.3 System Structure and the Query Interface 4

1.4 Spatial Query Optimization 5

1.4.1 Deductive query compilation and algebraic simplification . . . 5

1.4.2 Access plan generation and evaluation for deductive and object-oriented spatial queries 6

1.5 Information-Associated Spatial Join Index 7

1.6 Thesis Organization 7

2 Related Work 9

2.1 New Generation of Data Models and Database Systems 9

2.1.1 Extended relational database systems 10
2.1.2 Deductive and object-oriented database systems 11
2.2 Query Optimization Techniques 13
2.2.1 Query optimization in relational databases 13
2.2.2 Query optimization in extensible object-oriented database systems 14
2.2.3 Query languages 15
2.3 Spatial Databases 16
2.3.1 Spatial data modeling, manipulation functionality, query languages 16
2.3.2 Spatial system examples 18
2.3.3 Spatial query optimization 18
2.3.3.1 Spatial index 19

3 A DOOS Database System **21**
3.1 Architecture of a Deductive and Object-Oriented Spatial Database . . 21
3.2 Spatial Components and Languages 26
3.2.1 Spatial data representation 27
3.2.2 Spatial operations and procedure definitions 29
3.2.3 Rule definitions 32
3.2.4 Query interfaces 33
3.3 Chapter Summary 36

4 DOOD Spatial Query Compilation **37**

4.1 Compilation of Spatial Rules 38
4.2 Spatial Query Simplification 43
4.2.1 Spatial properties, equivalences and translation rules 44
4.2.2 Derivation of compound relationships 46
4.2.2.1 Disjointness (is_disjoint_from) 47
4.2.2.2 Overlapping (overlaps) 49
4.2.2.3 Inside (is_inside_of) 50
4.2.2.4 Containment (contains) 52
4.2.2.5 Adjacency (is_adjacent_to) 54
4.3 Chapter Summary 57

5 Spatial Query Execution in a DOOSDB **58**

5.1 Dynamic Connection Graph Transformation for Spatial Access Plan Generation 58
5.1.1 Introduction 58
5.1.2 Dynamic connection graph and access plan enumeration . . . 59
5.1.3 Cost estimation and selection of access plans 67
5.1.4 Analysis 69
5.2 Set-Oriented Spatial Computation and Optimization 70
5.2.1 Precomputation and memorization of spatial information . . . 71
5.2.2 I/O control, buffer management, and pipelined processing . . 74
5.2.3 Set-oriented spatial method computation 76
5.2.4 Approximate or alternative operations with reduced complexity 78

5.2.5 Rule-based and spatial semantics-based optimization 80
5.3 Chapter Summary 83

6 Information-Associated Spatial Join Index 84

6.1 Introduction 84
6.2 Distance-Associated Join Indices for Distance Range Search 87
6.2.1 Basic distance-associated join index 88
6.2.1.1 Definition and construction 88
6.2.1.2 Retrieval of spatial objects 90
6.2.2 Ring-structured distance-associated join index 94
6.2.3 Hierarchical distance-associated join index 96
6.2.3.1 Hierarchical DJI retrieval 100
6.2.3.2 Hierarchical structure for shortest distance on a network 102
6.2.4 Distance-associated spatial join index for nonzero-sized spatial objects 105
6.3 Spatial-Information-Associated Join Index with the Orientation Component 107
6.3.1 Basic spatial-information-associated join index 109
6.3.2 Zone-structured spatial-information-associated join index . . . 110
6.3.3 Hierarchical spatial-information-associated join index 114
6.4 Analysis and Simulation Results 117
6.4.1 Analytical model 118
6.4.1.1 Storage requirement 118

6.4.1.2 Processing cost 119
6.4.2 Simulation results 121
6.4.3 Analysis of the simulation results 124
6.5 Chapter Summary and Discussions 126

7 Conclusion **129**

7.1 Summary 129
7.2 Discussion 131
7.2.1 Knowledge discovery in large spatial databases 131
7.2.2 Spatiotemporal databases 134

A BNF of the Query Language DOOSQL **136**

Bibliography **139**

List of Tables

3.1 Spatial relation predicates. 31

3.2 Spatial functions. 31

4.1 Complex spatial relation derivation. 57

6.1 Parameters for performance analysis. 118

List of Figures

3.1 The general system architecture of a DOOS database 23

3.2 A spatial object hierarchy 25

3.3 A map which shows ranches, parcels and polluted areas 35

4.1 A counter-example to the converse of Formula 2 48

4.2 A counter-example to the converse of Formula 3 49

4.3 A counter-example to the converse of Formula 5 50

4.4 A counter-example to the converse of Formula 6 50

4.5 A counter-example to the converse of Formula 7 51

4.6 A counter-example to the converse of Formula 11 53

4.7 A counter-example to the converse of Formula 12 54

4.8 A counter-example to the converse of Formula 13 54

4.9 A counter-example to the converse of Formula 14 56

5.1 Candidate graphs in the enumeration of two access plans. 63

5.2 An access plan tree. 64

5.3 Simulation results for three access plan generation methods. 70

5.4 Derivation of maximum land pieces suitable for planting crops 74

6.1 Indices for three spatial objects. 90

6.2 Processing a spatial range query using the basic DJI. 92

6.3 An example of a ring-structured DJI. 95

6.4 A sample hierarchy for a HDJI 98

6.5 A simple two-level DJI and the index graph. 102

6.6 Search for the shortest distance between two spatial objects. 105

6.7 An example of zone-structured SJI. 112

6.8 A two level hierarchical SJI for a set of objects 116

6.9 Cost curves 122

6.10 Cost curves of spatial-information-associated. 123

6.11 Comparison with other structures 124

Chapter 1

Introduction

1.1 Motivations

A spatial database stores and manipulates data about objects relating to their locations and spatial extensions. Many database applications, such as geographic information systems, engineering databases, medical databases, need to store and access large volumes of spatially referenced data [19, 31, 141, 112, 114, 138]. A spatial database stores large volumes of spatially referenced data which usually have complex structures and require sophisticated data manipulation routines, data modeling tools, compilation, query processing, and indexing methods. Therefore, design and implementation of spatial databases is an important and challenging issue in database research.

A spatial database stores both spatial and non-spatial data (components) of spatial objects. Spatial database operations, such as *union of two polygons*, are in general more costly to process than traditional database operations. Spatial data representation in the database may directly affect the efficiency of query execution. Spatial indices are multi-dimensional and there may be many possible variations for different purposes, such as point location, spatial range search, etc. Spatial indices are usually more difficult to construct and to maintain than those used in relational databases

because they may be multidimensional. Since relational and many geo-relational operations are set-oriented, whereas spatial methods usually compute features for one spatial object at a time (i.e. tuple-oriented), impedance mismatch problems must be solved to achieve reasonable performance. These problems are difficult to handle elegantly and efficiently using the traditional relational database technology [75].

Starting in mid-1980s, deductive database and object-oriented database have been two influential directions in database research [131]. A deductive database system integrates logic programming with relational database technology and constructs a high-level, deductive query interface supported by rules; whereas an object-oriented database system integrates object-oriented programming with database technology and provides us with powerful tools for semantic data modeling, construction of class hierarchy and property inheritance, method manipulation, etc. Furthermore, an integration of the two paradigms leads to a deductive and object-oriented database (DOOD), which has also become a focus in recent research [76, 77]. This trend will undoubtedly influence the development of spatial database systems.

Interestingly, the challenging research issues on spatial data handling demonstrate a high demand for deductive and object-oriented database technologies and their integration in spatial databases. First, many spatial relationships can be expressed concisely and conveniently as logical rules and/or integrity constraints, on which powerful spatial reasoning mechanisms can be developed. A declarative query interface constructed based on deductive database methodology will allow users to define rules and pose queries at a much higher level than primitive representations of spatial objects, release users' burden of understanding and programming low-level primitive spatial data structures and lead to a desirable high-level programming interface [20, 23, 24, 37, 101, 107, 110]. Second, the complexity of spatial data modeling and complex spatial object management can be coped with object-oriented database technology [46, 71, 74]. Many spatial primitives are computationally intensive and are difficult to be defined by pure deduction rules. However, they can be naturally to be defined as methods by spatial functions or procedures implementing geometric algorithms and be associated with classes and class hierarchies. Therefore, a promising

direction in spatial data modeling is an integration of object-oriented and deductive (including extended-relational) methodologies, which leads to a deductive and object-oriented spatial database.

1.2 Problem Specification

The construction, utilization and maintenance of spatial databases, such as a geographical information system (GIS), include the following major tasks.

1. Data collection – A GIS system involves the capturing, transferring, validating and editing of spatial data in order to acquire and load error-free digital data into the GIS [40].

2. Database system design – A GIS system should support complex spatial data modeling, handling and management.

3. Query processing – The task of query processing and data analysis in a GIS is tremendous. Because of the huge volume of spatial data, the efficiency issue becomes more crucial in such a system.

4. Result presentation – The retrieved data will usually be presented to end users in a graphical format. A graphic user interface (GUI) provides an intuitive medium between human and computers [99].

This study concentrates on the second and the third steps in spatial database system design. In this thesis, a deductive and object-oriented spatial database (DOOSDB) is designed, which provides us with complex spatial data modeling functionality, a declarative query interface, spatial data handling functionality, and efficient query processing mechanisms. The DOOSDB spatial data model enhances a spatial database system with deductive and object-oriented features, such as complex objects, class hierarchies, property inheritance, data encapsulation and rules. A dual syntax, either

Prolog-like or SQL-like, is adopted in the specification of spatial rules/relationships and queries. Issues on spatial query optimization and processing in DOOSDB are investigated and spatial query optimization techniques are developed. The study presents an integrated view of a deductive and object-oriented spatial database, and a set-oriented query processing mechanism in such a database system. Next, we present an overview of this study: the DOOS database design, a spatial query language, spatial query optimization and spatial join indices.

1.3 System Structure and the Query Interface

In the DOOSDB, the relational model is extended with deductive and object-oriented features. A spatial database contains spatial data, nonspatial data, deduction rules and geometric procedures. It consists of three components: (i) *GDB* (which stores geometric facts extracted from an image database by preprocessing), (ii) *EDB* (an extensional database [139], i.e. a traditional relational database) and (iii) *IDB* (an intensional database [139], which consists of derived virtual relations defined by deduction rules and geometric procedures). The system is organized into three levels: (i) a *primitive level*, which contains the data about primitive geo-objects extracted from raw image data by image preprocessing techniques, such as edge-detection, line formation; (ii) a *procedural level*, which consists of a set of primitives defined in a procedural language in cooperation with logical and relational operators similar to the geo-relational algebra proposed in [53, 54], and (iii) a *deductive level*, which provides users with new geo-objects and the relationships among geo-objects and other EDB/IDB objects defined by a set of deduction rules.

In such a system, spatial and nonspatial data are effectively modeled. The interaction between a spatial database and a traditional relational database is supported uniformly in our design. Query optimization is the key for a high-performance spatial database. The system is constructed such that optimization can be performed at different levels to achieve maximal efficiency.

An extended-relational and object-oriented framework is adopted for modeling complex objects with class hierarchies, set-valued and list-valued attributes, etc. [61, 113, 121]. Essential spatial data types and spatial data handling operators are provided by the system. Objects are organized in class hierarchies with each class inheriting properties and operations from its super-classes.

An extended SQL-like language, DOOSQL is designed for handling spatial data. It supports (i) complex data modeling, (ii) geometric operations, (iii) spatial rule definitions, and (iv) extension of data types and their associated operations. This spatial database language provides an effective way to model and manipulate spatial data and extensibility for spatial database applications. The detailed design of the system, the data model and the query language will be discussed in Chapter 3.

1.4 Spatial Query Optimization

To ensure efficient evaluation of a high-level query language in large spatial databases, set-oriented query optimization must be explored in DOOSDB. Our study focuses on the performance improvements of spatial query evaluation in the following aspects.

1.4.1 Deductive query compilation and algebraic simplification

A compilation approach is developed to decompose rules and queries into primitives containing no IDB components [92].

Compilation also performs *parameter specification* (denoting parameter types and instantiation requirements referred as modes) consistency checking of rule definition and parameter specifier derivation for IDB predicate parameters.

Algebraic simplification is then performed on the compiled rule expressions based

on relational algebra, geometric algebra and equivalence rules. Some implicit spatial relationships can be derived based on the existing spatial data and geometric properties without spatial object retrieval or geometric computation.

Spatial rule/query compilation transforms a high-level spatial query into a query consisting of only relational primitives, spatial primitives and method calls. Compiled queries may also have some transitive closure operations on those predicates/primitives. A detailed study of spatial query compilation is presented in Chapter 4.

1.4.2 Access plan generation and evaluation for deductive and object-oriented spatial queries

The second step in query optimization is the generation and selection of query access plans for compiled spatial queries. A dynamic connection graph transformation approach is proposed for optimizing compiled spatial query expressions consisting of EDB primitives, spatial primitives, and built-in functions [93]. A dynamic connection graph of a compiled spatial query represents the possible legitimate data flow among the EDB predicates and spatial predicates with the instantiation constraints. A heuristic algorithm is developed for access plan enumeration and selection. A connection graph transformation provides a dynamic picture for spatial query optimization. Suboptimal query access plans can be selected from among the candidate plans generated based on the analysis of the connection graphs.

It is important to optimize the processing of relational and geo-relational operations together with computationally-intensive spatial methods. Since relational and precomputed geo-relational operations are set-oriented, whereas spatial methods usually compute features for one spatial object at a time (i.e. tuple-oriented), impedance mismatch problems must be solved in order to achieve reasonable performance. Set-oriented spatial operation techniques, such as preprocessing and the reuse of preprocessed or intermediate computations, are developed. Examples are used to demonstrate the potential improvements that set-oriented techniques can bring to

spatial operations. Detailed study will be presented in the second part of Chapter 5.

1.5 Information-Associated Spatial Join Index

Spatial indexing is a multi-dimensional task and therefore is more challenging than that for relational databases. Spatial join relationships are especially computationally intensive. A flexible spatial-information-associated join index is developed in Chapter 6 to facilitate dynamic spatial range queries [94]. The idea is to associate some spatial information, e.g. distance measurement (distance-associated join index), with each join index record in order to reduce geometric computations at query processing time. By organizing index records into B+-trees, spatial range queries as well as other distance-related queries can be processed efficiently. Based on this basic distance-associated join index structure, two structured distance join indices, *ring-structured* and *hierarchical*, are proposed to enhance search performance in more sophisticated geometric environments. Other spatial information can also be associated with the join index to improve spatial join operations. Experimental results demonstrate that the precomputation of spatial join indices and their storage may substantially improve the performance of query processing.

1.6 Thesis Organization

This thesis is organized as follows. Chapter 2 contains a review of previous related work on new data models, new database systems, database query languages, query optimization techniques and spatial indexing methods. Chapter 3 presents the design philosophy, system structure and functionality of a DOOSDB system. A declarative spatial query language DOOSQL is developed. Spatial query examples are also presented to illustrate the flexibility and expressiveness of the language. Chapters 4 and 5 address the optimization issues and techniques for spatial databases based on the set-oriented query processing philosophy. Chapter 4 presents an algebraic

optimization approach, namely compilation and simplification of spatial rules and queries. Chapter 5 describes access plan generation and evaluation for compiled spatial queries. A dynamic connection graph technique is developed for generating suboptimal access plans. Set-oriented optimization techniques for object-oriented spatial operations are also presented with illustrative examples. An information-associated join indexing technique for efficient implementation of spatial join operations is developed in Chapter 6. Index construction and retrieval algorithms, variations of the join indices for different applications, complexity analysis and performance studies, etc. are presented. Chapter 7 summarizes this research and discusses future research issues, including knowledge discovery in large spatial databases and the construction of intelligent spatial databases.

Chapter 2

Related Work

In this chapter, we briefly survey previous work related to the development of deductive and object-oriented database systems, including discussions on new data models, new generation database systems, query optimization techniques and spatial database systems.

2.1 New Generation of Data Models and Database Systems

New database applications have become increasingly important in database research [56, 127, 134, 149]. Traditional database systems are inadequate for many new database applications, such as spatial applications where complex data relationships and structures must be handled [30, 75]. There are many approaches to the development of new database systems [18, 133], such as extending existing database systems, e.g. relational databases with new data types and their associated manipulation functionality, etc. Three new data modeling approaches are surveyed here: (i) an extended relational approach, (ii) an object-oriented approach, and (iii) a deductive approach.

These approaches lead to new database systems which support complex objects,

knowledge management, class inheritance, integrity constraints, time traversal and transitive closures. These features are indispensable components of any effective spatial database.

2.1.1 Extended relational database systems

Relational database techniques and query languages are well developed [32, 139]. It is natural to take advantage of existing systems and extend them to serve new applications. Research in this area is exemplified by the following systems.

- Starburst, developed at the *IBM Almaden Research Center*, is an extended relational database system that provides set-oriented operations and a declarative language [56]. User-defined data types and complex objects are supported with rule-based optimization.

- The DASDBS project at *ETH, Zurich* supports complex objects by nested relational schema, set-orientation in spatial data retrieval, communication between different interfaces and multi-transaction management [127]. Due to its extensibility, application-oriented front ends and externally defined types can be implemented. Generic access methods for image objects are also provided.

- POSTGRES is an extension to INGRES for supporting new application databases [134]. The database management system is extended with object management and knowledge management capabilities. User-defined types, functions and active semantics, such as triggers, are supported. Path expressions, nested queries, transitive closure, inheritance, and time travel are also provided. The rule system and the storage management system are used for query optimization.

- The EXODUS project at *the University of Wisconsin* is an extensible system that facilitates the fast development of high performance, application-specific databases [20]. Because of the variety of applications, no single data model can meet all of the requirements of the new application domains simultaneously.

Hence, a reasonable compromise is for the system developers to provide a powerful set of basic building blocks that can be configured, in an extensible way and with minimal efforts, to meet the needs of different applications [20, 47, 89, 127]. EXODUS consists of a set of tools for constructing user friendly front ends, such as a type manager and a rule-based query optimizer.

2.1.2 Deductive and object-oriented database systems

Object-oriented database systems integrate object-oriented programming with database systems to support the new functionality required by applications and to improve database programming productivity. The object-oriented data model supports rich data semantics, class hierarchies, methods, property inheritance, extensibility and persistence. [67, 81, 79]. There are many object-oriented database systems that have been developed as research projects or commercial products, such as GemStone [97], Ontos [6], Orion [12], Iris [142], ObjectStore [106] and O2 [10]. Some of these systems are reviewed next.

- GemStone, from *Servio Logic*, uses a general purpose database programming language, OPAL, as its data definition language (DDL) and data manipulation language (DML). OPAL supports navigational access as well as associative retrieval. Indexing and clustering on objects are also supported for performance tuning.

- Ontos, a commercial product of *Ontologic*, is an object manager that uses C++ as its host language. Persistence is supported through a library of system classes. A generic type is provided to facilitate the translation of memory objects and disk objects.

- Iris at *Hewlett-Packard Laboratory* represents attribute values, relationships and the behavior of objects based on an object and a function model. A rule-based query translator compiles Iris functional expressions into execution trees. The execution trees are optimized using rule-based transformation routines.

- The O2 project built at *Altair* is an object-oriented database system supporting complex objects, object identity, encapsulation, typing inheritance, overriding and extensibility [10]. A user interface generation tool is provided and a programming environment is also supported.
- The Orion project at *MCC* defines a complex object hierarchy as a nested relation. Object-oriented queries can be transformed into corresponding SQL-like relational queries. Objects with simple structures and predicates can be processed more efficiently.
- ObjectStore, a commercial product by Object Design Inc., supports persistency and a C++ programming environment. New functionality, such as collaborative concurrent control and versioning, is provided for database design. ObjectStore focuses on object mapping, caching and clustering techniques for efficient query processing.

A declarative query interface allows the user to pose queries at a much higher level than the primitive level [20, 23, 24, 37, 101, 107, 110], thus releasing the user from the burden of understanding and programming low-level spatial data structures. This is the philosophy of deductive databases [139].

The LDL system is a logic-based database system [28] which supports advanced data and knowledge representation, set operations and recursion using a declarative logic-based language. Some other deductive databases include CORAL at the University of Wisconsin, GlueNail at Stanford University and EKV-1 at ECRC [116]. Some deductive database languages include COL which manipulates complex objects and CRL which works on nested relations [3, 148]. The magic sets, counting and transitive closure algorithms are used to resolve recursion [4, 11, 15, 57, 60, 124]. F-Logic and HiLog are recently proposed in the study of high-order logics and integration of deductive and object-oriented databases [77, 26]. Theses high-order logics can represent features of object-oriented languages, such as inheritance, so that a language with a high-order syntax can be mapped to its first-order semantics.

The integration of deductive and object-oriented database systems becomes an active area in new database design [82]. Research has been done to extend deductive databases with object identity and inheritance [149]. Inference rules and deductive query interfaces are integrated into object-oriented programming systems [145]. Beeri proposed formal models for object-oriented databases [14]. Declarative query languages are presented with SQL extensions, the full calculus and deduction. Functions can be viewed as restricted relations. Abiteboul proposed a deductive and object-oriented database language with a logic-based core language supporting types, objects and extensibility [2]. Grumbach proposed the integration of functions with rewrite rules in Datalog [49]. Deductive and object-oriented databases present a promising direction in new database design.

2.2 Query Optimization Techniques

One of the objectives of query optimization is to minimize the response time for a given query language and mix of query types in a given system environment [72, 133]. Different techniques have been developed for relational systems [85, 88, 129, 136]. Strategies for spatial query optimization have been proposed recently [7, 64].

2.2.1 Query optimization in relational databases

Early topics studied on relational query optimization include equivalence of relational expressions[139], access path selection [129] and query decomposition techniques [144].

Query decomposition is a strategy used for query processing in INGRES [144]. A multi-variable query is decomposed into a sequence of one-variable queries. The execution sequence is determined by estimating the cost using statistics and heuristics. Access plan generation is a process that generates execution plans for a given query so that the plan with the least computational cost may be selected for efficient query execution [68, 129]. A major efficiency concern is join order selection. Even for a

medium size query, e.g. 10−20 conjunctive predicates, the search space is enormous. Heuristics, statistical information, and dynamic programming are used in access plan generation and selection. The optimizer of System R generates access paths by restricting the solution space to binary processing trees and by using dynamic programming for the search [129]. Krishnamurthy, Boral and Zaniolo proposed optimization of nonrecursive queries in deductive databases that has a quadratic complexity [85]. An acyclic recursive query optimization approach has also been proposed [86]. Nested block SQL-like query optimization and global query optimization are studied in [78].

For complex queries in large databases, randomized optimization has been proposed to improve an initial solution until a local optimum is obtained. Typical examples of such an approach include simulated annealing [69] and iterative improvement [136]. Lanzeloott and Valduriez extended randomized and generic search strategies for query optimizers [44, 88].

Parametric optimization generates several execution plans, each of which is optimal for a subset of possible values of run-time system parameters, such as buffer size [70]. Based on two phase randomized optimization algorithms, sideway information passing is used to increase efficiency for new plan generation at the parametric vicinity. Thus, multiple suboptimal access plans can be generated according to different parameters without significant time increase.

2.2.2 Query optimization in extensible object-oriented database systems

Extensibility is a desired feature for new application databases. However, query optimization in an extended system is a challenging task [9, 67, 81, 126]. For examples, information about new operators in object-oriented database systems may not be available. Two general approaches are used: a graph-based approach for algebraic simplification and a transformation rule-based approach for application-dependent optimization. Object clustering is important for object retrieval in such systems.

Greafe and DeWitt [47] proposed transformation rules and implementation rules for reconstructing a query tree and for replacing query language operators with executable methods. An expected cost factor is associated with each rule. The estimated cost is the current cost times an estimated cost factor. At each step of the search, the rule with the least estimated cost is selected. Rule-based optimization is used in many new database systems [13, 56, 142].

Optimization in Orion is based on a query graph [80]. An access plan is generated by ordering the nodes of the query graph. Its two-file approach for retrieval of parts of a complex object is more flexible and efficient than a single file.

Straube and Ozsu proposed a two stage optimization in object-oriented databases [135]. During the first stage, the logical expression is simplified by rewriting rules. During the second stage, each logic operation is mapped to a set of physical data manipulation operations. Access plan selection is based on estimated execution cost.

Efficient clustering is important for efficient object retrieval. Cheng and Hurson proposed an effective clustering schema for complex objects in object-oriented systems [27]. Chan, Ooi and Lu proposed extensible buffer management of indexes [22]. Object clustering is also a major optimization concern in DASDBS and ObjectStore [127, 106].

2.2.3 Query languages

Research on query languages for supporting new applications takes several approaches, including (i) extending a traditional database query language, such as SQL, with new data representation and manipulation functions, (ii) developing logic-based languages, (iii) developing functional languages.

Extended relations with $\neg$1NF nested relational normal form to represent complex objects in new applications [120]. A nested relational algebra and an SQL-like nested query language SQL/NF are also proposed [122].

Pistor and Traunmuellet suggested a database language for sets, lists and tables [113]. The language is based on non-first-normal-form (NF^2). An extended NF^2 algebra is provided. An SQL-like query interface provides a liberal syntax for handling the new data types.

Logic-based Data Language (LDL), designed and implemented at *MCC*, is a deductive database manipulation language which supports advanced data and knowledge representation, set operations and recursion [28].

Han and Li proposed a deductive-ER model and its query language DERDL [61]. The language supports complex objects, such as tuple-valued, list-valued attributes and recursive definitions. Rule definitions use a Prolog-like syntax. The query language has a dual syntax, namely either SQL-like or Prolog-like. It combines the declarative style of relational languages with the expressive power of Prolog.

2.3 Spatial Databases

A spatial database system requires effective data and knowledge representation, a high-level query language and efficient spatial data manipulation functions [34, 92, 101, 108, 125]. Spatial query languages should provide essential spatial data types and commonly used spatial manipulation operators in addition to relational database functionality. We review briefly recent work on the spatial data representation and new database query languages.

2.3.1 Spatial data modeling, manipulation functionality, query languages

Research on spatial data representation has been performed in the fields of computer graphics, geographic information systems, etc. [5, 39, 83, 96, 104]. Spatial object representation includes the topology, geometry and thematic of spatial objects.

Poiker and Christman established an effective data structure, Triangulated Irregular Networks (TIN), for terrain modeling [114]. The data structure interpolates a terrain surface from a set of point data samples from the surface by an effective triangulation algorithm. Other data structures for thematic maps are proposed in [19].

Other frameworks have been presented in the GIS area for a unified representation of geographical phenomena [40, 112]. These studies describe the requirements for geographical information systems and multilevel abstraction of spatial data and its relationships, such as conceptual representation, high-level functional representation, detail implementation format, etc.

In the area of computer graphics, data structures such as winged edge and boundary representation are developed for solid geometric modeling and image synthesis [39, 103, 104]. Bezier patches and B-spline surfaces are used for modeling smooth surfaces, and fractal techniques are applied for terrain simulations [17].

Güting proposed a geo-relational algebra language (Gral) which extends relational algebra by integrating geometric data types and operations [53]. Most commonly used topological relations, geometric object generation functions and geometric measurement functions are included in Gral. Geo-relational algebra is aimed at providing the functionality needed to handle spatial data.

Egenhofer summarized the functionality requirements for a spatial system [34], which include abstract data types with corresponding operations, display mechanism for visualization, pointing devices for extended dialog, legends for maps, windowing operations, etc. Spatial data operations involve procedural and complex computations. Interactive query mechanisms and SQL-like extensions for spatial query languages have been explored [23, 121].

Efficient geometric routines are crucial to an operational spatial database system. Many spatial algorithms have been developed in the field of computational geometry, such as Voronoi diagrams for spatial location [25, 33, 111, 115, 137]. An extended spatial system extracts spatial data from satellite image data. Image enhancing and object extraction techniques are used for image processing [146, 45, 111].

2.3.2 Spatial system examples

ARC/INFO at ESRI is a commercial geographic information system [102]. It employs a combination of traditional geographic data handling techniques and relational database techniques to handle geographic data. A relational DBMS is used to handle non-spatial information. Spatial data is processed by the specialized procedures.

The PROBE project at the *GTE Lab* and the *IBM Almaden Research Center* is a system gearing for spatial applications [110]. Z-ordering is used for linearizing two-dimensional bitmap objects by a curve passing through a full plane. Z-ordering indexes are effective for accessing image objects, and for performing operations on them, such as spatial join.

System Gral extends a relational system for spatial applications with geometric algebra which provides geometric types and their manipulation functions [54]. An algebraic language is used for querying. Queries are translated into their equivalent procedure sequences. Rule-based query optimization is systematically developed [13].

Oosterom and van de Bos presented an object-oriented approach for the design of GIS [141], which explores data abstraction, extensibility and software reuse for the implementation of new application systems. Luk and Choi proposed a generic object-oriented spatial database which can be extended into domain-specific database systems by building additional software layers on top of it [29].

It is essential to model complex spatial objects and relationships among them with a spatial query language. There are some extensions of SQL for spatial applications such as PSQL [123], and GEOQL [108], which extend SQL with spatial data types and manipulation functionality.

2.3.3 Spatial query optimization

Spatial query optimization is a difficult task due to the complex spatial data type and sophisticated spatial manipulation functions. Rule-based optimizations and strategies

have been applied in some spatial database systems [13, 134]. We review some query optimization examples in spatial databases.

Ooi proposed extended decomposition techniques for spatial query optimization in [108]. His thesis proposed a global optimization strategy for an extension of SQL which requires additional indexing structures to materialize the additional relationships.

Optimization in Gral is based on optimization rules[13, 127]. Geo-relation algebra is translated into a geo-relational database operation sequence. A query can be specified as a sequence of operations and each operation is defined by its collection of rules together with the selected control strategy. Aref and Samet presented a set of strategies for spatial query optimization [7] which can also be specified in the form of optimization rules.

Optimization in PROBE utilizes Z-ordering to reduce a two-dimensional problem to a one-dimensional problem [109]. The transformed problem can be dealt with traditional techniques. This technique is effective for image objects.

To assess spatial relationships among generated objects, one technique is to derive these relationships based on the known relationships among the component objects. Egenhofer proposed a matrix method for spatial reasoning [35]. If A and B satisfy one relationship, and B and C satisfy another, some possible relationships between A and C can be inferred from the existing relationships without spatial computation.

Data in DASDBS is stored linearly with an index associated with each complex record. Spatial data is clustered with its geometric neighbors [127, 143]. An access manager is designed for managing spatial accesses. The clustering of spatial objects improves spatial query processing efficiency.

2.3.3.1 Spatial index

Spatial databases have been widely used in geographical applications, engineering applications, and many others. Spatial indexing mechanisms are essential for processing queries involving spatial search. R-trees [55], R+-trees [130], Quad-trees [125], K-D-B

trees [118], and Grid files [105], among others, have been popular as indexing structures for spatial object retrieval [48, 50, 65, 110, 132, 128]. There are also structures that can be used for multidimensional attribute indexing [66, 91]. An index for large extended objects was proposed in [52].

Some research has been done for rectangular range searches, such as the range tree [16]. Circular range search has been approximated by rectangular range search [115]. In order to retrieve objects in a distance range, a rectangle circumscribing this range is searched. The search result needs to be further tested for the original range. K-D trees and range trees are used for rectangular range search.

Join indices were first developed by Valduriez to enhance the performance of join operations in relational databases [140]. Join indices reduce the number of I/O operations needed and thus improve the performance of join operations. As an extension to join indices for spatial database applications, Rotem proposed a spatial join index structure which converts geometric computations of certain spatial relationships into simple spatial join index files [119]. Join indices store spatial object identifier pairs for those objects having these spatial relationships. Queries involving spatial joins can be processed by retrieving spatial join indices rather than performing geometric algorithms. Furthermore, queries related to fixed distances can be processed by constructing spatial join indices based on ϵ-overlap, where ϵ is a fixed distance defined by the database designer.

This chapter presented a brief review of new database designs, query optimization with emphasis on query languages and spatial databases. Many successful database systems combine various techniques to provide effective data modeling and manipulation power for new applications.

Chapter 3

A DOOS Database System

Based on previous studies, we present an overview of the proposed DOOS database system in this chapter. A user friendly interface for spatial data modeling and manipulation is also provided.

3.1 Architecture of a Deductive and Object-Oriented Spatial Database

An object-oriented system provides rich semantic modeling power and extensibility which are essential for a spatial database system. Databases using the logic programming paradigm can be a natural evolution from relational databases [43, 84, 90, 98]. The logic approach to databases has a number of advantage, such as it has a sound underlying theory and the language in first-order logic proof theory is richer than its counterpart in relational theory. The combination of object-oriented and deductive methodologies can provide the strengths of both approaches and achieve a deductive and high performance system. With deductive techniques developed, such as transitive closure algorithms, recursive queries can be computed efficiently.

A Deductive and Object-Oriented Spatial (DOOS) data model was proposed in our

study [92], which models complex spatial objects and supports a high-level deductive query interface. A DOOSDB system is aimed at spatial applications, such as GIS. The system adopts successful features from both object-oriented programming and logic programming, and develop a set of techniques to alleviate the impedance mismatch problem in integrating the relational technology with spatial application methods.

In our design, a DOOSDB contains spatial data, nonspatial data, deduction rules and computationally-intensive methods. It consists of three major components: (i) *GDB*, which stores spatial facts extracted from an *image database* by preprocessing, (ii) *EDB*, an extensional database [139], which stores nonspatial data in a relational form, and (iii) *IDB*, an intensional database [139], which consists of virtual relations defined by deduction rules and spatial computational routines referred to as *methods*.

The DOOSDB system supports a spatial database with a high-level query interface. The syntax of the interface may be either an SQL-like query language or a Prolog-like query language. High-level primitives are defined by deduction rules or computational routines. For efficient processing, deduction rules are precompiled, system supported spatial function are optimized, and the general control structures of the methods are analyzed and stored as well. Extensibility is supported. Moreover, the system analyzes and collects the database statistics and other meta-knowledge in order to assist in query optimization. Figure 3.1 outlines the general architecture for query processing and optimization in the DOOS database.

Our design of spatial object storage structures adopts the SAND (Spatial And Nonspatial Data) architecture developed by Aref and Samet [7] (also in [107]), in which spatial information and corresponding nonspatial information are stored separately and linked together via forward and backward links.

Suppose a collection of objects, O, is referred to by the pair $\langle R, S\rangle$ where R is a relation that stores nonspatial attribute instances of O, and S is a spatial data structure that stores the spatial attribute instances of O. Notice that spatial and nonspatial components should be kept synchronized through all operations. For example, given a pair $\langle R, S\rangle$, op_r (e.g. selection) or op_s (e.g. windowing) should return

the pair $\langle R_1, S_1 \rangle$ instead of just R_1 or S_1. That is, relational-based and spatial-based operators will be extended in the following way:

$$x_{op_r}(\langle R, S \rangle) = \langle op_r(R), sp_{extract}(op_r(R), S) \rangle. \tag{3.1}$$

$$x_{op_s}(\langle R, S \rangle) = \langle db_{extract}(R, op_s(S)), op_s(S) \rangle. \tag{3.2}$$

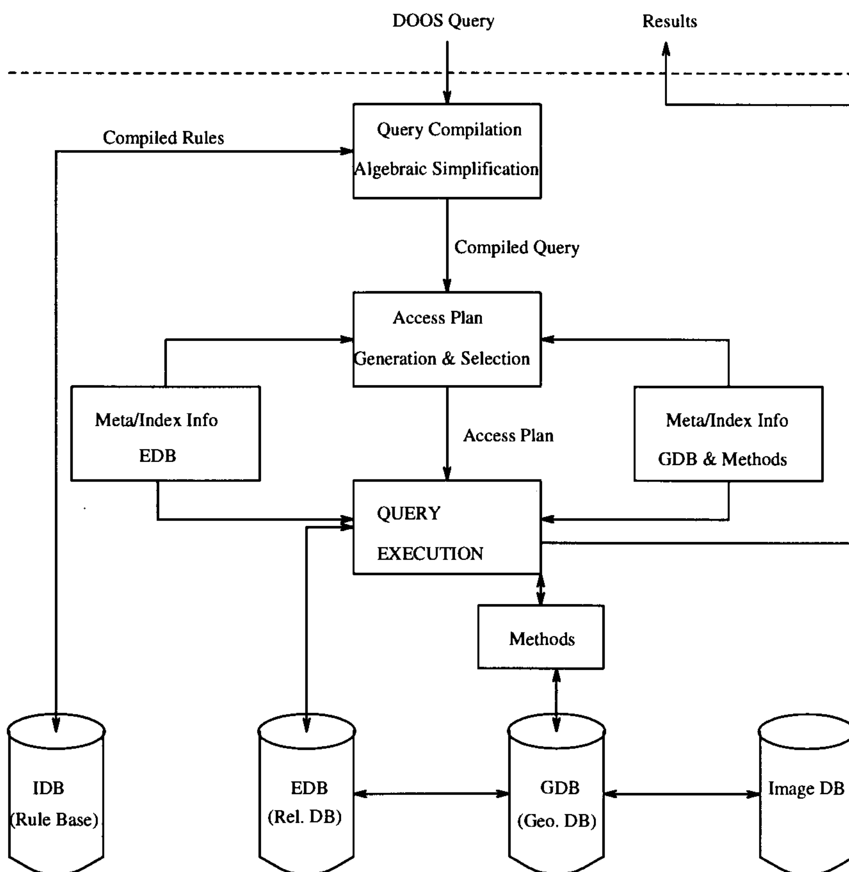


Figure 3.1: The general system architecture of a DOOS database

Note that equation (3.1) indicates that a relational-based operator $x_{op_r}(\langle R, S \rangle)$ is performed by first extracting nonspatial data using the relational operator $op_r(R)$ and

then extracting the corresponding spatial portion by $sp_{extract}(op_r(R), S)$. Similarly, we have a spatial-based operator $x_{op_s}(\langle R, S \rangle)$ defined in equation (3.2).

Using the SAND architecture, a set of interesting query processing strategies has been developed for spatial query processing [7]. For instance, consider the optimization of the implementation of a spatial join, such as *is_adjacent_to, contains*, where a spatial join combines related entities from two spatial entity sets into a single entity set. The sequence of relational-based and spatial-based operations can be reordered to facilitate the merge of joined objects. A join can be performed based on the intersection of pointers (tuple_ids or spatial_ids). Relational operators can be pushed into $db_{extract}$, and spatial operators can be pushed into $sp_{extract}$ to reduce the size or number of objects to be worked on. Some intermediate results can be pipelined with subsequent operations to save the creation of temporary relations or the corresponding data structures. Under certain circumstances, subsequent operations can work directly on some temporary intermediate structures instead of creating new copies of the data. Projections can be performed as early as possible, especially when the target list contains only nonspatial attributes or only spatial attributes. When two spatial or relational operations refer to the same spatial or relational attribute, only one $sp_{extract}$ or $db_{extract}$ need be performed. These optimization techniques have been discussed in [7] and will be applied by our optimizer when possible.

Apart from being organized according to other class hierarchies and property inheritance rules common in object-oriented database systems, spatial objects are also organized into hierarchies in the DOOSDB system. The primitive spatial types are POINT, LINE and POLYGON. A (super-)class can be constructed by combining several existing classes, such as GEO whose subclasses are the combinations of POINT, LINE and POLYGON. By constructing the object hierarchy, properties and methods defined for a class can be inherited by its subclasses. For example, *geo_intersection* is defined on two object instances of type GEO, hence it is applicable to any objects that belong to subclasses of GEO, e.g. intersecting a region (typed **POLYGON**) and a highway (typed **LINE**). Hierarchies are also used in spatial rule compilation to check predicate parameter type consistency and derive rule parameter types.

Conceptually, a DOOS database contains a collection of persistent spatial and nonspatial objects which belong to classes (and which are in turn organized into class hierarchies) in a database schema. The root of the class hierarchy is a special class "*Object*" which contains the common methods for all kinds of objects, such as "*create_class*", etc. Each class is associated with a set of attributes and/or methods which are defined by deduction rules, computational routines, property inheritance rules, class composition (aggregation) hierarchies, class associations, or concrete values. An example of the schema outline of a DOOS database is presented in Figure 3.2, in which the class hierarchy/association information is defined as follows.

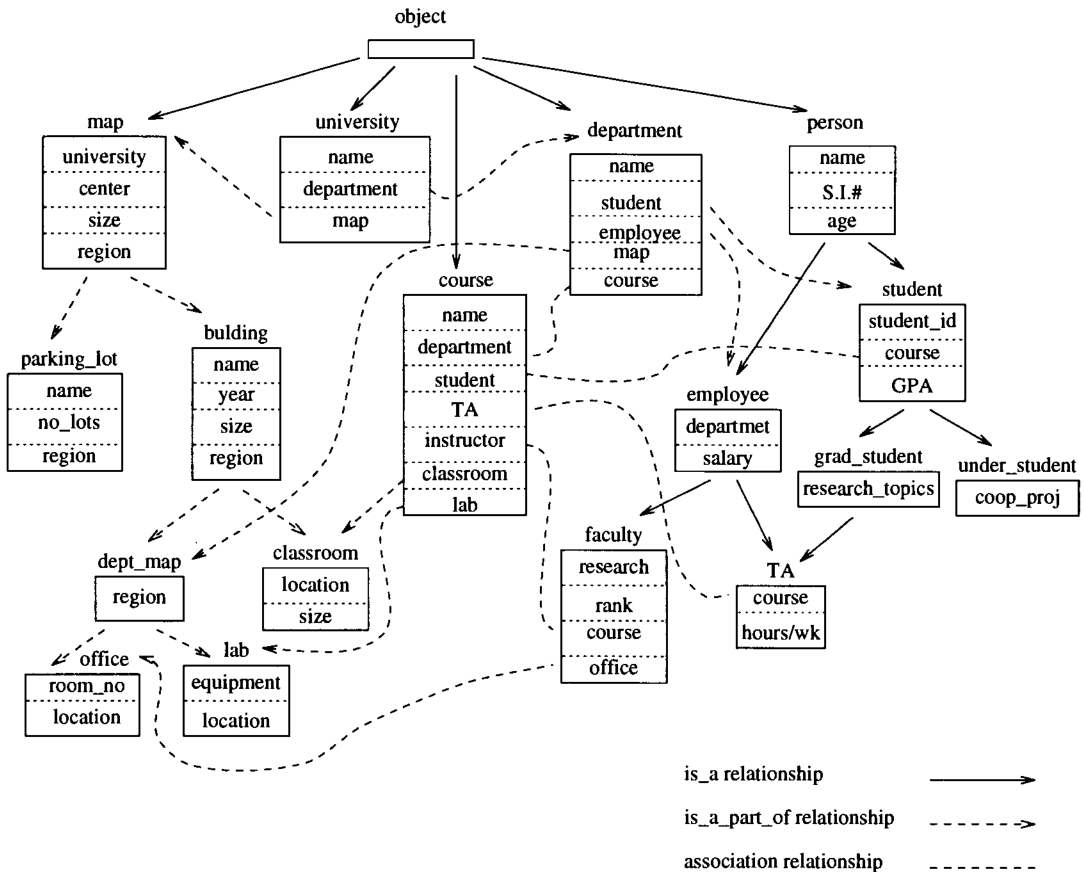


Figure 3.2: A spatial object hierarchy

1. *is_a* relationship (class/subclass hierarchy) is defined by a solid arrow in which the class at the arrow tail is a superclass of the class pointed by the arrow head. For instance, class *employee* is a superclass of class *faculty.*

2. *is_a_part_of* relationship (class composition hierarchy) is defined by a dashed arrow in which the attribute at the arrow tail is a description of the "component" class pointed by the arrow head. For example, class *department* is a component (attribute) of class *university.*

3. *class association relationship* is defined by a dashed line where two classes refer to each other. For example, the attribute *course* in the class *department* is associated with the attribute *department* in the class *course.*

In summary, a deductive and object-oriented spatial system provides a uniform high-level interface to users. It supports data semantic modeling and spatial functionality.

3.2 Spatial Components and Languages

A high-level spatial database query language should provide spatial object modeling functions, rule definition capability for expressing complex spatial relationships and a declarative query interface. We now propose a deductive and object-oriented spatial query language DOOSQL to facilitate high-level spatial queries. This language extends DERDL [61] and SQL/NF [122] with spatial data modeling and manipulating functions. It has the following features, (i) a nested relational framework for representing complex objects which supports tree-structured schema, set attributes and list-valued attributes, (ii) an object organization into a class hierarchy with inheritance, (iii) a rule definition language that uses in a Prolog-like syntax, (iv) a query interface with dual syntaxes, i.e. SQL-like or Prolog-like.

3.2.1 Spatial data representation

Our nested relational framework is based on the non-first-normal-form relation($\neg 1NF$) proposed in [120]. A nested relation R is of the form R = ($R_1, \ldots, R_i, \ldots, R_n$), where attribute R_i can be either an atomic attribute or a nested relation. A nested relation is in a partition normal form(PNF) if all atomic attributes at the external level are a key of the relation and if all of its sub-relations are in PNF [120]. A nested relation in partition normal form has the nice property that the nesting/unnesting operations are reversible. We will assume that all relations are in PNF. A nonrecursive relation can be represented as a schema tree.

A nested relational framework can accommodate list-valued attributes and set-valued attributes. A set-valued attribute is defined by the keyword **setof** while a list-valued attribute is described by the keyword **sequenceof**. A tuple-valued attribute can be referred to at either the attribute level or the component level. Sets and lists may optionally have a name.

The followings are some meta-symbols for defining the query language in extended Backus Normal Form (BNF).

::= defines non-terminal symbol
⟨ ⟩ denotes a non-terminal symbol
[ ] denotes an optional component of the language that may appear at most once.
{ } for an optional component of the language that may appear any number of times.

⟨*schema*⟩	::=	**schema** ⟨*schema_body*⟩
⟨*schema_body*⟩	::=	⟨*name*⟩ (⟨*attri_def*⟩ {, ⟨*attri_def*⟩ })
⟨*attri_def*⟩	::=	⟨*atom_attri_def*⟩ | **setof** ⟨*subschema*⟩
		| **sequenceof** ⟨*subschema*⟩
⟨*subschema*⟩	::=	⟨*type*⟩ | ⟨*schema_body*⟩
⟨*atom_attri_def*⟩	::=	⟨*name*⟩ : ⟨*type*⟩

In addition to the basic attribute types in the traditional relation, namely *REAL, BOOLEAN, INT* and *STRING*, a set of essential spatial types are supported in the extended schema. Here is a part of syntax for object definition.

Example 3.1 Spatial primitive definitions.

Typical primitive spatial data types/objects provided by the DOOS database system are *POINT*, *LINE* and *POLYGON*.

1. **POINT** represent a point, i.e. a pair of real numbers in two dimensional space:

   schema POINT(x: REAL, y: REAL)

   For example, the center of an object can be represented by the point (2, 3).

2. **LINE** is used to describe a line consisting of a number of line segments and a line is represented by a sequence of points:

   schema LINE(points: **sequenceof** POINT)

   The key word **sequenceof** indicates that *points* is a list-valued attribute. For example, an instance of a line can be represented by ((2, 3), (6, 10)).

3. **POLYGON** is used to describe a simple polygon by listing its contiguous vertices:

   schema POLYGON(points: **sequenceof** POINT)

   A polygon data object is assumed with the last point on the list connected to the first one on the list. An instance of a POLYGON is ((0, 3), (3, 3), (3, 0)).

These primitive spatial data types can be used to build more complex spatial objects. A sample object schema definition is presented in Example 3.2.

Example 3.2 A complex spatial object *region* is defined as follows.

schema *region*(*name* : *STRING*,
population : *INT*,
geo : *POLYGON*,
setof *highways*(*name* : *STRING*,
route : *LINE*)
setof *districts*(*name* : *STRING*,
area : *POLYGON*)
)

Notice that in this definition, *highways* and *districts* are two set-valued attributes. □

3.2.2 Spatial operations and procedure definitions

Relational operations, such as *selection*, are extended to manipulate these newly added primitive spatial data types. Equality is extended for new primitive spatial structures as follows.

1. **Structure and value equality** of two non-atomic attributes indicates that both have the same structure and that corresponding components have the same values. For example, the equality of two points p_1 and p_2 is defined as,

$$p_1 = p_2 \iff (p_1.x = p_2.x) \land (p_1.y = p_2.y)$$

2. **Semantic equality** refers to the semantic equivalence of two representations for the same spatial object; these may not be necessarily structure or value equivalent. Here are some examples.

   a) LINE equality

   Given two undirected lines $l_1(p_{11}, \ldots, p_{1i}, \ldots, p_{1n})$ and $l_2(p_{21}, \ldots, p_{2j}, \ldots p_{2m})$,

$$l_1 = l_2 \iff (n = m) \wedge (\forall_{i:\ 1 \leq i \leq n}\ (p_{1i} = p_{2i})).$$

b) POLYGON equality

Given regions $r_1(p_{11}, \ldots, p_{1n})$ and $r_2(p_{21}, \ldots, p_{2m})$,

$$r_1 = r_2 \iff (n = m) \wedge (\exists_{i:\ 1 \leq i \leq n}\ \forall_{j:\ 1 \leq j \leq n}\ (p_{1i} = p_{2\ mod(i+j,n)+1})).$$

A spatial database system should provide basic manipulation operations on geometric objects that include (i) *logic operators*, which describe the relationship among geometric objects, for example *is_inside_of(X, Y)* which returns **TRUE** if X is inside of Y, (ii) *geometric transformation operators*, which take geo_objects as parameters and create new geometric objects, for example, *geo_union(X, Y)* which computes the union of X and Y , (iii) *feature evaluation functions*, which evaluate properties of the geometric objects, for example, *distance(X, Y)* which calculates the distance between the two points X and Y, and (iv) *aggregation functions*, which calculate aggregation value of a set of data, for example, *sum* computes the sum of a set of numbers. Some typical geometric operators and their parameter specifiers are listed in Tables 3.1 and 3.2. The names of the predicates and functions are self-explanatory. A function can be converted to the equivalent predicate [63]. In the following discussion, a spatial function, *geo_func*, will be referred to as $geo_func(\vec{X})$ or its corresponding predicate form $geo_func(\vec{X}, Y)$, where $\vec{X}$ is the input vector and Y is an output parameter that will contain the function value. Similarly, spatial predicates are in an infix format in an SQL-like query (to simulate natural language) and are in a prefix format in a Prolog-like query (to be consistent with Prolog). Commonly used aggregation functions include *sum, minimum, maximum, average* and *count*.

User-defined or application-specific procedural methods, such as the maximum throughput of a highway network, are very important in an extensible spatial database system for new applications since it needs to be adaptable to different customers.

An important distinction between a procedural primitive and a relational one lies in the specification of the *application modes* of the parameters in a procedure and

Parameter types	Predicate	Parameter types
GEO	overlaps	GEO
GEO	is_disjoint_from	GEO
GEO	is_inside_of	POLYGON
POLYGON	contains	GEO
POLYGON	is_adjacent_to	POLYGON

Table 3.1: Spatial relation predicates.

Function (parameter types)	Function type
length(LINE)	REAL
area(POLYGON)	REAL
boundary(POLYGON)	LINE
distance(POINT, POINT)	REAL
geo_union(POLYGON, POLYGON)	POLYGON
geo_intersection(POLYGON, POLYGON)	POLYGON
geo_difference(POLYGON, POLYGON)	POLYGON

Table 3.2: Spatial functions.

the attributes in a relation. Every attribute in a relation can be instantiated and inquired at will. However, a parameter in a procedure is often restricted to some specific accessing mode(s), i.e. either instantiation only, denoted as **in** mode; inquiry only, denoted as **out** mode; or both, denoted as **any** mode. For example, a geometric procedure $geo_union(X, Y, Z)$ returns the union of two polygons X and Y in Z. X and Y should be instantiated (with the mode **in**) and Z could be either instantiated or inquired (with the mode **any**). Otherwise, if X and Z were instantiated but Y were inquired, an infinite number of Y's could be derived since X and Y may partially overlap. Notice that the mode of a parameter of a finite relation is always **any**.

Our system supports the extension to new data types and new operations. The declaration of a spatial procedure includes: a procedure name, procedure parameters, and their parameter specifiers. The syntax of procedure declarations is presented below and examples of procedure declarations will follow.

$$\langle procedure \rangle ::= \textbf{procedure} \langle name \rangle \; (\langle para_def \rangle \; \{, \langle para_def \rangle \; \} \;)$$
$$\langle para_def \rangle ::= \langle Name \rangle : \langle para_spec \rangle$$
$$\langle para_spec \rangle ::= \langle type \rangle \; \langle mode \rangle$$

Example 3.3 Procedures *boundary* and *gMaxUnion* are defined as follows.

procedure boundary(X: POLYGON **in**, Y: LINE **out**)

procedure gMaxUnion(X: **setof** GEO **in**, Y: **setof** GEO **out**)

When a new procedure is defined, the system will register the procedure name and its parameter names with their specifiers. A user-defined procedure is treated in the same way as one defined by the system. For instance, *boundary* could be a system defined procedure, which takes a polygon and returns its boundary. A user-defined procedure, such as *gMaxUnion*, will be imported in the system by directly linking compiled code blocks with the system or by interpreting them. □

These operators inject a lot of vital power to the relational system for inquiring on and manipulating spatial information.

3.2.3 Rule definitions

Many spatial relationships, such as containment, within, connected-to, etc., are defined recursively. It is necessary to use recursive rule compilation techniques in a DOOS database. The syntax of the rule definition in DOOS is similar to that in Prolog. Examples of spatial rule definitions will be presented.

$$\langle rule \rangle ::= \langle predicate \rangle :- \langle predicate \rangle \{, \langle predicate \rangle \}.$$
$$\langle predicate \rangle ::= \langle name \rangle (\langle Name \rangle \{, \langle Name \rangle \})$$

Example 3.4 Definitions of IDB predicates.

Let $rel_inside(X : GEO, Y : POLYGON)$ be a precomputed relation representing the fact that X is inside of Y. The IDB predicate $within(X, Y)$ is defined below.

$inside(X, Y)$:- $is_inside_of(X, Y)$.

$inside(X, Y)$:- $rel_inside(X, Y)$.

$within(X, Y)$:- $inside(X, Y)$.

$within(X, Y)$:- $inside(X, Z), within(Z, Y)$.

Here the new predicate *within* is defined by a relation *rel_inside*, a geometric predicate *is_inside_of* and their transitive closure. An application-oriented predicate can also be defined. If an object class $pollution_map(X : POLYGON)$ is defined in the database, a predicate $polluted_parcel_in(X, Y)$ indicating that X is polluted land parcel and is inside of Y can be defined as below.

$polluted(X)$:- $pollution_map(Y)$, $overlaps(X, Y)$.
$polluted_parcel_in(X, Y)$:- $parcel(X)$, $polluted(X)$, $within(X, Y)$.

The parameter specifier of the IDB parameters is derived from those that define them by a rule compilation process which will be illustrated in the next chapter.

3.2.4 Query interfaces

The dual interface of DOOSQL is similar to that of the Deductive-ER query language [61]. A query can be posed using an SQL-like syntax or a Prolog-like equivalent form. The SQL-like query syntax is as follows.

⟨*sql_query*⟩	::=	**select** ⟨*result*⟩ {, ⟨*result*⟩ }
		from ⟨*name*⟩ {, ⟨*name*⟩ }
		where ⟨*pred_expression*⟩
⟨*result*⟩	::=	⟨*attribute*⟩ | ⟨*func_name*⟩ (⟨*attribute*⟩ {, ⟨*attribute*⟩ })
⟨*pred_expression*⟩	::=	⟨*pred_term*⟩ { ⟨*logic_link*⟩ ⟨*pred_term*⟩ }
⟨*logic_link*⟩	::=	**and** | **or**
⟨*pred_term*⟩	::=	[**not**] ⟨*predicate*⟩ |
		[**not**] (⟨*math_expression*⟩ ⟨*comp*⟩ ⟨*math_expression*⟩) |
		[**not**] (⟨*geo_obj*⟩ ⟨*geo_predicate*⟩ ⟨*geo_obj*⟩)

Example 3.5 A spatial query in an SQL-like language.

We examine the database illustrated in Figure 3.3, which consists of the following objects: districts, ranches, parcels and pollution maps, which are defined as follows.

schema $parcel(name : STRING, region : POLYGON)$,
$district(name : STRING, region : POLYGON)$,
$ranch(name : STRING, polluted : BOOLEAN, region : POLYGON)$.

Suppose that the query is to *"find the total area of unpolluted ranches which are adjacent to polluted parcels in district A"*. In combination with the power of deduction and an easy-to-read SQL-styled language, this query can be posed effectively. The predicate *is_adjacent_to* is commonly used, therefore it is assumed to be supported by the system. An IDB predicate $polluted_parcel_in(X, Y)$ is defined in Example 3.4.

The query in SQL-like format is:

select $sum(area(ranch.region))$
from $ranch, parcel, district$
where $district.name = 'A'$
and $ranch.polluted =$ **FALSE**
and $polluted_parcel_in(parcel.region, district.region)$
and $ranch.region$ *is_adjacent_to* $parcel.region$

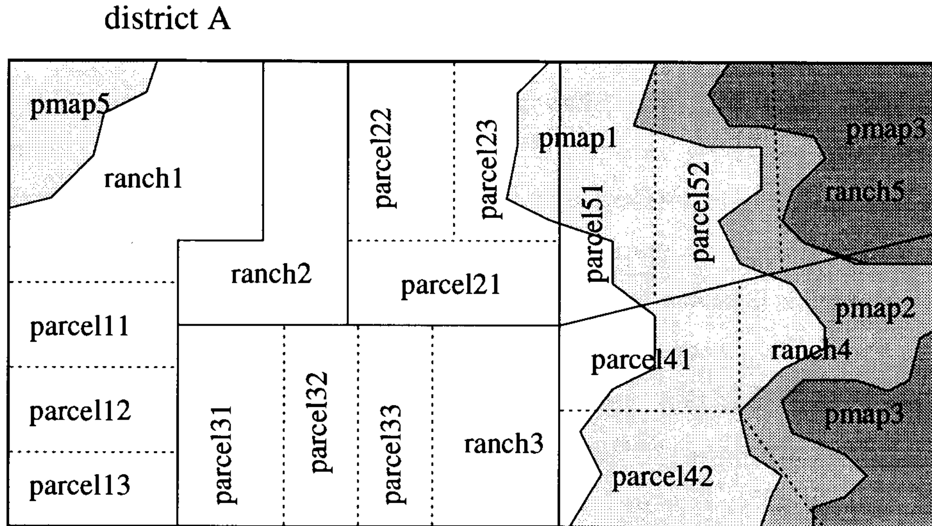


Figure 3.3: A map which shows ranches, parcels and polluted areas

As a result of the execution of this query, the areas of $ranch_3$, $ranch_4$ and $ranch_5$ are returned. □

It can be a good option sometimes to adopt the Prolog-like syntax in the presentation of a query. The extended BNF of a query using the Prolog-like syntax is presented as follows.

$$\begin{aligned}
\langle logic_query \rangle &::= \; ?- \langle predicate \rangle \{, \langle predicate \rangle \}. \\
\langle predicate \rangle &::= \langle name \rangle (\langle parameter \rangle, \langle parameter \rangle) \\
\langle parameter \rangle &::= \langle constant \rangle \mid \langle Variable \rangle
\end{aligned}$$

A deductive and object-oriented spatial language has been outlined, data structures have been developed and data manipulation functions have been proposed. The compilation of a spatial query transforms a query in either SQL-syntax or Prolog-syntax into a uniform compiled equivalence. The language structures involving definition and data retrieval are specified by their BNF in Appendix A.

3.3 Chapter Summary

An integration of the deductive and object-oriented paradigms in the design of spatial databases has been proposed in this chapter. The spatial database provides a declarative query interface which can be compiled and decomposed using well-developed compilation approaches. Such a design and implementation philosophy of spatial databases combines the advantages of both logic and procedural methods, thus creating a user-friendly environment and achieving both processing efficiency and expressiveness. The system is geared to spatial applications and therefore it can be more effectively and efficiently for spatial queries. The system supports both built-in geometric data types with standard spatial operators and user-defined data types with associated spatial methods. The former can be processed efficiently while the later is indispensable for flexibility. An extended SQL spatial query language has been designed to support spatial data types, their manipulation functions, rule definitions, and extensibility.

Chapter 4

DOOD Spatial Query Compilation

A high-level deductive query interface has been implemented efficiently using the compilation approach [139]. The compilation process can be divided into three phases: (i) query independent IDB rule compilation, (ii) system independent optimization, which includes the compilation of a deductive spatial database and algebraic simplification, and (iii) the system dependent optimization of spatial queries. The first phase compiles the IDB predicates defined in the IDB into operations on the EDB, the GDB and their transitive closures. Simplification and optimization can be performed on the compiled results. The second phase is invoked when a query is submitted to the system; it compiles and simplifies a specific spatial query using information about query instantiation. The third step is the continuation of the second phase which analyzes and optimizes a specific spatial query based on information about the query instantiation, inquiry, compiled geo-primitives, and statistical information about the EDB and the GDB. The result is an optimized query processing plan which is then submitted to the query processor. We examine system independent optimization, namely the first phase and the second phase, in this chapter and leave the discussion of the third phase to the next chapter.

Compilation of the IDB is a process which transforms rules into a form containing only relational and spatial primitives and method calls. Compilation of the IDB

has been studied extensively in deductive database research. Detailed compilation and optimization techniques can be found in previous studies on deductive databases [43, 59, 62, 73, 100, 139].

Many spatial relationships, such as *containment, inside* and *connected-to*, are defined recursively. Thus it is necessary to apply recursive rule compilation techniques in DOOS databases. Fortunately, most recursive relationships in spatial data applications are in relatively simple forms, such as transitive closures or linear recursions, whose implementations have been studied extensively in deductive databases. We will study deduction rules and their transformation techniques in spatial database systems.

In a DOOS database system, deduction rules can be compiled into expressions consisting of primitive predicates and operations before queries are posed to the system. The compilation of a deduction rule transforms the rule definition into a sequence of primitive relational operations, spatial operations and method calls (if some primitives are defined by methods) on spatial and/or nonspatial data. The rule compilation may be performed independently of queries, and the compiled program can be optimized and stored for later query processing.

4.1 Compilation of Spatial Rules

When a query is submitted to the system, its deductive predicates should first be resolved against the compiled rules. The results should then be further simplified and analyzed according to simplification rules.

As mentioned in Chapter 3, the instantiation constraints (modes) of procedure parameters distinguish spatial procedural predicates from data relations and therefore affect the processing of spatial queries.

During query processing, each input parameter, denoted by **in** should be instantiated before the method is called. A parameter in a method can be instantiated

by query constants, GDB/EDB accessing, or from information computed using other procedures or query predicates. If a procedure contains some uninstantiated parameters, the query processor should determine whether the data can be fetched from GDB/EDB or computed by other procedures. The invocation of such a procedure will have to be delayed until all of its input parameters have been instantiated.

Since attributes in a relation and procedure parameters are typed, the compilation of an IDB should perform the type consistency checks. This validation process often results in eliminating contradicting rules. A compiled program usually has a simpler set of types, thus reducing the cost of query processing. Type checking is performed by checking the consistency of the types of the corresponding variables in the procedures/predicates. This checking process tosses away the conflicting portions of parameter specifiers in the compiled formula and results in a compiled formula which is associated with a set of consistent parameter specifiers. This set can then be used in further compilation and query processing. We now examine the type checking operation in details.

Although an EDB attribute is defined to be of a specific type in its schema definition, its mode definition is omitted, since it always has a mode of **any** (either inquired or instantiated). On the other hand, a parameter (attribute variable) of an IDB predicate is defined to be of a specific type and mode using its IDB predicate definition. Since an IDB predicate may be defined by more than one rule, different rules may specify different types and modes for a specific parameter. One task of the compilation is to merge these parameter specifiers appropriately.

The power set of geo-primitives, $POINT$, $LINE$ and $POLYGON$, provides all of the types available for geo-objects. There exists a partial order among all possible geometric data types; this order can be defined by the type hierarchy. A spatial data hierarchy can be specified in a DOOS database, such as { *POINT, LINE, POLYGON*} $\subset$ *GEO*, which indicates that a POINT type is a subtype of type GEO, denoted as " POINT $\prec$ GEO". Notice that $type_1 \prec type_2$ if $type_1$ is subsumed by (or is a special case of) $type_2$, i.e. $type_1$ is compatible with and more restrictive than $type_2$. The

type of an IDB predicate parameter can be derived by taking the most restricted of the compatible types given to the parameters in the rules defining the predicate. If the variables with the same name in different predicates which define an IDB have conflicting types, an error will be reported and the compilation of the rule fails.

Example 4.1 Derivation of types and modes for the variables in a compiled rule.

Let the types and the modes of the attributes in the primitive relations *road* and *city*, and the variables in the geometric procedure *geo_intersection* be defined as

1. *road*(X: LINE **any**),
2. *city*(Y: POLYGON **any**), and
3. *geo_intersection*(X: GEO **in**, Y: GEO **in**, Z: GEO **out**),

Suppose that a rule *road_thru_city* is defined as follows.

road_thru_city(X, Y, Z) :- road(X), city(Y), geo_intersection(X, Y, Z).

Resolution is performed between the rule and the definition as follows: (1) *the most specific type of the type hierarchy that is consistent with both the rule and the definition* is resolved as the resulting type of the rule and (2) *the most general mode which is consistent with both* is resolved as the resulting mode.

In this example, the resulting type of the variable X in the head of the rule, *road_thru_city*, should be of type $LINE$ because the type of X in *road* is $LINE$ and in *geo_intersection* is GEO. Similarly, the type of Y is $POLYGON$ and that of Z is $LINE$. The resulting mode for X in *road_thru_city* is **any** because X is **any** in *road* and is **in** in *geo_intersection*. Similarly, the resulting mode for Y is **any** and that for Z is **out**. □

Example 4.2 Compilation of a deductive rule. We examine the type checking in the compilation process of a predicate,
available_river_side_space(X, Y), which returns **TRUE** if and only if X is a piece of available space adjacent to river Y. The definition uses another IDB predicate *adjacent(X, Y)* which returns **TRUE** if and only if a geo-object X is a neighbor of another geo-object Y, or if X and Y share a common boundary. Suppose that schema relations, the headers of the procedures, and IDB rules are defined as follows.

schema neighbor(X: POLYGON, Y: POLYGON), river(X: LINE), unused_space(X: POLYGON).

procedure boundary(X: POLYGON **in**, Y: LINE **out**), geo_is_intersected(X: GEO **in**, Y: GEO **in**).

$adjacent(X, Y)$:- $neighbor(X, Y)$.

$adjacent(X, Y)$:- $boundary(X, Z), boundary(Y, W), geo_is_intersected(W, Z)$.

$available_river_side_space(X, Y)$:- $unused_space(X)$, $river(Y)$, $adjacent(Y, X)$.

Type checking is performed during the compilation of the IDB predicates; it often results in eliminating some incompatible rules. For example, the compilation process of *available_river_side_space (X, Y)* detects that the first definition of *adjacent* should be excluded from the compiled rules because the type of a river(Y) is *LINE*, which does not match the type *POLYGON* of the variable Y in the EDB predicate *neighbor*.

In the compilation result, both the types and the modes of variables of *adjacent* are more restrictive than the originally declared, because of the restrictions on those of the variables in their defining rules. The type and mode of *adjacent* is either (X: POLYGON **any** , Y: POLYGON **any**) or (X: POLYGON **in**, Y: LINE **out**).

$$adjacent(X, Y) = neighbor(X, Y) \cup (boundary(X, Z),\ boundary(Y, W),\ geo_is_intersected(W, Z)).$$

Similarly, the mode and type of *available_river_side_space* should be (X: POLYGON **in** , Y: LINE **in**).

$$available_river_side_space(X, Y) = unused_space(X),\ river(Y),$$
$$boundary(X, Z),\ boundary(Y, W),\ geo_is_intersected(W, Z). \quad \square$$

During compilation, the IDB predicates are transformed into forms which consist of only EDB predicates and spatial methods and are easy to analyze further when a query is submitted to the system.

Example 4.3 The compilation of IDB rules for the spatial database of Example 3.4.

Recall that from Example 3.4 we have the following definitions and IDB rules:

schema pollution_map(P: POLYGON), rel_inside(X: GEO, Y:POLYGON),
parcel(X: POLYGON).

$polluted(X)$:- $pollution_map(Y),\ intersect(X, Y)$.

$polluted_parcel_in(X, Y)$:- $parcel(X),\ polluted(X),\ within(X, Y)$.

$inside(X, Y)$:- $is_inside_of(X, Y)$.

$inside(X, Y)$:- $rel_inside(X, Y)$.

$within(X, Y)$:- $inside(X, Y)$.

$within(X, Y)$:- $inside(X, Z),\ within(Z, Y)$.

The rule *polluted(X)* indicates that X is polluted if X intersects with any pollution_map Y. The rule *polluted_parcel_in_region (X, Y)* indicates that X is a polluted parcel within region Y. The rule *inside(X, Y)* indicates that X is inside of Y based on either relation $rel_inside(X, Y)$ or the geometric predicate *is_inside_of (X, Y)*. The rule *within (X, Y)* indicates that X is within area Y if X is inside Y or if X is inside Z which is, in turn, within Y.

The compilation results are in the following compiled IDB predicates:

$$polluted_parcel_in(X,Y) = parcel(X),\ pollution_map(Z),$$
$$geo_is_intersected(Z,X),\ within(X,Y).$$
$$within(X,Y) = inside^{+}(X,Y).$$
$$inside(X,Y) = rel_inside(X,Y) \cup is_inside_of(X,Y).$$ □

Notice that in the compiled form, the predicate $polluted_parcel_in(X,Y)$ is still represented using an intermediate predicate $within(X,Y)$. This should be viewed as a concise notation. It does not imply that the computation of the predicate $polluted_parcel_in(X,Y)$ cannot start before the completion of the computation of the intermediate predicate $within(X,Y)$. Similar arguments hold for the predicate $within(X,Y)$ which is represented by an intermediate predicate $is_inside_of(X,Y)$. Furthermore, the predicate $within(X,Y)$ is represented using the transitive closure notation, $inside^{+}(X,Y)$, since it is defined by a set of recursive rules.

Most spatial database application programs can be written using deduction rules, procedural definitions or their combinations. By compiling IDB rules into simpler forms and performing the type checking at compilation time, most application programs can be processed efficiently. Although recursive query processing poses new challenges to efficient evaluation, most recursive rules can be compiled into simple forms, such as transitive closures or asynchronous chain recursions which can be evaluated efficiently [62]. The compilation and efficient processing of recursive queries have been studied extensively in deductive database research, such as [59, 139]; it will not be addressed here.

4.2 Spatial Query Simplification

Simplification of algebraic expressions has been studied extensively [139] and most of the existing results can be applied to spatial query optimization. Furthermore, spatial

properties and spatial equivalence rules can be used to perform algebraic simplifications which transform a compiled rule (obtained by rule compilation) or a compiled query expression (obtained by resolving query predicates with the compiled rules) into simpler expressions which can be evaluated more efficiently. In addition to relational algebra used for simplification of relational expressions, spatial query simplification uses geometric algebra and other spatial properties [54].

4.2.1 Spatial properties, equivalences and translation rules

Spatial properties can be used to simplify spatial expressions or to compute a virtual spatial relation using a set of existing or precomputed spatial relations.

Definition 4.1 A predicate $p(A, B)$ is **symmetric** if $p(A, B) = p(B, A)$. A predicate $p(A, B)$ is **transitive** if $p(A, B)$ and $p(B, C)$ imply $p(A, C)$. A predicate $p_1(A, B)$ is a **converse** of another predicate $p_2(A, B)$ if $p_1(A, B) = p_2(B, A)$. A predicate $p_1(A, B)$ is a **complement** of another predicate $p_2(A, B)$ if $p_1(A, B) = \neg p_2(A, B)$.

Example 4.4 The symmetric, transitive, complement and converse properties of some common relations can be illustrated using the following logic rules.

1. "symmetry": $is_adjacent_to(A, B)$:- $is_adjacent_to(B, A)$.
2. "converse": $is_inside_of(A, B)$:- $contains(B, A)$.
3. "transitivity": $is_inside_of(A, B)$:- $is_inside_of(A, T), is_inside_of(T, B)$.
4. "complement": $is_disjoint_from(A, B)$:- $\neg overlaps(A, B)$. □

Some spatial properties can be expressed by algebraic equivalence expressions. The computational cost is reduced if the transformed expression can be evaluated using fewer or less costly spatial operations than the original one. For example, using a set equivalence rule shown below,

$$(A \cap B) \cup (A \cap C) = A \cap (B \cup C),$$

a spatial computation that involves three spatial function calls,

$$geo_union(geo_intersection(A, B),\ geo_intersection(A, C)),$$

can be transformed into one that requires only two spatial function calls,

$$geo_intersection(A,\ geo_union(B, C)).$$

Similarly to the specification of query optimization rules in extensible spatial database systems [13], integrity constraints and conditional simplification rules can be specified explicitly by spatial database experts to facilitate the simplification process. For example, since *the total area of two non-overlapping regions* can be computed by a simple summation of the areas of the two regions, the rule can be specified as a spatial transformation rule as shown below.

$$\textbf{if} A \cap B = \phi \textbf{ then } area(A \cup B) = area(A) + area(B).$$

Another important simplification technique in spatial query processing is to derive a complex spatial relationship from known spatial information without geometric computations.

The spatial functions which generate new spatial objects, such as *geo_union*, are often costly to compute. However, using precomputed component information and precomputed spatial relationships among these components, dynamic computation of spatial functions sometimes can be avoided, thus the cost of such computations can be reduced. Precomputation of certain spatial relationships can be performed on a relatively stable domain if these relationships are primitive and frequently used. It is unrealistic to precompute and store all geometric predicates, or to register spatial relationships among all of possible generated objects. For example, one cannot store all possible objects constructed by *geo_union*. Nevertheless, it is desirable to derive spatial relationships for generated objects from the precomputed spatial relationships of their primitive components. The following are some possible ways to derive such relationships.

1. [Equivalence condition] A compound relation can be determined from its component relations. For example, from A *is disjoint from* B and A *is disjoint from* C, it is derived that A *is disjoint from* $(B \cup C)$.

2. [Necessity condition] If the component conditions do not hold, the compound condition will not hold. For example, the necessity condition for A *overlaps with* $(B \cap C)$, is that A *overlaps with* B and A *overlaps with* C.

3. [Sufficiency condition] If the component relations hold, the compound condition will hold. For example, a sufficient condition for A *is_disjoint_from* $(B \cap C)$ is that A *is_disjoint_from* B or A *is_disjoint_from* C.

Equivalence conditions can be used in spatial query compilation [7, 13, 21, 72, 92]. Compound relations can be replaced by their equivalent component relations. Necessary conditions can be used to derive a predicate to be **FALSE** from that its necessary condition is false. Sufficient conditions can be used to derive a predicate to be **TRUE** from that its sufficient condition holds. The geometric operation is performed only if the compound condition cannot be determined by the component conditions. The derivation of compound relationship is discussed in the next subsection.

4.2.2 Derivation of compound relationships

Let us examine a set of frequently used topological relations, i.e. *contains, is_inside_of, is_adjacent_to, overlaps, is_disjoint_from* in combination with three geometric set operators *geo_union, geo_intersection* and *geo_difference* denoted by $\cup$, $\cap$ and $-$ respectively. Notice that *is_inside_of* and *contains* are converse predicate relations, *is_disjoint_from, overlaps* and *is_adjacent_to* are symmetric predicates, *contains* and *is_inside_of* are both transitive relations and *is_disjoint_from* and *overlaps* are complementary relations.

The following symbols are used in the following discussion. The symbol $\Longleftrightarrow$ connects two equivalence expressions. As some compound relations cannot be exactly

determined from their components, let $\Longleftarrow$ denote *sufficiency*, and $\Longrightarrow$ *necessity*. We adopt the set notation proposed in [35], where $\partial\mathcal{A}$, $\mathcal{A}^\circ$ and $\mathcal{A}^{-1}$ are mutually exclusive and stand for the points of the *boundary*, the *interior* and the *exterior* of A respectively. The letter $\mathcal{A}$ stands for $\mathcal{A}^\circ \cup \partial\mathcal{A}$. Symbol θ is used to refer to an arbitrary binary relationship. Assume that point sets $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ and $(\mathcal{B}\ \theta\ \mathcal{C})$ are non-empty.

Now, we prove some equivalence relationships, necessity conditions and sufficiency conditions. Examples will be given when the equivalent condition does not hold.

4.2.2.1 Disjointness (is_disjoint_from)

Definition 4.2 *A is_disjoint_from B* $::= \mathcal{A} \cap \mathcal{B} = \phi$

Formula 1 A is_disjoint_from (B ∪ C) $\Longleftrightarrow$ (A is_disjoint_from B) ∧ (A is_disjoint_from C)

Proof:

1. $\Longleftarrow$ If A is_disjoint_from B and A is_disjoint_from C, by definition,

   $(\mathcal{A} \cap \mathcal{B}) = \phi$, and $(\mathcal{A} \cap \mathcal{C}) = \phi$.

   Hence $(\mathcal{A} \cap \mathcal{B}) \cup (\mathcal{A} \cap \mathcal{C}) = \phi \cup \phi = \phi$,

   and so $\mathcal{A} \cap (\mathcal{B} \cup \mathcal{C}) = \phi$.

   Therefore A is_disjoint_from (B ∪ C).

2. $\Longrightarrow$ The proof in the other direction is similar.

Thus Formula 1 holds. □

Formula 2 A is_disjoint_from (B ∩ C) $\Longleftarrow$ (A is_disjoint_from B) ∨ (A is_disjoint_from C)

Proof: We consider the following two cases.

case 1: if A is_disjoint_from B, by definition, $\mathcal{A} \cap \mathcal{B} = \phi$

By associativity, we have $\mathcal{A} \cap (\mathcal{B} \cap \mathcal{C}) = (\mathcal{A} \cap \mathcal{B}) \cap \mathcal{C} = \phi \cap \mathcal{C} = \phi$.

Therefore A is_disjoint_from (B ∩ C).

case 2: By symmetry, if *A is_disjoint_from C*, then *A is_disjoint_from (B ∩ C)*.

Hence Formula 2 holds. □

Figure 4.1 shows an example where A is disjoint from (B ∩ C) while A overlaps with both B and C. Therefore the converse of Formula 2, $A\ is_disjoint_from\ (B \cap C) \Longrightarrow (A\ is_disjoint_from\ B) \vee (A\ is_disjoint_from\ C)$, does not hold.

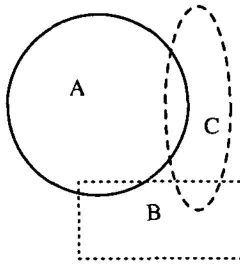


Figure 4.1: A counter-example to the converse of Formula 2

Formula 3 A is_disjoint_from (B − C) ⟸ (A is_disjoint_from B) ∨ (C contains A)

Proof: We consider the following two cases.

case 1: if A is_disjoint_from B, by definition, $\mathcal{A} \cap \mathcal{B} = \phi$.

Because $(\mathcal{B} - \mathcal{C}) \subseteq \mathcal{B}$ and $\mathcal{A} \cap (\mathcal{B} - \mathcal{C}) = \phi$, A is_disjoint_from (B − C).

case 2: if C contains A, $\mathcal{C} \supseteq \mathcal{A}$, $(\mathcal{B} - \mathcal{C}) \subseteq (\mathcal{B} - \mathcal{A})$

Because A is_disjoint_from (B − A), A is_disjoint_from (B − C).

Hence Formula 3 is justified. □

Figure 4.2 shows a case where A overlaps (B − C) while A overlaps with B and C does not contain A. Therefore the converse of Formula 3, $A\ is_disjoint_from\ (B - C) \Longrightarrow (A\ is_disjoint_from\ B) \vee (C\ contains\ A)$, does not hold.

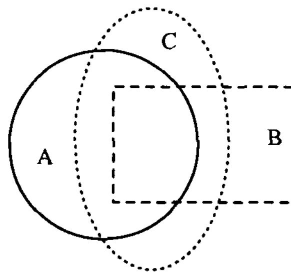


Figure 4.2: A counter-example to the converse of Formula 3

4.2.2.2 Overlapping (overlaps)

Definition 4.3 $A\ overlaps\ B ::= \mathcal{A} \cap \mathcal{B} \neq \phi$

Formula 4 A overlaps (B ∪ C) $\Longleftrightarrow$ (A overlaps B) ∨ (A overlaps C)

Proof: This is the contrapositive of Formula 1. □

Formula 5 A overlaps (B ∩ C) $\Longrightarrow$ (A overlaps B) ∧ (A overlaps C)

Proof: This is the contrapositive of Formula 2. □

Figure 4.3 shows an example where A overlaps with both B and C whereas A does not overlaps with (B ∩ C). Therefore the converse of Formula 5, $A\ overlaps\ (B \cap C) \Longleftarrow (A\ overlaps\ B) \wedge (A\ overlaps\ C)$, does not hold.

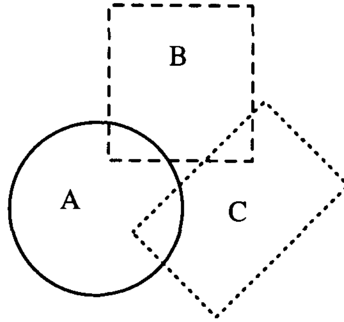


Figure 4.3: A counter-example to the converse of Formula 5

Formula 6 A overlaps (B − C) $\Longrightarrow$ (A overlaps B) $\wedge$ $\neg$ (C contains A)

Proof: This is the contrapositive of Formula 3. ◫

Figure 4.4 shows an example where A does not overlap with (B − C) while A overlaps with B and C does not contain A. Therefore the converse of Formula 6, *A overlaps (B − C)* $\Longleftarrow$ *(A overlaps B)* $\wedge$ $\neg$ *(C contains A)*, does not hold.

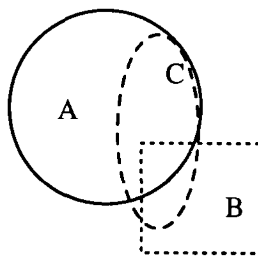


Figure 4.4: A counter-example to the converse of Formula 6

4.2.2.3 Inside (is_inside_of)

Definition 4.4 *A is_inside_of B* $::= \mathcal{A} \subseteq \mathcal{B}$

Formula 7 A is_inside_of (B $\cup$ C) $\Longleftarrow$ (A is_inside_of B) $\vee$ (A is_inside_of C)

Proof: We consider the following two cases.

case 1: If A is_inside_of B, by definition, $\mathcal{A} \subseteq \mathcal{B} \subseteq (\mathcal{B} \cup \mathcal{C})$.

Therefore A is_inside_of (B ∪ C).

case 2: if A is_inside_of C, then by definition $\mathcal{A} \subseteq \mathcal{C} \subseteq (\mathcal{B} \cup \mathcal{C})$.

Hence Formula 7 is true. □

Figure 4.5 shows an example where A is inside (B ∪ C) whereas A is not inside of either B or C. Therefore the converse of Formula 7, $A\ is_inside_of\ (B \cup C) \Longrightarrow (A\ is_inside_of\ B) \vee (A\ is_inside_of\ C)$, does not hold.

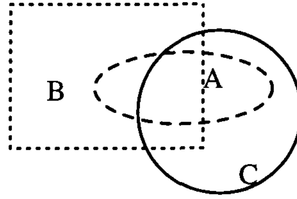


Figure 4.5: A counter-example to the converse of Formula 7

Formula 8 A is_inside_of (B ∩ C) $\Longleftrightarrow$ (A is_inside_of B) ∧ (A is_inside_of C)

Proof:

1. $\Longleftarrow$ If A is_inside_of B and A is_inside_of C, then $\mathcal{A} \subseteq \mathcal{B}$ and $\mathcal{A} \subseteq \mathcal{C}$.

   If $a \in \mathcal{A}$, then $a \in \mathcal{B}$ and $a \in \mathcal{C}$.

   Since this holds for all $a \in A$, it follows that $a \in (\mathcal{B} \cap \mathcal{C})$ therefore $\mathcal{A} \subseteq (\mathcal{B} \cap \mathcal{C})$.

   A is_inside_of (B ∩ C) follows.

2. $\Longrightarrow$ Analogously, the formula can be proven in other direction.

Hence Formula 8 holds. □

Formula 9 A is_inside_of (B − C) $\Longleftrightarrow$ (A is_inside_of B) $\wedge$ (A is_disjoint_from C)

Proof:

1. $\Longrightarrow$ If *A is_inside_of (B − C)*, then by definition

   $\mathcal{A} \subseteq (\mathcal{B} \cap \mathcal{C}^{-1}) \subseteq \mathcal{B}$, and therefore A is_inside_of B.

   Similarly, $\mathcal{A} \subseteq (\mathcal{B} \cap \mathcal{C}^{-1}) \subseteq \mathcal{C}^{-1}$ and thus, A is_disjoint_from C.

2: $\Longleftarrow$ If *A is_inside_of B* and *A is_disjoint_from C*, by definition,

   $\mathcal{A} \subseteq \mathcal{B}$ and $\mathcal{A} \cap \mathcal{C} = \phi$,

   i.e. $(\mathcal{A} \subseteq \mathcal{C}^{-1})$, and so $\mathcal{A} \subseteq (\mathcal{B} \cap \mathcal{C}^{-1})$ follows.

   Therefore *A is_inside_of (B − C)*, and Formula 9 holds. □

4.2.2.4 Containment (contains)

Definition 4.5 *A contains B* ::= $\mathcal{A} \supseteq \mathcal{B}$

Formula 10 A contains (B ∪ C) $\Longleftrightarrow$ (A contains B) $\wedge$ (A contains C)

Proof:

1. $\Longrightarrow$ If A contains (B ∪ C), then by definition $\mathcal{A} \supseteq (\mathcal{B} \cup \mathcal{C})$.

   Hence $\mathcal{A} \supseteq \mathcal{B}$ and $\mathcal{A} \supseteq \mathcal{C}$, and therefore A contains B and A contains C.

2. $\Longleftarrow$ The proof for the other direction is similar.

Hence Formula 10 holds. □

Formula 11 A contains (B ∩ C) $\Longrightarrow$ (A overlaps B) ∧ (A overlaps C)

Proof: If A contains (B ∩ C), by definition, $\mathcal{A} \supseteq (\mathcal{B} \cap \mathcal{C})$.

If $(\mathcal{B} \cap \mathcal{C}) \neq \phi$ (assumption), $\mathcal{A} \cap (\mathcal{B} \cap \mathcal{C}) \neq \phi$.

Therefore $(\mathcal{A} \cap \mathcal{B}) \neq \phi$ and $(\mathcal{A} \cap \mathcal{C}) \neq \phi$, and Formula 11 holds. □

Figure 4.6 shows that A overlaps with both B and C but A does not contain (B ∩ C). Therefore the converse of Formula 11, *A contains (B ∩ C)* $\Longleftarrow$ *(A overlaps B) ∧ (A overlaps C)*, does not hold.

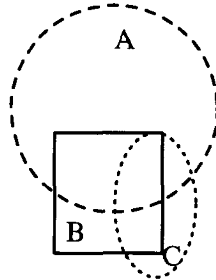


Figure 4.6: A counter-example to the converse of Formula 11

Formula 12 A contains (B ∩ C) $\Longleftarrow$ (A contains B) ∨ (A contains C)

Proof: If A contains B, then by definition $\mathcal{A} \supseteq \mathcal{B}$.

Because $\mathcal{B} \supseteq (\mathcal{B} \cap \mathcal{C})$, it follows that $\mathcal{A} \supseteq (\mathcal{B} \cap \mathcal{C})$.

Similarly, if $\mathcal{A} \supseteq \mathcal{C}$, then $\mathcal{A} \supseteq (\mathcal{B} \cap \mathcal{C})$, and Formula 12 holds. □

Figure 4.7 shows that A contains (B ∩ C) but does not contain either B or C. Therefore the converse of Formula 12, *A contains (B ∩ C)* $\Longrightarrow$ *(A contains B)* ∨ *(A contains C)*, does not hold.

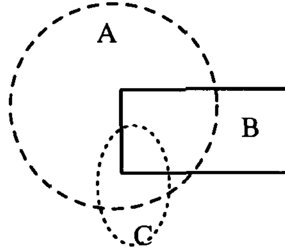


Figure 4.7: A counter-example to the converse of Formula 12

Formula 13 A contains (B − C) $\Longleftarrow$ A contains B

Proof: If A contains B, then by definition, $\mathcal{A} \supseteq \mathcal{B}$.

Because $\mathcal{B} \supseteq (\mathcal{B} - \mathcal{C})$, it follows that $\mathcal{A} \supseteq (\mathcal{B} - \mathcal{C})$.

Therefore A contains (B − C) and Formula 13 holds. □

Figure 4.8 shows an example where A contains (B − C) while A does not contain B. Therefore the converse of Formula 13, *A contains (B − C)* $\Longrightarrow$ *A contains B*, does not hold.

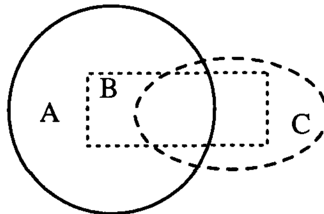


Figure 4.8: A counter-example to the converse of Formula 13

4.2.2.5 Adjacency (is_adjacent_to)

Definition 4.6 *A is_adjacent_to B* ::= $(\partial\mathcal{A} \cap \partial\mathcal{B} \neq \phi) \wedge (\mathcal{A}^\circ \cap \mathcal{B}^\circ = \phi)$.

Formula 14 A is_adjacent_to (B ∪ C) $\Longleftrightarrow$
((A is_adjacent_to B) ∧ (A is_adjacent_to C)) ∨
((A is_adjacent_to B) ∧ (A is_disjoint_from C))∨
((A is_adjacent_to C) ∧ (A is_disjoint_from B))

Proof: 1. $\Longrightarrow$ If A is_adjacent_to (B ∪ C), then by definition,

$\partial\mathcal{A} \cap \partial(\mathcal{B} \cup \mathcal{C}) \neq \phi$ and $\mathcal{A}^\circ \cap (\mathcal{B} \cup \mathcal{C})^\circ = \phi$.

Because B ∪ C contains some boundary of B or C, $\partial(B \cup C) \subseteq \partial B \cup \partial C$

Similarly, $(B \cup C)^\circ \supseteq B^\circ \cup C^\circ$, we have,

$$\partial A \cap (\partial B \cup \partial C) \supseteq \partial\mathcal{A} \cap \partial(\mathcal{B} \cup \mathcal{C}) \neq \phi,$$

$$(\partial\mathcal{A} \cap \partial\mathcal{B}) \cup (\partial\mathcal{A} \cap \partial\mathcal{C}) \neq \phi. \tag{4.1}$$

$$A^\circ \cap (B^\circ \cup C^\circ) \subseteq \mathcal{A}^\circ \cap (\mathcal{B} \cup \mathcal{C})^\circ = \phi,$$

$$(\mathcal{A}^\circ \cap \mathcal{B}^\circ) \cup (\mathcal{A}^\circ \cap \mathcal{C}^\circ) = \phi. \tag{4.2}$$

There are three possible cases to be considered for equation (4.1) to hold:

case 1: $(\partial\mathcal{A} \cap \partial\mathcal{B} \neq \phi)$ and $(\partial\mathcal{A} \cap \partial\mathcal{C} \neq \phi)$.

combining with equation (4.2), we have

(A is_adjacent_to B) ∧ (A is_adjacent_to C).

case 2: $(\partial\mathcal{A} \cap \partial\mathcal{B} \neq \phi)$ and $(\partial\mathcal{A} \cap \partial\mathcal{C} = \phi)$.

In combination with equation (4.2), we have

(A is_adjacent_to B) ∧ (A is_disjoint_from C).

case 3: $(\partial\mathcal{A} \cap \partial\mathcal{B} = \phi)$ and $(\partial\mathcal{A} \cap \partial\mathcal{C} \neq \phi)$.

In combination with equation (4.2), we have

(A is_adjacent_to C) ∧ (A is_disjoint_from B).

2: $\Longleftarrow$ The proof in the other direction is similar.

Hence Formula 14 holds. □

Formula 15 A is_adjacent_to (B − C) $\Longleftarrow$ (A is_adjacent_to B) $\wedge$
(A is_disjoint_from C)

Proof: If A *is_adjacent_to* B and A *is_disjoint_from* C, then by definition,

$\partial\mathcal{A} \cap \partial\mathcal{B} \neq \phi$ and $\mathcal{A}^\circ \cap \mathcal{B}^\circ = \phi$.

Furthermore, $\mathcal{A} \cap \mathcal{C} = \phi$, i.e. $\partial\mathcal{A} \cap \partial\mathcal{C} = \phi$ and $\mathcal{A}^\circ \cap \mathcal{C}^\circ = \phi$.

Hence $\partial A \cap \partial C = \phi \Longrightarrow \partial A \cap \partial(B - C) = \partial A \cap \partial B$

$\mathcal{A}^\circ \cap (\mathcal{B} \ - \ \mathcal{C})^\circ \supseteq (\mathcal{A}^\circ \cap \mathcal{B}^\circ) \ - \ (\mathcal{A}^\circ \cap \mathcal{C}^\circ) = \phi$.

Therefore A is adjacent to (B − C).

Hence Formula 15 holds. □

Figure 4.9 shows an example in which A is not disjoint from C but is adjacent to (B − C). Therefore the converse of Formula 15, *A is_adjacent_to (B − C)* $\Longrightarrow$ *(A is_adjacent_to B)* $\wedge$ *(A is_disjoint_from C)*, does not hold.

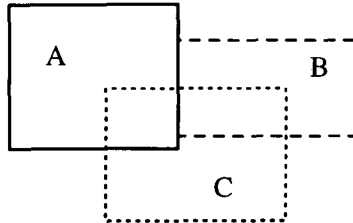


Figure 4.9: A counter-example to the converse of Formula 14

These equivalences, necessary conditions and sufficient conditions can be represented as optimization rules. The query processor can test the condition and replace a complex forms by their simplified equivalence.

In summary, we have derived and proved some interesting formulas for compound spatial relations from their component relations. Table 4.1 is a summary of the result where $\subseteq$, $\supseteq$, $\pitchfork$, $\not\pitchfork$ and $||$ are abbreviations for *is_inside_of*, *contains*, *overlaps*, *is_disjoint_from* and *is_adjacent_to* respectively.

θ	object relationship					
		$A\ \theta\ (B \cup C)$		$A\ \theta\ (B \cap C)$		$A\ \theta\ (B - C)$
$\subseteq$	$\Longleftarrow$	$(A \subseteq B) \vee (A \subseteq C)$	$\Longleftrightarrow$	$(A \subseteq B) \wedge (A \subseteq C)$	$\Longleftrightarrow$	$(A \subseteq B) \wedge (A \not\Cap C)$
$\supseteq$	$\Longleftrightarrow$	$(A \supseteq B) \wedge (A \supseteq C)$	$\Longleftarrow$ $\Longrightarrow$	$(A \supseteq B) \wedge (A \supseteq C)$ $(A \Cap B) \wedge (A \Cap C)$	$\Longleftarrow$	$(A \supseteq B)$
$\Cap$	$\Longleftrightarrow$	$(A \Cap B) \vee (A \Cap C)$	$\Longrightarrow$	$(A \Cap B) \wedge (A \Cap C)$	$\Longrightarrow$	$(A \Cap B) \wedge \neg(C \supseteq A)$
$\not\Cap$	$\Longleftrightarrow$	$(A \not\Cap B) \wedge (A \not\Cap C)$	$\Longleftarrow$	$(A \not\Cap B) \vee (A \not\Cap C)$	$\Longleftarrow$	$(A \not\Cap B) \vee (C \supseteq A)$
$\parallel$	$\Longleftrightarrow$	$(A \parallel C) \wedge (A \parallel B)$ $\vee (A \parallel B) \wedge (A \not\Cap C)$ $\vee (A \parallel C) \wedge (A \not\Cap B)$			$\Longleftarrow$	$(A \parallel B) \wedge (A \not\Cap C)$

Table 4.1: Complex spatial relation derivation.

These formula are useful for spatial query optimization. The application of these conditions in query optimization can be rule-based. All equivalences can be applied directly in query compilation. Equivalent component conditions can be substituted for compound relations. When left side of an equivalence formula is matched, it is replaced by the right side of the equation. Necessary conditions and sufficient conditions can be utilized at execution time to reduce geometric computations. Using these conditions, we can significantly reduce the cost of geometric query evaluation.

4.3 Chapter Summary

The compilation and algebraic simplification of spatial rules and queries has been studied in this chapter. Important differences between spatial rule and relation rule compilations are (i) parameter specifier checking and derivation, (ii) utilization of geometric properties for simplification, and (iii) derivation of complex spatial relationships from simple ones without geometric computations. The compilation process provides a simplified spatial query expression consisting only of primitive spatial predicates and relations.

Chapter 5

Spatial Query Execution in a DOOSDB

In this chapter, we present a dynamic connection graph for spatial query access plan generation. A heuristic branch-and-bound algorithm will be used for access plan search. Set-oriented query evaluation techniques for efficient spatial query execution will also be presented.

5.1 Dynamic Connection Graph Transformation for Spatial Access Plan Generation

5.1.1 Introduction

It is necessary to perform a systematic study of access plan generation for deductive spatial databases. In this chapter, a dynamic connection graph transformation approach is proposed for optimization of compiled spatial queries. Here is an outline of the approach. For each compiled query, a *connection graph* is constructed which represents the possible data flow among EDB and spatial predicates in the compiled

query. A spatial predicate is *unavailable* for optimization if it does not have enough *available* inputs. Similarly, its output is *unavailable* for further operations if it has not been computed even if all inputs have been obtained. The connection graph is modified dynamically as the situation changes and more spatial predicates can participate in the optimization process. The connection graph transformation provides a dynamic picture of spatial query optimization. Suboptimal query access plans can be selected from among the set of candidate plans generated based on the analysis of connection graphs.

5.1.2 Dynamic connection graph and access plan enumeration

A connection graph represents a set of candidate query execution sequences. For each given query, the connection graph consists of a set of nodes and a set of edges. Each node corresponds to a predicate in the query. When two predicates share a parameter, an edge is added between these two nodes. Since a spatial predicate is usually implemented by a spatial routine with fixed parameter modes (such as *input* or *output*), a spatial predicate is ready for computation only if all of its *input* mode parameters have been instantiated. This constraint is reflected in the dynamic connection graph.

An essential difference between dynamic connection graphs from connection graphs used in relational query optimization is that the graph nodes of dynamic graphs may have different statuses. There are three possible statuses for a node in the connection graph: *unavailable, input-available* and *output-ready*. A node is **unavailable** (denoted by a *triangle*) if the predicate contains some uninstantiated input parameters; it is **input-available** (denoted by a *square*) if all of the *input* parameters are instantiated and the predicate is ready for evaluation; and it is **output-ready** (denoted by a *circle*) if the spatial operation has been performed and the output is ready to be used for other operations. Obviously, the status of a node may change as the analysis proceeds. The status of an EDB relation node is always **output-ready**.

Definition 5.1 A *connection graph* G = (V, E) consists of a set of vertices V and a set of edges E.

1. V represents a set of nodes, one for each EDB relation or GDB predicate in the compiled query expression. Each node is in one of the three possible statuses: *unavailable* ($\triangle$), *input-available* ($\square$), and *output-ready* ($\bigcirc$), and the status may change dynamically. Each occurrence of a predicate in the compiled query should be treated as a distinct node (with a distinct label).

2. E represents a set of edges, each connecting two nodes where corresponding predicates share parameters (attributes). That is,

$$E = \{\langle v_i, v_j, A\rangle \mid v_i, v_j \in V,\ A \text{ is a set of attribute names in both } v_i \text{ and } v_j\ \}.$$

$\square$

During the query analysis process, a set of *candidate graphs* (dynamic connection subgraphs) are maintained. Each candidate graph represents a set of operations that are valid at the current stage (thus none of the *unavailable* nodes is included). After the execution of a spatial operator, an *unavailable* node may become *input-available* and be included into a new candidate graph for the analysis of the next step. Thus analysis continues until all the components in the graph have been analyzed, and a suboptimal plan generated. By applying heuristics and statistics obtained during the execution of geometric procedures, the enumeration process prunes unpromising access plans and selects promising ones. The following assumptions are made throughout our discussion.

Assumption 1 The input to the optimizer is a set of compiled query expressions.

That is, function symbols are resolved into functional predicates during the rectification process [63]. For example, "$geo_union(X, Y) > 100$" is converted to "$geo_union(X, Y, Z),\ Z > 100$". A compiled query is in a conjunctive normal form consisting of EDB predicates, GDB predicates and their transitive closures.

Assumption 2 The only cost considered in the analysis is the computational cost.

The cost of query processing should consist of computational and I/O cost. Since I/O cost is in general proportional to the computational cost, this assumption is adopted to simplify our discussion. When necessary, I/O cost can be easily taken into consideration since the I/O cost model has been studied extensively in relational systems.

Assumption 3 Relational and spatial data statistics are available, and the cost estimate of spatial operators reflects the complexity of the corresponding spatial algorithms.

Assumption 4 Transitive closures will be handled by deductive techniques. The approximated cost estimation is available.

Example 5.1 Let us consider the query: *find a cedar forest whose area exceeds* 10 km^2:

$$? - forest(X),\ area(X, A),\ A > 10,\ type(X, cedar),$$

where *forest* and *type* are data relations and *area* is a spatial operator. Of many possible query execution plans, one could be: (1) performing selection on *type*, (2) joining *forest* and *type*, and then (3) computing *area*. It is a reasonable plan since selection provides a strong constraint, and the retrieval of the EDB relation *forest* is less expensive than the computation of the spatial function *area*. □

Similar observations lead to the following heuristics in query plan selection:

1. Only valid operations (excluding uninstantiated predicates) should be considered at any time,
2. Perform selection first (i.e. push in constants as soon as possible),
3. Computationally less demanding routines should be performed earlier to extract additional constraints that may reduce the cost of later computations,

4. The computation of geometric operations should be delayed since they are in general more costly than relational ones,

5. The computation order among several instantiated geometric routines, should be determined by their cost estimates.

The core of the query plan generation algorithm is an enumeration process which transforms a connection graph into a tree of candidate access plans. Each candidate plan consists of a valid operation sequence and its corresponding data flow. An edge links each operator with its input data. The graph retrieval process is done using a branch-and-bound algorithm with heuristic search. The estimated cost is based on the cost of each estimated operation and the size of the input set and is accumulated along the retrieval path. Edges in the candidate graph are ordered using heuristics. Here is an outline of the algorithm. Let us first show an example of access plan generation.

Example 5.2 Let us examine the query, *"find one's land pieces (and their areas) that are suitable for planting both crops and tea trees"*:

$? - crop_land(X, Owner), tea_land(Y, Owner), geo_intersection(X, Y, Z), area(Z, A).$

Notice that $crop_land(X, Owner)$ and $tea_land(Y, Owner)$ are EDB relations, that $geo_intersection(X, Y, Z)$ is a spatial predicate which takes two input objects X and Y and returns their intersection in Z, and that $area(Z, A)$ is a spatial predicate which computes the area A of the region denoted by the input parameter Z. Obviously, the value of Z must depend on the output of $geo_intersection(X, Y, Z)$ in the query.

Let *c_land*, *t_land* and *geo_int* refer to *crop_land*, *tea_land*, and *geo_intersection* respectively. The connection graph is shown in Figure 5.1 (a). The candidate subgraph which contains available operations is in Figure 5.1 (a) with currently unavailable operations in the dashed box. Node *area* is excluded from the candidate graph because it is currently *unavailable*.

The first step of the enumeration algorithm has two choices: (i) *c_land* ⋈ *t_land*, or (ii) *eval(geo_int)*. There are a total of 10 possible graph change sequences (candidate plans). Figure 5.1 presents two of the plans which correspond to the sequences {(a), (b), (c), (g)} and { (a), (d), (e), (f), (g)} respectively.

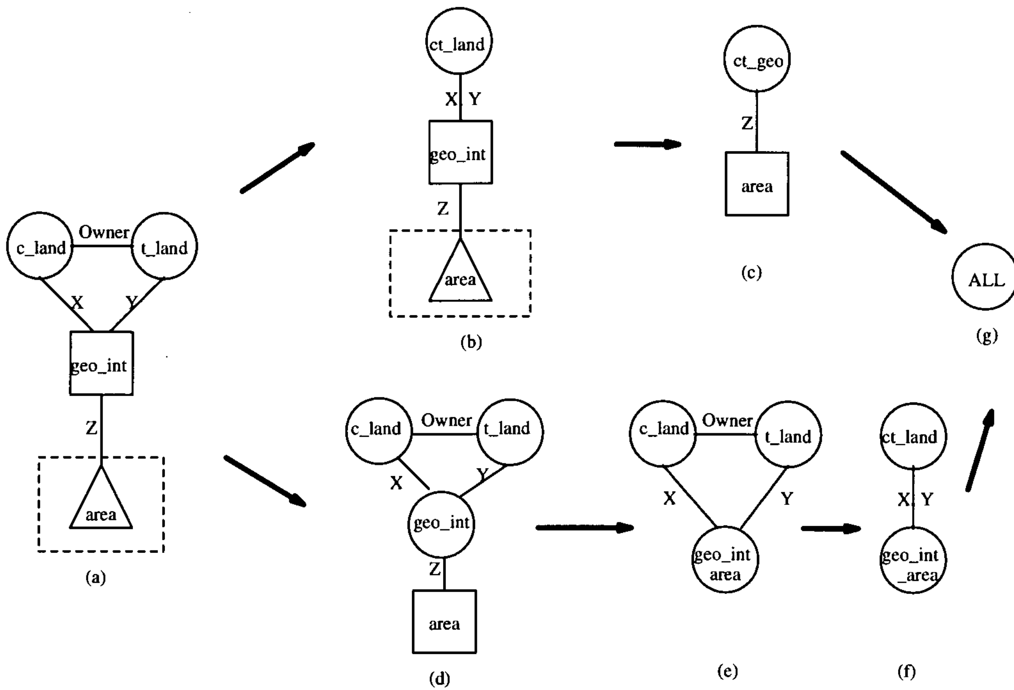


Figure 5.1: Candidate graphs in the enumeration of two access plans.

The first plan P_1 starts with *c_land* ⋈ *t_land*. Thus *c_land* and *t_land* are merged into *ct_land*, and edges $\langle c_land, geo_int, X\rangle$ and $\langle t_land, geo_int, Y\rangle$ are consolidated into one edge $\langle ct_land, geo_int, \{X, Y\}\rangle$ as shown in Figure 5.1 (*b*). The only option next is to evaluate *geo_int* which takes inputs X and Y from *ct_land* and merges the two nodes into one *ct_geo*. Since *geo_int*'s output is available to *area* as input, *area* is added to the candidate graph for plan generation, as shown in Figure 5.1 (*c*). At this stage, only one choice is available, that is, to evaluate *area*. The final result is illustrated in Figure 5.1 (*g*).

The second plan P_2 starts by evaluating *geo_int*, which takes input from *c_land* and *t_land* and changes the status of *geo_int* into *available*, and that of *area* into *input-available* as shown in Figure 5.1 (*d*). There are then four possible choices: (i) *c_land* $\bowtie$ *t_land*, (ii) *t_land* $\bowtie$ *geo_int*, (iii) *c_land* $\bowtie$ *geo_int*, and (iv) eval(*area*). Assume that eval(*area*) is performed first with input *geo_int* as shown in Figure 5.1 (*e*). Nodes *geo_int* and *area* are merged into *geo_int_area*, and the graph becomes a typical three-way join whose analysis with respect to relational query optimization has been done [139]. The resulting access plan tree is shown in Figure 5.2 in which a node with double circles represents an operation. ◻

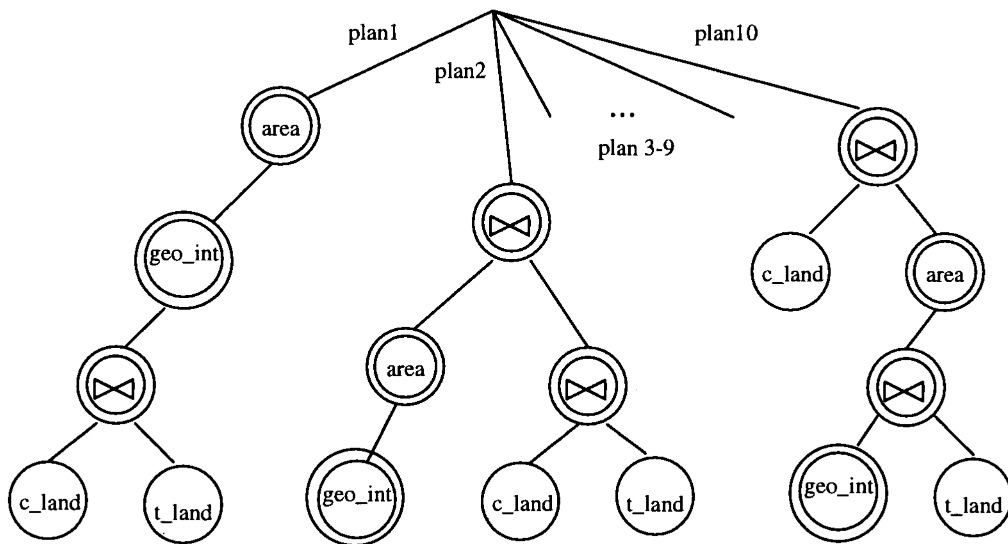


Figure 5.2: An access plan tree.

Algorithm 5.1 Selection of a suboptimal query access plan for a compiled deductive spatial query using an enumeration approach.

Input. A compiled deductive spatial query and data statistics.

Output. A suboptimal access plan for the compiled query.

Method.

1. **Preprocessing** : (i) parse the query, (ii) create a connection graph which represents the data flow, (iii) initialize the current candidate graph, (iv) set the upper-bound cost C_{min} to *maxint*, and (v) order the edges of the candidate graph by the heuristics discussed before.

2. **Enumeration** : Generate the access plans and select one.

   Procedure *enumeration*(g_i, G_i, C_i, P_i)

   /* The current candidate graph g_i is a subgraph of the current connection graph G_i, C_i is the current accumulated estimated evaluation cost, P_i is the path from the root of the access plan tree respectively. */

begin

    if g_i only contains a single *available* node /* All nodes merged into one */

        then if $C_i < C_{min}$

            then $\{C_{min} := C_i;$

                update the suboptimal path to the current path P_i; }

        else /* the current candidate graph has operation candidate. */

            for each candidate operation o **do**

                {

                $C_o :=$ estimated cost of operation o on input data;

                $C_i' := C_i + C_o$;

                if $C_i' > C_{min}$

                then return; /* The path is pruned since it is unpromising. */

                else

                  {

                  append operation o to path P_i to form P_i'

update the current g_i and G_i to the new g_i' and G_i'

/*The detail of the graph update is presented in the note. */

$enumeration(g_i', G_i', C_i', P_i')$;

}

}

end

Note: The graph updates are performed for two legitimate operations as described below:

[1] Join two *available* nodes.

(i) merge the vertices and edge(s) involved in operation o into a new node representing the output relation of o; and (ii) link corresponding edges to the new vertex and merge edges sharing the same vertices.

[2] Evaluate an *input-available* node.

(i) update the node status to *available*, (ii) modify the status of the nodes taking input from it if necessary, and (iii) include the new *input-available* nodes and the appropriate edges into the updated candidate graph. □

Theorem 5.1 *The dynamic connection graph transformation method generates sub-optimal access plans.*

Proof Sketch : *The proof is based on the following observations.*

Observation 1 The candidate graph presents all possible choices at any given time and therefore represents all possible access plans.

Observation 2 A spatial procedure is evaluated only after all of its *input* parameters instantiated. Join operations are only performed on two *output-ready* nodes.

Observation 3 The generated access plans correspond to spanning trees of the connection graph. Every time an edge is selected, two nodes are merged. The process terminates when all nodes have been merged into one. There are no loops formed by the selected edges, because all edges adjacent to merged vertices are connected to this new vertex and edges joining the same vertices are merged. The resulting set satisfies all predicate conditions.

Observation 4 The generation of access plans is aborted only if the partial cost has exceeded the upper-bound cost. □

5.1.3 Cost estimation and selection of access plans

Cost estimation is crucial in the control of the generation of access plans and in the selection of generated access plans. Cost estimation and access path selection have been studied extensively in relational database systems. Some considerations specific to spatial data processing should be integrated into spatial query optimization.

First, the cost of processing a spatial routine can be estimated based on the history of the processing and the size of the input parameters. For example, the cost of processing $geo_intersection(X, Y)$ grows proportionally to the number of sides of each input polygon. The estimated cost could be formulated as $\mu \times number_of_sides(X) \times number_of_sides(Y)$, where μ is a coefficient obtained from experiments. More precisely, a cost estimation table can be built based on the execution history and different characteristics of input parameters and spatial computation.

Second, special characteristics should be taken into consideration in the cost estimation. One nice property of many spatial predicates is that their output parameters are functionally dependent on their input parameters. For example, in $geo_intersection(X, Y, Z)$, at most one spatial object Z can be generated by taking a pair of spatial objects X and Y as inputs.

Example 5.3 We examine the cost evaluation process to observe the difference between the traditional three-way join optimization and the connection graph transformation approach.

Let $\mid A \mid$ be the number of tuples in relation A, and $n_{a \in A}$ be the number of distinct values of attribute a in relation A. Assume that $\mid c_land \mid = \mid t_land \mid = 1000$, that $\mid ct_land \mid = 3000$, and that $n_{X \in c_land} = n_{Y \in t_land} = 100$.

Plan P_1 starts with *c_land* ⋈ *t_land*, and the derived 3000 tuples of *ct_land* form the input of *geo_int*. The evaluation of *geo_int* may generate fewer tuples, say 300, for further computation. Spatial operators which take two input parameters from two relations must pairwisely combine the two, i.e. compute the cross product of the two input sets. Plan P_2 starts by evaluating *geo_int*. Since there are only 100 distinct values in X and Y from *c_land* and *t_land* respectively, the execution input size is their Cartesian product, 100×100, i.e. 10000 tuples. Assuming the same rate for tuples satisfying *geo_int* and *area* as that for *geo_int* in plan P_1, 1000 tuples are fed into *area* and resulting *geo_int_area* with 100 tuples. *geo_int_area* then participates in three-way join with *c_land* and *t_land*. □

Cost estimation leads to a sharp contrast between these two plans. The total cost of P_1 is the sum of the cost of joining two relations with 1000 tuples, the cost of performing 3000 spatial operation *geo_int* and the cost of computing *area* 300 times. On the other hand, the cost of plan P_2 is the total cost of performing 10000 geometric operation *geo_int*, 1000 evaluation of operator *area* and of computing the three-way join of one relation with 100 tuples and two 1000-tuple relations. Experiments have shown that geometric operations are usually very expensive compared to relational operations. Obviously, the dominant component of the cost of the second plan is the 11000 geometric operations, which is likely to be much more expensive than the total cost of plan P_1. In this case, after comparing this partial cost of the plan P_2 with the upper-bound cost provided by plan P_1, the generation of all access plans starting with the evaluation of *geo_int* is terminated. Hence, it is important to use heuristics, such as *delaying geometric operations* in the selection of the processing order.

5.1.4 Analysis

It is well known that the number of possible plans is enormous in the case of a large graph. The efficiency of access plan generation depends on effective tree pruning. The branch-and-bound algorithm helps in pruning plans whose partial path cost exceeds the current upper-bound. With good heuristics, potentially low cost plans are generated at an early stage so that effective pruning can be achieved.

Simulation experiments have been performed to evaluate 3 different methods. A reasonable parameter for the cost estimation is the number of edges in the connection graph of a query, since each edge represents a possible join operation. The cost is simulated using the accumulated path length in the access plan generation. The simulation was performed on a SUN/4 SPARC/14.28MHz-workstation with 7 MIPS under the SunOS. The simulation program was written in Sun C compiler without optimization. The simulation results are presented in Figure 5.3 in which there are three curves *naive*, *branch_bound* and *heuristic* corresponding to three algorithms: *general search*, *branch-and-bound*, and *heuristic branch-and-bound*. Each curve illustrates the cost of access plan generation and selection as a function of the edges in the graph.

Curve *naive* shows that the general search algorithm soon becomes too expensive to utilize. Using the branch-and-bound algorithm, the search for a partial path whose cost exceeds the upper-bound value terminates, thus improving the performance of the algorithm as shown by the curve *branch_bound.* The heuristics algorithm promotes the early generation of low cost access plans and prunes less promising paths at an early stage, thus improving the computation effectively as shown in curve *heuristic.* Moreover, the maintenance of candidate graphs which contain only currently valid actions significantly improves computation efficiency.

In summary, the dynamic connection graph approach captures data flow constraints in spatial queries and facilitates effective access plan generation for them.

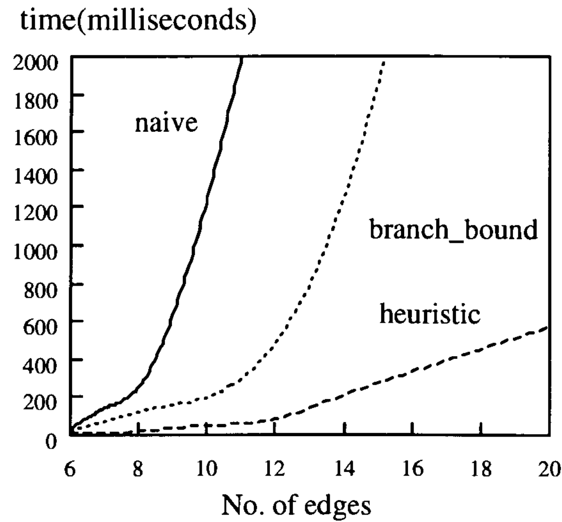


Figure 5.3: Simulation results for three access plan generation methods.

5.2 Set-Oriented Spatial Computation and Optimization

A major challenge in DOOS query processing is the impedance mismatch between tuple-oriented spatial computations and set-oriented relational computations. Spatial computations are usually performed by taking one particular vector of input parameters and performing costly spatial computation by accessing the corresponding spatial data structures. Such tuple-oriented spatial computation may lead to repetitive accesses of the same spatial objects and repetitive computations of the same or similar spatial primitives. Obviously, set-oriented evaluation should be promoted in spatial computation to minimize redundancy in disk accesses and spatial computations. The optimization techniques for set-oriented spatial computations are classified into the following groups according to the nature of optimizations:

1. precomputation and memorization of spatial information,
2. I/O control, buffer management, and pipelined processing,

3. set-oriented spatial method computation,
4. approximate or alternative operations with reduced complexity, and
5. rule-based and spatial semantics-based optimization.

5.2.1 Precomputation and memorization of spatial information

This set of techniques can be chosen by the statistics and other meta-information in the system. It is used for frequently invoked geo-predicates and functions.

Technique 1 *Materialization of properties of spatial objects and spatial predicates.*

The primitive spatial data, such as lines, points, polygons, etc. stored in the GDB are information at the primitive level, which are often several levels lower than the object-level information [1]. Spatial reasoning and query answering are often related to the information at the object-level. Although the nonspatial object-level information, such as the semantic meaning of a polygon, are often stored in the EDB and linked to the corresponding spatial data via pointers in the SAND architecture [7], some object-level properties, such as the area of a primitive spatial object denoted by $area(Sp_obj, Total_Area)$, are often defined by spatial algorithms (methods) or rules. If such object-level information is used frequently, instead of performing spatial computation at the query processing time, it is wise to perform precomputation and materialize (i.e. store) them as a spatial (EDB) relation. Such a materialized relation should be updated accordingly if the corresponding spatial data is modified. Similarly, for other frequently-used, relatively stable, and computationally intensive spatial predicates, materialization can also be performed by precomputation.

The materialized spatial predicates are treated as existing spatial relations. Indexing structures can be constructed on materialized relations to facilitate efficient access. Incremental updates can be performed on the materialized relations when

appropriate. With materialization, dynamic computation of such spatial predicates is transformed to simple data retrieval of materialized relations.

Technique 2 *Precomputation of spatial join relationships and construction of spatial join indices.*

Similarly to the materialization of the spatial predicates relevant to individual spatial objects, precomputations can be performed on spatial relationships between two or more spatial objects. For example, $adjacent(O_1, O_2)$ is a spatial relationship between two spatial objects.

The major difficulty for "materialization" of such spatial relationship is the potential huge size of the generated relation. Such a problem can be reduced by an information-associated spatial join index [94], which stores the join indices of the spatial relationship which meet certain conditions together with some frequently inquired information. For example, a distance-associated join index file in the format of $\langle O_1.oid, O_2.oid, distance(O_1, O_2)\rangle$ can be constructed to register the distance relationship between two spatial objects O_1 and O_2. To reduce the size of join indices, a hierarchically organized spatial join index file can be constructed to register the spatial join index relationship for spatial objects belonging to the same hierarchical level and located within the same local region (see the detailed discussion in Chapter 6). Spatial relationships between remotely connected objects can be computed based on the existing spatial relationships by accumulation, transitivity or other computation methods specified by rules or methods [94].

Technique 3 *Memorization of partially computed results.*

In the evaluation of spatial queries, repetitive and redundant computations of spatial components or subcomponents may occur within one query or one group of similar queries, e.g. in the derivation of composite spatial objects, or in comparison with a set of similar spatial queries, etc. A *dynamic tabular technique* described below can be explored to eliminate such redundant spatial computation. A small table is associated

with each spatial predicate or method to memorize the previously computed results. When a method is invoked, its input parameters are checked against the table to see whether such a set of parameters was computed before. If the corresponding entry is found in the table, the previously computed result is returned. Otherwise, the method is invoked, and the table is updated with the insertion of the newly computed results. Notice that a set of parameters may sometimes generate uninteresting results, such as value-out-of-bounds, domain-out-of-interest, not-computable, etc. Such precomputed information should also be registered in the table to avoid repeating such uninteresting computation as well. The technique is illustrated in the following example.

Example 5.4 Let $land_type(O, T)$ be an EDB relation in which O is a spatial object and T is the land type of the object, and *adjacent* and *geo_union* be two spatial predicates. Figure 5.4(a) shows the land pieces. Suppose that a method $geo_max_union(T, O)$ is a spatial procedure which finds the maximum size of land pieces suitable for planting crop type T and returns the result in O, and there are three types of crops, t_1, t_2 and t_3.

Given a type t and a set of *land* objects, the major steps involved in finding the maximum continuous land pieces suitable for planting a type t crop are as follows.

begin

O := ϕ;

for each object o in *land* **do**

  if $land_type(o, t)$ **then** O := O $\cup$ $\{o\}$;

while there exist uncomputed object pairs (o_1, o_2) in O **do**

  if adjacent(o_1, o_2) **then** O := O $-$ $\{o_1, o_2\}$ $\cup$ $\{geo_union(o_1, o_2)\}$.

end

For the land pieces shown in Figure 5.4(a), three types, t_1, t_2 and t_3 with suitable land pieces are inquired, which should return {a, b, c, d, e}, {a, b, c, d, e, f, g} and {a, b, c, f, g} respectively. We assume that the name of the union of two land pieces

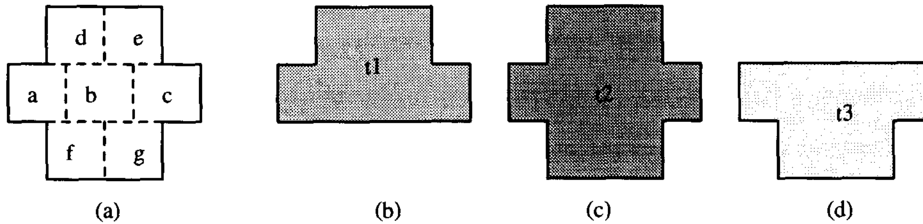


Figure 5.4: Derivation of maximum land pieces suitable for planting crops

is obtained by concatenation of the names of the pieces. Let us consider only the *geo_union* operation. Without using the tabular method, the execution sequences are as follows.

$t_1 : \underline{a \cup b},\ \underline{ab \cup c},\ \underline{d \cup e},\ \underline{abc \cup de}$
$t_2 : a \cup b,\ ab \cup c,\ d \cup e,\ abc \cup de,\ \underline{f \cup g},\ \underline{abcde \cup fg}$, and
$t_3 : a \cup b,\ ab \cup c,\ f \cup g,\ \underline{abc \cup fg}$

A total of 14 *geo_union*'s are executed. By replacing repetitive spatial computations with simple table lookups, the computation will involve only 7 *geo_union*'s shown with underlines. Figure 5.4(b)-(d) shows the maximum size of land pieces suitable for planting different kinds of crops. Since the spatial union is more costly than a simple table lookup, the saving of intermediate computation results and the transformation of tuple-oriented spatial computations into set-oriented data retrieval may significantly reduce the cost of query processing. □

5.2.2 I/O control, buffer management, and pipelined processing

This set of techniques are commonly used in operating systems and database systems. However, it is more crucial in spatial databases. Many techniques developed in these areas can be applied to spatial query processing.

Technique 4 *Set-oriented retrieval of spatial objects.*

Retrieval of spatial objects and their associated spatial data structures takes a major portion of the spatial query processing cost. Repetitive accesses of the same set of spatial objects may happen when computing a spatial join, executing the same method within a loop, or using the same set of spatial objects in a nested loop within one method execution. Repetitive spatial data accesses may substantially degrade the system performance.

Set-oriented secondary-storage access and intelligent buffer management can be explored in a way similar to that used for the optimization of join operations in relational database systems. Take spatial join as an example. When a spatial join is performed on two sets of input spatial objects, the two sets should be fetched in a set-oriented manner and the fetched spatial objects should remain in main memory in case they need to be accessed by future join operations. Several interesting spatial join techniques have been developed by Aref and Samet [7], such as intersection of spatial-ids or tuple-ids, pushing spatial operators into $sp_{extract}$ or pushing database operators into $db_{extract}$, etc. These techniques should be explored here to improve the performance of spatial joins. If the data volume is too large to fit in the main memory, a reasonable ordering of data access, such as putting the smaller set of spatial objects in the inner loop of a nested loop join will reduce I/O access in the spatial join. This is similar to the optimization of nested loop join in relational query processing.

Technique 5 *Creation or destruction of temporary data structures: repetitive vs. pipelined processing.*

If some intermediate spatial or relational data is to be retrieved repetitively in the computation, temporary access structures, such as indexing structures, can be created to reduce the access cost. On the other hand, if some intermediate spatial data will never be used again, modification can be performed directly on the spatial data dynamically generated (note that such modification destroys the intermediate spatial data or indexing structure) in the sequential or pipelined processing [7].

5.2.3 Set-oriented spatial method computation

This set of techniques are developed for minimizing the effect of impedance mismatch and reducing redundant geometric computation.

Technique 6 *Loop and block optimization within method execution.*

Similarly to loop and block optimizations in optimized compiler construction, optimized processing can be further explored within the method execution. Data flow analysis can be performed on a reasonably well-structured method to identify the expensive or repetitively computed part (such as looping or block structures), especially for the costly spatial computations. Shared computation of common subexpressions, group fetching and set-oriented computation, saving of intermediate results, etc., are useful techniques to optimize the execution of a single method. Many code optimization techniques developed in compiler construction can be applied to the optimization of spatial method computations.

Example 5.5 Common subexpressions within a nested loop can be moved outside of the loop to save repetitive computations of spatial subroutines in the method computation. Suppose that the method contains the following block of code,

for i := 1 **to** m **do**
  for j := 1 **to** n **do**
    $total_area[i,j] := geo_area(polygon_1[i]) + geo_area(polygon_2[j]);$

Suppose $area(polygon1[i])$ and $area(polygon2[j])$ are costly spatial computations. By data flow analysis, the code block can be transformed into the following code block,

for i := 1 **to** m **do**
  $tmp_1[i] := geo_area(polygon_1[i]).$

for j := 1 **to** n **do**

$tmp_2[j] := geo_area(polygon_2[j])$.

for i := 1 **to** m **do**

for j := 1 **to** n **do**

$total_area[i,j] := tmp_1[i] + tmp_2[j]$;

Obviously, the unoptimized code involves $2 \times m \times n$ costly spatial computations; whereas the optimized code involves only $m + n$ spatial computations. □

Technique 7 *Set-oriented execution of spatial operators and methods.*

In a nested loop computation or recursive query evaluation, a spatial method/operator is often executed iteratively. Calls to the same method using similar parameters may indicate the potential of repetitive execution. Smart execution ordering and the saving of partially computed results are important heuristics in the optimization of such queries.

Example 5.6 In the computation of the areas for a set of pairwise adjacent spatial objects, one may use the following query expression:

$$adjacent(X,Y), Total_area = area(geo_union(X,Y)).$$

Suppose that two adjacent objects do not overlap (which may be indicated by a deduction rule). The expression can be simplified as follows.

$$adjacent(X,Y), Total_area = area(X) + area(Y).$$

Furthermore, the above expression can be processed by taking advantage of set-oriented evaluation. For a given set of spatial objects, the computation can be performed by fixing each spatial object X, computing its area $area(X)$, then checking

against every other spatial object Y to see whether it is adjacent to X (which can be done efficiently using an R-tree or another spatial indexing structures), computing $area(Y)$ only if Y is adjacent to X, and finally returning the sum. Also, for each computed Y, an entry of $\langle Y, area(Y)\rangle$ can be inserted into a temporary table. By such optimization, a region which is not in the answer set will not be computed; furthermore, the area of each (X or Y) region in the answer set needs be computed only once. □

5.2.4 Approximate or alternative operations with reduced complexity

Using alternative operations is based on the following ideas, (i) use simplified computation where high precision is not required, and (ii) perform refinement only on the area which may possibly satisfies the query.

Technique 8 *Simplified, alternative spatial operations.*

Spatial routines carrying the same name (*overloaded*) with different input or output requirements (i.e. different parameter specifications) may be processed by different implementations with dramatically different processing complexity. For example, for the same (overloaded) spatial predicate *geo_intersection*, testing whether two spatial objects (such as two regions) intersect is much less expensive than computing their intersection. An expert user will use the less costly operator *is_geo_intersected* in the query rather than *geo_intersection*. However, a smart query optimizer should not rely on the user's expertise but should select the less costly operator automatically by query requirement analysis (such as the examination of the inquired variables). As another example, if a query is interested in nonspatial/spatial features only, the computation of spatial/nonspatial operations can be avoided. Query analysis, which involves parameter specification analysis and query requirement analysis, can be performed to identify the necessary operations and reduce more expensive spatial operations to less expensive ones to save processing cost while still serving the user's interest.

Example 5.7 Let a spatial query be to print the names of the major highways which pass through the city of Vancouver. Suppose that a relation $region(Region_Name, Region_Geo)$ is stored in the EDB with *Region_Geo* pointing to the corresponding region in the GDB, and the predicate major_highway is defined by a deductive predicate $major_highway(Highway_Name, Highway_Geo)$. The query can be formulated in the logic syntax as follows.

$$? - major_highway(?H_Name, H), region(vancouver, R),$$
$$geo_intersection(H, R, H_segments).$$

Note that "?" in front of H_Name indicates that only H_Name is inquired in the query. Based on the above analysis, one processing plan proceeds as follows:

1. Perform selection in the EDB relation *region* using the region name "*vancouver*", and retrieve its geo-region R by following the corresponding geo-pointer;

2. Find each road, H in the road_map which is a *major_highway* based on the compiled deduction rule.

3. Check whether each selected major highway H intersects the geo-region R. If it is, print the name of H.

Notice that in this computation, no full $geo_intersection(H, R, H_segments)$ is performed, instead, only a less costly operation $is_geo_intersected(H, R)$ is performed. Furthermore, no temporary map for the major highway H which are intersected with Vancouver is created in the computation since the user is only interested in the name rather than the spatial entity of H. Obviously, other processing plans, such as *first finding the roads in the region of "vancouver" and then checking whether it is a major highway, etc.*, are also possible. The one which takes the best advantages of the characteristics of the query should be the most efficient one for query evaluation. □

Technique 9 *Approximate computation based on a multi-resolution spatial data model.*

Another promising direction in the optimization of spatial computations is to perform approximate computation based on a multi-resolution spatial data model. A multi-resolution spatial data model can be constructed based on different granularity of resolution in the spatial database [117]. For example, a relatively low (coarse) resolution can be constructed to approximate the original high-resolution database, which results in a less precise but much smaller sized spatial database. Computation can be performed first on such a smaller spatial database to locate the interesting regions and derive approximate results. Refined computations are performed in the high resolution database, only when necessary, on those interesting regions.

A recent study on efficient computation of spatial joins by Günther [51] proposes a generalization tree technique to speed up spatial join computation, which can be viewed as another interesting example of implementation of spatial operations by a multi-level or hierarchical approach: A higher-level (bigger) region is first examined to filter out those regions which cannot participate in the spatial join. Only those which have not been filtered out will be examined in a refined (lower-level) region.

5.2.5 Rule-based and spatial semantics-based optimization

Rule-based optimization is commonly used in extensible databases. In spatial database system, optimization rules can provide optimizer with information about user-defined spatial operators. Rule-based constraint check may eliminate impossible solutions.

Technique 10 *Static constraint/rule enforcement in deductive query compilation.*

The techniques for static enforcement of constraints and rules in deductive query compilation are presented in Chapter 4. Such constraint enforcement will restrict the compiled query expression to a reduced set with appropriate types, modes, and spatial operations associated to a simplified compiled expression.

Technique 11 *Rule/heuristic-based query plan generation.*

Because of the large search space in the generation and selection of suboptimal query evaluation plans, it is beneficial to specify query optimization rules and heuristics by experts and use them in rule-based query plan generation. Query optimization using expert query transformation/optimization rules has been studied in extensible database systems [41, 47]. GEO-Kernel [143] and Gral [54] are two spatial database systems based on extensible architectures which use rules to describe query transformation and to choose among different implementations of primitive database operations and application-dependent operations. Güting [13] studied many-sorted algebra supporting extensibility for spatial applications and proposed a translation-rule-based query optimization method for Gral.

Technique 12 *Dynamic constraint enforcement in method and query evaluation.*

A spatial algorithm or a spatial operator may pose constraints on the characteristics of the spatial objects to be generated. For example, a *geo_union* algorithm may pose constraints on the number of polygon edges generated. A spatial operator, such as *area*, may have the following mathematical property:

$$area(geo_union(A, B)) \leq area(A) + area(B).$$

A user/expert may pose constraints explicitly as part of the query. These constraints can be enforced in the processing by *pushing the constraints as deeply as possible* to filter out the objects which cannot satisfy such constraints at the earliest stage.

Similar to query optimization in extensible spatial database systems [54], rules and integrity constraints can be used in the optimization of method and query evaluation. For example, using the constraint, $north_of(X, Y), north_of(Y, X) \implies \phi$, one can filter out the pairs of objects that cannot satisfy the constraint at the method or query computation time. Integrity checking can also be performed by precomputation using constraint networks [36]. The application of other kinds of integrity constraints can also be explored in the method or query evaluation.

Technique 13 *Optimization in the computation of aggregate functions based on the semantics of aggregation.*

Spatial aggregate functions are popular in spatial queries. Many spatial aggregates, such as the shortest traversal distance, the maximum-sized region, etc., require the extraction of the maximum or minimum values from all the possible spatial combinations, which could be very costly in computation. However, the nature of the problem may indicate that it is often unnecessary to compute all of the possible spatial combinations. Heuristics and monotonicity properties can be applied to prune the search space in computation. Further, saving intermediate results will facilitate such optimization.

Example 5.8 Let the query be to find the shortest driving distance between two spatial points (such as two buildings) p_1 and p_2. Since the shortest distance is a monotonic function, the monotonic behavior of spatial operators can be explored in the computation. For example, saving the currently derived minimum driving distance and its associated path for a set of frequently referenced pairs of spatial points (such as the major road intersections) will be useful at pruning any path with longer driving distance than the current minimum one. Also, the driving direction information may also help guide the search. Furthermore, all the paths computed so far with accumulated distance greater than the currently computed minimum distance between p_1 and p_2 can be dropped automatically in further computation. □

In summary, set-oriented spatial computation techniques can be used to improve spatial query processing. The application of some of the techniques can be rule-based. Another techniques can to be integrated into the optimizer. The implementation aspect of the technique will be studied in future research.

5.3 Chapter Summary

We have studied deductive spatial query optimization and developed a dynamic connection graph transformation approach for query plan generation and selection. A candidate graph has been proposed to dynamically maintain currently available operation alternatives; it models the data flow constraints among data relations and spatial predicates. A systematic transformation from the connection graph to access plans has been presented. Data statistics and empirical cost estimates of spatial operators are used in cost estimation. Heuristics are applied for preliminary ordering of different execution options. Thus, the enumeration algorithm generates potentially promising access plans first, thereby providing a tight upper-bound for pruning less promising plans at an early stage. Our preliminary experiments indicate that such an optimization mechanism effectively generates a suboptimal access plan for a given deductive spatial query.

The dynamic connection graph transformation and optimization techniques are not limited to spatial database applications. They can be applied to other kinds of database systems which integrate data relations with complex data types and procedural methods.

In addition to the enumeration approach, other approaches, such as randomized search, generic search, etc. have been developed in relational query optimization [88, 136]. These approaches, in principle, should be applicable to spatial query optimization as well. It is an interesting research issue to apply statistical approaches to spatial query optimization. The techniques listed above represent a set of interesting techniques for set-oriented processing and optimization of spatial queries integrated with spatial computational methods and deduction rules. Many other query optimization techniques can be further studied and developed in this direction.

Chapter 6

Information-Associated Spatial Join Index

6.1 Introduction

Spatial range search can be specified using a viewing window. A commonly used window may be a rectangle, a circle or an annulus. Circular range search is particularly useful in daily life, environment sciences and aviation. For example, a hazard situation at a place may affect the region within a specified distance.

Spatial join indexing proposed by Rotem [119] is a promising approach for answering queries involving intersection, containment, etc. However, many spatial queries are related to spatial ranges. The proposed mechanism explored only ϵ-overlap, which is a fixed distance mapping. It is difficult to construct a large number of spatial join index files corresponding to different query distance values. Moreover, it is impossible for a database designer to anticipate and enumerate all kinds of query ranges. Therefore, simple spatial join indices may not be effective for queries involving various distances or distance ranges among objects. For example, when a query is to find all spatial objects whose distance from a given spatial object ranges between 10 and 20

kilometers, the ϵ-overlap approach encounters difficulties since its indexing structure cannot express this kind of distance range constraints.

In this chapter, a general and flexible indexing structure, the *spatial-information-associated join index*, is proposed and investigated. The idea behind the new join indexing structure is to associate with each join index record some piece(s) of information (referred to as *information attribute(s)*), which registers some important information related to this pair of spatial objects. By precomputing this information at the spatial join index construction time, the computational cost of spatial data retrieval can be reduced substantially. Two important and frequently inquired spatial measurements, *distance* and *orientation*, are taken into consideration. By associating one or both measurements, the spatial-information-associated join indexing structure provides an efficient way to answer spatial queries, especially, spatial range queries.

A **distance-associated join index** structure will be the first focus of our investigation. It is a 3-tuple structure which contains two related object identifiers and the distance between them. The distance between two spatial objects could be the geometric distance between their reference points, the shortest highway distance between them, their Manhattan distance, etc. depending on the application. In general, this distance information is quite costly to compute. Obviously, the precomputation and registration of this distance information at index construction time may substantially reduce the computational cost at query processing time. By organizing the distance-associated join index records into B+-trees, many complicated distance-related queries, such as distance-range queries and nearest neighbor queries, can be answered efficiently.

Based on a basic distance-associated join index structure, two structured distance-associated join indices, *ring-structured* and *hierarchical*, are proposed to enhance search performance in sophisticated geometric environments. Ring-structured distance-associated join indices partition the join index file into several index files based on certain distance ranges. A query related to a given spatial range only needs to access those ring index structures which overlap with the inquired range. Hierarchical

distance-associated join indices resemble multiple-scaled maps, where smaller objects such as houses and buildings are associated with nearby objects such as highway intersections or major buildings. This method reduces the overall join index file size, and it can be used for hierarchically organized spatial environments. An important application of the hierarchical distance-associated join index is the search for the shortest distance between two spatial objects. Finding the shortest path can be accomplished by a sequence of join operations.

By adding an orientation attribute, the distance-associated join indexing structure can be further extended for efficient processing of queries relevant to *distance* and/or *orientation*. Such a **spatial-information-associated join index** structure is a 4-tuple structure containing both distance and orientation information. Thus, the basic distance-associated join index is extended to a basic spatial-information-associated join index. Two dimensional indexing structures can be constructed based on the preference of either distance or orientation. Moreover, a zone-structured join index can be constructed from the ring-structured distance-associated join index with angular subdivision. Such zone-structured join indices partition the join index file into zones based on the distance and orientation ranges. Thus, objects located in the same "zone" are clustered in the join index file. Similarly, a hierarchical distance-associated join index can be enhanced with directional information to form a hierarchical spatial-information-associated join index. In general, a spatial-information-associated join index structure stores the precomputed distance and orientation information. It reduces or avoids geometric computation at query processing time and thus substantially improves the performance of spatial query processing. This is the motivation for the construction of spatial-information-associated join indices.

6.2 Distance-Associated Join Indices for Distance Range Search

Spatial joins are common in databases which store images, pictures, maps and drawings. Such joins are costly to compute, hence the use of spatial join indices can be valuable, provided that they can be created and maintained efficiently. Since many spatial joins are performed among spatial objects within certain distance ranges, and since many other distance-related joins can be considered as special cases of spatial range joins, our design pays special attention to *spatial range queries*, which inquires about certain spatial objects in relation to other spatial objects within a certain distance range.

As an example of range queries, consider a region map database in which schools, galleries and regional parks, are marked as points or small regions. The following kinds of spatial range queries are often made.

1. Given a location, find regional parks that are beyond 30 miles but within 60 miles of this location.

2. Find every gallery-school pair whose distance is less than 1 mile.

The distance-related predicate of these queries can be abstracted into the following form,

$$D_{min} < distance(A, B) \leq D_{max},$$

where D_{min} and D_{max} are variables.

Notice that $distance(A, B) \leq D_{max}$ can be a special case in which $D_{min} = 0$. To facilitate spatial joins in complex environments, three kinds of distance-associated spatial join indices are proposed and studied in the following three subsections: *basic, ring-structured* and *hierarchical.*

6.2.1 Basic distance-associated join index

Basic distance-associated join indices (*basic DJI*, or *BDJI*) are an indexing mechanism which associates with two spatial objects (i.e. their identifiers) a piece of distance information. It optimizes distance-related queries by computing the distance between every pair of static spatial objects at index construction time rather than at query processing time. Therefore, the information about the distance between any two spatial objects is available in the spatial join index at query processing time.

6.2.1.1 Definition and construction

Definition 6.1 Given two spatial relations R_1 and R_2, the **basic distance-associated join index** records are generated by associating with each pair of objects $o_i \in R_1$ and $o_2 \in R_2$ the distance between them.

$$BDJI = \{\langle o_i, o_j, d_{i,j}\rangle \mid o_i \in R_1 \wedge o_j \in R_2 \ \wedge \ d_{i,j} = distance(o_i, o_j)\},$$

where $distance(o_i, o_j)$ is a function which takes two object identifiers o_i and o_j and returns the distance between these two objects.

Notice that $distance(o_i, o_j)$ is usually defined according to the specific application. For example, the distance between two buildings can be defined as the distance from the center of one building to the center of the other, the shortest distance between them, or their Manhattan distance (the shortest street distance), etc. The computation of distance may involve I/O and complex geometric computations and is thus a relatively expensive process. The distance attribute in the join index record can be used to directly answer queries about the distance between two spatial objects, or answer spatial range queries by comparing it against the distance constraints provided in the queries.

The basic distance-associated join indices are constructed as follows. Join indices are sorted first by the first attribute, so that queries related to a specified object can

be answered efficiently. B+−tree is built on the primary index. Records with the same first attribute value are then sorted according to their distance attributes. To speed up search for range queries, a secondary index is created based on the values of the distance attribute. We now present the algorithms used for creation, retrieval, and maintenance of basic DJIs.

Algorithm 6.1 Creation of a basic distance-associated join index.

Input. Spatial object relations R_1 and R_2.
Output. Construction of a basic DJI.
Method.

1. For each pair of spatial objects o_i and o_j, where $o_i \in R_1$ and $o_j \in R_2$, compute their distance and generate an index record $\langle o_i, o_j, d_{i,j} \rangle$.
2. Sort the DJI records by their first attribute and construct the primary index for the DJI.
3. Sort every subset of index records with identical first attributes, sort the records according to the distance attribute and construct a secondary index. □

To simplify our analysis, it is assumed that R_1 and R_2 contain the same objects, that is, the DJI's thus constructed will reflect the spatial distance relationship within a unique set of spatial objects. It is straightforward to generalize the results to DJI's between two distinct sets of spatial objects.

Theorem 6.1 *The time complexity for the construction of the basic distance-associated join index of a database with N spatial objects is $O(N^2 \log N)$.*

Proof: Pairing up N objects and computing the distance between each pair, the first step takes $O(N^2)$. The time complexity of the second step, sorting N^2 records, is $O(N^2 \log N^2)$, i.e. $O(N^2 \log N)$. The size of the secondary index for each object is

$N-1$ and it takes $O(N \log N)$ to sort the records. There are N groups of such indices so the third step takes $O(N^2 \log N)$. Therefore, the overall computational complexity is $O(N^2 \log N)$. □

To illustrate the algorithm, the indices for three simple spatial objects are shown in Figure 6.1. Indices are first sorted by the first attribute, i.e. o_1, o_2 and o_3. Records with the same first attribute value are then sorted by their distance value. For example, for the same first attribute o_1, the second attribute o_3 is before o_2 because o_3 is closer to o_1 than o_2.

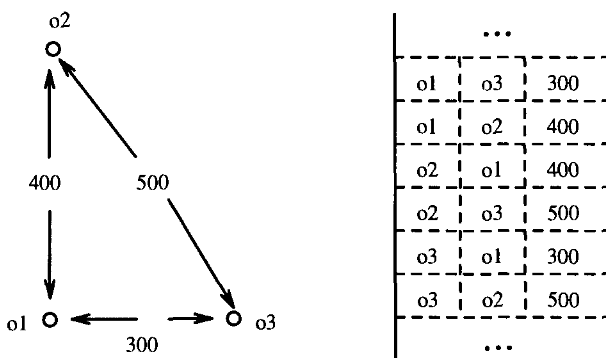


Figure 6.1: Indices for three spatial objects.

6.2.1.2 Retrieval of spatial objects

Suppose that a basic DJI file is constructed on N spatial objects. We now present a retrieval algorithm for the spatial range query: *Find all objects whose distance from object o_i is between D_{min} and D_{max}.*

Algorithm 6.2 Data retrieval for a typical spatial range query.

Input. (i) an object identifier o_i, (ii) the query distance range $(D_{min}, D_{max}]$.

Output. Every object in the database whose distance from o_i is within the range.

Method.

1. Search for object o_i using the primary index,
2. Search along the secondary index until the distance value reaches D_{min}, and
3. Read the leaf index records until the distance value becomes greater than D_{max}.

□

Theorem 6.2 *The computational complexity of the algorithm for retrieving a database with N objects using distance-associated join index is* $O(\log N + k)$.

Proof: Since the time complexity for the retrieval of an object in a database of N objects using B-trees is $O(\log N)$, each of the first and second steps takes $O(\log N)$ I/O time. The I/O cost in the third step, k, is proportional to the number of tuples satisfying the query divided by the number of index records stored in one data page. Therefore, the time complexity for retrieval is $O(\log N)$. □

This index structure also facilitates the processing of other similar kinds of queries. For example,

1. the search for the spatial object which is closest to a given one, which takes $O(\log N)$ time; and
2. the search for all pairs of spatial objects satisfying a given distance constraint, which takes $O(N \log N + k)$ time.

Example 6.1 Range search using basic DJI.

Given object o_2, find all of the objects whose distance from o_2 is between 300 and 450. Figure 6.2 shows a portion of the B+-tree. The search proceeds as follows. Search for o_2 using the primary index, which selects record R_1. Search for 300 using the secondary index, which obtains record r_1. Following the linked list at the leaf-level,

the search reaches record r_2 which is beyond D_{max}. Thus, the retrieval terminates with one resulting object o_1 returned. The search path is indicated using the dark arrows. □

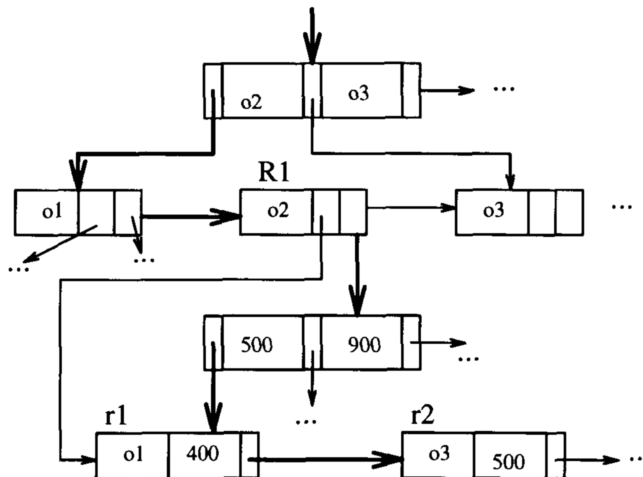


Figure 6.2: Processing a spatial range query using the basic DJI.

Not only can the basic DJI be created and used efficiently for data retrieval as shown above, but they are also easy to maintain. The following algorithm handles the insertion of spatial objects. It is easy to extend this algorithm to handle deletions and updates.

Algorithm 6.3 Updating the basic DJI after the insertion of a spatial object.

Input. Object o_i to be inserted

Output. An updated basic DJI after o_i is inserted into the database.

Method.

1. For each spatial object o_j in the database, construct the basic DJI record $\langle o_i, o_j, d_{i,j} \rangle$ and the record $\langle o_j, o_i, d_{j,i} \rangle$;

2. Cluster the index records whose first attribute is o_i and sort them according to the distance values;

3. Construct a B+-tree on this set of records and insert it into the existing B+-tree (for DJI's) as a branch;

4. Insert each record $\langle o_j, o_i, d_{j,i} \rangle$ into the existing B+-tree (for DJI's). □

Theorem 6.3 *The time complexity of the insertion algorithm used in an insertion of a database with N objects is* $O(N \log N)$.

Proof: Ignoring the cost of object distance computation as discussed previously, the first step takes $O(N)$ time to construct N basic DJI records. The second step takes $O(N \log N)$ time to sort them. The third step takes $O(\log N)$ to find the right place to insert the new set of index records. Finally, the fourth step requires $O(N \log N)$ time to insert N records. Therefore, computational complexity of the algorithm is $O(N \log N)$. □

Similarly, the deletion of a spatial object from the database will also take $O(N \log N)$ time. Since the basic DJI provides a reasonable cost for index maintenance, it is expected to be an interesting candidate to replace the runtime geometric computation of distances between sets of objects. However, since distance-associated join indices provide association among all pairs of spatial objects, the total number of index records to be maintained in the basic DJI file will be the square of the number of spatial objects in the database. As the number of objects in the database increases, the size of the basic DJI file will increase quadratically. It will thus be impractical to construct and store such a huge index file for relatively large spatial database.

Fortunately, many range searches in practical applications are confined to the vicinity of a spatial object. It is natural to specify a cutting radius (or scope value) for most spatial objects. For example, a fire station 60 miles away should be ruled out as a candidate to use in an emergency. In this case, 60 miles may be set as its

cutting radius. Two objects are related if and only if the distance between them is within their specified cutting radii.

6.2.2 Ring-structured distance-associated join index

Following the same philosophy of reducing the size of distance-associated join index files, a *ring-structured distance-associated join index* (*ring-structured DJI* or *RDJI*) can be constructed. The ring structure partitions one DJI file into several files based on different distance ranges. For example, the objects with a distance within 100 meters are partitioned into one ring, those with a distance between 100 to 500 meters are partitioned into the second ring, etc. For a query over a specified spatial range, the search can be confined only to those ring-structured DJI files which overlap the specified range.

Different standards can be used as the criteria in the partition or creation of ring-structured DJI's, such as equal-distance, equal-area, progressively increasing distance range, etc.

Definition 6.2 Given a set of n distance radii, each r_k specified by r_k = R(k), where R(k) is a function which decides the radii for ring k, a set of the **ring-structured distance-associated join indices** is a set of join index files, each of which is constructed corresponding to the rings specified by the radii, that is,

$$RDJI_k = \{\langle o_i, o_j, d_{i,j}\rangle \mid r_{k-1} < d_{i,j} \leq r_k\},$$

where k = 1, ..., n.

The index construction algorithm is similar to Algorithm-6.1. Figure 6.3 (a) shows a set of concentric rings centered at o_1 with equal-distance radii: $r_k = 10 \times k, k = 1, 2, 3$. Figure 6.3 (b) illustrates portions of ring-structured DJI files. For example, if a spatial range query inquires about spatial objects whose distance from each other between 10 and 20, only the file $rdji_2$ is searched.

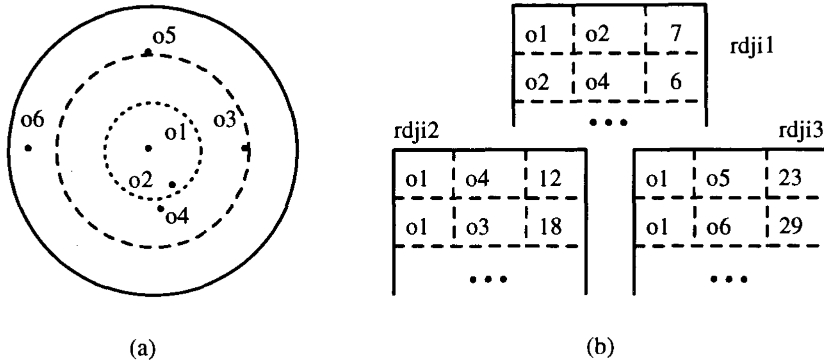


Figure 6.3: An example of a ring-structured DJI.

We now present a spatial range query retrieval algorithm using the ring-structured DJI.

Algorithm 6.4 Data retrieval for a spatial range query using the ring-structured index structure.

Input. (i) an object identifier o_i, (ii) the query distance range $(D_{min}, D_{max}]$.

Output. Every spatial object whose distance from o_i is within the specified range.

Method.

1. Select the ring files which overlap with the query range. A ring index file should be searched if (i) its lower or upper distance bound lies between D_{min} and D_{max}, or (ii) D_{min} or D_{max} is within its lower and upper distance bounds.

2. For each selected ring index file,

   [1] Search for the inquired object identifier using the primary index;

   [2] If the ring's range is completely covered by the query range, collect all objects in the index files related to the object identifier.

   [3] If the ring's range is partially covered by the specified query range, search in the secondary index to find and collect the portion which lies within the specified query range. □

The time complexity for processing a spatial range query using the ring-structured DJI structure should be the same as that using the basic DJI. However, because the ring structure partitions the basic DJI file into several smaller files, given a specified distance range, the files to be searched can be determined before accessing the database. Therefore, it is expected that the ring structure may improve the performance of query processing when compared with the basic DJI.

The ring structure has obvious advantages over the basic DJI when the spatial range of the query is covered by one or a small number of rings. If the full range of a ring is completely within the specified query range, there is no need to compare the distance values since every object in the ring whose first object identifier matches the specified object identifier satisfies the query. When the spatial range in a query involves many rings, it requires the search of the primary index of each involved ring. Searching many rings may add overhead to memory and buffer management. Therefore, we need to partition the distance-associated join indices wisely in the construction of ring-structured DJI files. For example, one may choose the radius for the innermost ring to be 100 meters for the queries on neighborhood, and the second ring to be from 100 to 1000 meters for those relevant to shopping, schooling, bus-stops, etc. By doing so, frequent inquiries are likely to use only one or a small number of rings.

6.2.3 Hierarchical distance-associated join index

Since both basic DJI and ring-structured DJI store all of the spatial object pairs of a spatial data relation, the size of the total index file(s) is proportional to the size of the cross-product of the spatial relation with itself (notice that only the keys rather than full tuples are stored). For a reasonably large spatial data relation, it is often impractical and unnecessary to consider all pairs of spatial objects. For example, it is rarely useful to relate school buildings in one city to houses in another suburban city.

Hierarchical views are commonly used in solving spatial problems in a complex world. When scheduling a flight from one continent to another, most small cities

are ignored in the planning. When driving a car to work, most individual houses are omitted in the calculation. Based on a similar point of view and assumption, spatial objects can be partitioned and classified correspondingly to fit into maps with different scales.

Analogously to multi-scale maps, a *hierarchical distance-associated join index* (*hierarchical DJI* or *HDJI*) can be constructed to organize spatial objects into different levels. Within one city block, it could be useful to construct distance-associated join indices to represent distances between individual houses and street intersections. With a larger scale, only highway intersections or major buildings in the city will be represented in the join indices. Queries about the distance between your house and your friend's in another suburban city can still be answered by referring to more than one hierarchical join index file.

Suppose that a spatial data relation consists of l interrelated object sets R_1, R_2, $\ldots$, R_l, with different scale scope values S_1, S_2, $\ldots$, S_l respectively, where the scope value of S_i is an order of magnitude larger than that of S_{i-1}. For example, the distance between two houses is on the order of 10 meters whereas the distance between highway intersections is at an order of 1000 meters. An object in R_i is at a *higher level* than one in R_j when $i > j$. Each object at level i has one parent object at level i+1, and this parent object should also be included in the join indices for class C_i. That is, object classes are constructed as follows:

Definition 6.3 Given R_1, $\ldots$, R_l, $C_{l1}, C_{l2}, \ldots, C_{ln_l}$ is a partition of R_l and $C_{i1}, C_{i2}, \ldots, C_{in_i}$ is a partition of $R_i \cup R_{i+1}$ where $i = 1, \ldots, l-1$.

A set of **hierarchical distance-associated join indices** are constructed on the object classes. The join index on C_k is constructed based on the following formula,

$$HDJI_k = \{\langle o_i, o_j, d_{i,j}\rangle \mid o_i \in C_{km} \wedge o_j \in C_{km}\},$$

where $k = 1, \ldots, n$ and $m = 1, \ldots, n_k$.

Figure 6.4 shows a simple example of a hierarchical tree.

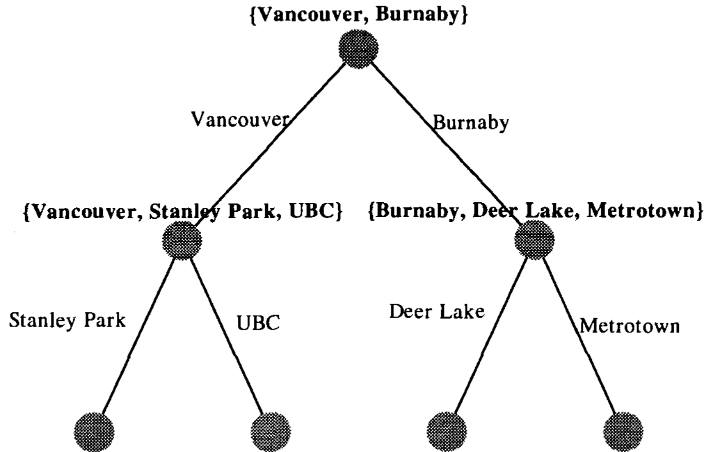


Figure 6.4: A sample hierarchy for a HDJI

The following theorem shows that hierarchical DJI reduces the space complexity for the distance-associated join index, where the *degree* for a nonleaf-node is the number of direct child-nodes, and the ratio between the maximum degree and the minimum degree of the hierarchy is assumed to be bounded by a constant.

Theorem 6.4 *The space complexity S_{ind} of the hierarchical distance-associated join indices of a database with N objects is*

$$\frac{N^2(K-1)}{2c^2(K^l-1)} - N \leq S_{ind} \leq \frac{2c^2N^2(k-1)}{k^l-1}$$

where the minimum and the maximum number of children of non-leaf nodes are k and K respectively, where c denotes $\lceil \frac{K}{k} \rceil$, and $l+1$ is the number of levels in the hierarchy.

Proof: Consider the tree T whose edges at a given level denote the spatial objects, whose internal nodes $v_1, \ldots, v_n$ represent related object sets, and whose m leaves are at level l. The number of index records corresponding to an internal node v_i equals the number of pairs of objects that belong to the object set stored at v_i.

Since $\operatorname{avg}\deg(v_i) = (2N-m)/n$, it follows that $N/n \leq \operatorname{avg}\deg(v_i) \leq (2N)/n$, and we get $\max\deg(v_i) \leq 2cN/n$ and $\min\deg(v_i) \geq N/(cn)$. Therefore, we see that

$$\begin{aligned} S_{ind} &= \sum_{v_i} \binom{\deg(v_i)}{2} \\ &= \frac{1}{2}\sum_{v_i} \deg(v_i)^2 - \frac{1}{2}\sum_{v_i} \deg(v_i) \\ &\leq \frac{1}{2} n \frac{4c^2N^2}{n^2} \\ &\leq \frac{2c^2N^2}{n} \end{aligned}$$

On the other hand,

$$\begin{aligned} S_{ind} &= \sum_{v_i} \binom{\deg(v_i)}{2} \\ &= \frac{1}{2}\sum_{v_i} \deg(v_i)^2 - \frac{1}{2}\sum_{v_i} \deg(v_i) \\ &\geq \frac{1}{2} n \frac{N^2}{c^2n^2} - N \\ &\geq \frac{N^2}{2c^2n} - N \end{aligned}$$

Because

$$\frac{k^l - 1}{k-1} \leq n \leq \frac{K^l - 1}{K-1},$$

we have

$$\frac{N^2(K-1)}{2c^2(K^l-1)} - N \leq S_{ind} \leq \frac{2c^2N^2(k-1)}{k^l-1}$$

□

When $l = 1$ the complexity is $O(N^2)$. Consider another instance where the HDJI is stored as a complete k-ary tree, then $k = K$ and $l = \log_k N$. Thus we get

$$S_{ind} \leq \frac{2N^2(k-1)}{N-1}$$

and therefore $S_{ind} \in \theta(kN)$.

6.2.3.1 Hierarchical DJI retrieval

Suppose that a range query is to find all objects within a certain distance range $(D_{min}, D_{max}]$ from a given object o_i. In most cases, solving a range query using the hierarchical DJI involves searching the hierarchical DJI's by climbing up and stepping down the hierarchy. Thus, a search can be partitioned into two phases: the *ascending phase* and the *descending phase.* First, find the class level l such that o_i is in R_l. Then, collect the objects whose distance from o_i is between D_{min} and D_{max} by joining the partitioned index files whose objects are located within the range. Suppose the current scope is s_1. If $s_1 < D_{max}$, climb up the hierarchy by joining the upper level spatial join index file. In the ascending phase, an object whose distance from o_i is less than D_{min} minus the lower level scope value will not be included in the intermediate relation T for later descending, because its descendants, distance to o_i will always be less than D_{min}. However, it should be included in the temporary relation *Temp* for further ascent since further ascending may generate satisfiable answers. In the descending phase, an object whose distance from o_i is less than D_{min} minus lower level scope value or greater than D_{max} will not be collected in T since its descendants cannot satisfy the query. At the end, only the objects whose distance from o_i lies between D_{min} and D_{max} will be included in the result relation. Notice that a newly generated index with a shorter distance will replace the index of the same object pair with a longer path distance. In other words, the object distance stored in every intermediate or final relation is the shortest path distance.

Algorithm 6.5 Processing of a range query using the hierarchical DJI.

Input. (i) a spatial object o_i, and (ii) the spatial range: D_{min} and D_{max}.

Output. All spatial objects o_j such that $D_{min} < distance(o_i, o_j) \leq D_{max}$.

Method.

1. [*Initialization*] (assuming $S_0 = 0$).

   Find *level* such that $o_i \in R_{level}$;

$level_i := level;$

$Temp := \{\langle o_i, o_j, d_{i,j}\rangle \mid \langle o_i, o_j, d_{i,j}\rangle \in HDJI_{level} \wedge d_{i,j} < D_{max}\};$

$T := Temp - \{\langle o_i, o_j, d_{i,j}\rangle \mid d_{i,j} + S_{level-1} < D_{min}\}.$

2. [*Ascending phase*]

while $S_{level} < D_{max}$ **do** {

$level := level + 1.$

$Temp := \{\langle o_i, o_k, d_{i,k}\rangle \mid \langle o_i, o_j, d_{i,j}\rangle \in Temp \wedge \langle o_j, o_k, d_{j,k}\rangle \in HDJI_{level} \wedge d_{i,k} = d_{i,j} + d_{j,k} \wedge d_{i,k} < D_{max}\};$

$T := T \cup (Temp - \{\langle o_i, o_j, d_{i,j}\rangle \mid d_{i,j} + S_{level-1} < D_{min}\})\}.$

3. [*Descending phase*]

while $level > level_i$ **do** {

$level := level - 1.$

$T := T \cup \{\langle o_i, o_k, d_{i,k}\rangle \mid \langle o_i, o_j, d_{i,j}\rangle \in T \wedge \langle o_j, o_k, d_{j,k}\rangle \in HDJI_{level} \wedge d_{i,k} = d_{i,j} + d_{j,k} \wedge d_{i,k} + S_{level-1} > D_{min} \wedge d_{i,k} < D_{max}\}\}.$

4. **return** o_j where $\{o_j \mid \langle o_i, o_j, d_{i,j}\rangle \in T \wedge d_{i,j} > D_{min}\}.$ □

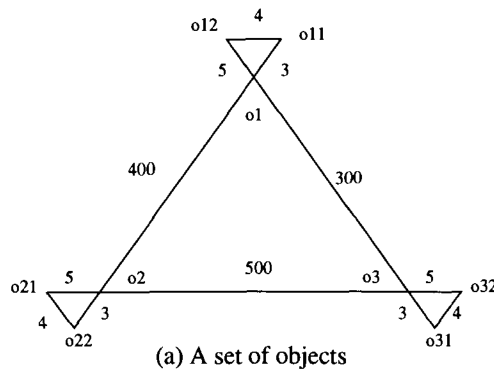
A detailed query execution process is presented in the following example.

Example 6.2 A range query on the hierarchical index.

Figure 6.5 shows a two-level hierarchical distance-associated join index (HDJI) where $S_1 = 1000$ and $S_2 = 10$ for a simple object setting. Given an inquired object o_{22}, find all objects whose distance from o_{22} is between 200 and 450.

1. First, search for object o_{22} in the hierarchical DJI.
2. By hierarchical ascent, $T = \{\langle o_{22}, o_1, 403\rangle\}$.
3. By hierarchical descent, $T = \{\langle o_{22}, o_1, 403\rangle, \langle o_{22}, o_{11}, 406\rangle, \langle o_{22}, o_{12}, 408\rangle\}$.

Notice that at the first iteration in the ascending phase, $\langle o_{22}, o_2, 3\rangle$ is included in the temporary relation *Temp* but not in the result relation T because the sum of 3 and 0 is less than the lower bound 200. However, it is used for the computation of the shortest distance to object o_1. The records related to o_3 are not generated because its accumulated distance to o_{22} has exceeded the upper bound of the range. Therefore, the answer to the query is o_1, o_{11}, o_{12}. □



(a) A set of objects

o1	03	300
o1	o2	400
o2	o1	400
o2	o3	500
o3	o1	300
o3	02	500

(b) level 2 index

o1	o11	3
o1	o12	5
o11	o1	3
o11	o12	4
o12	o11	4
o12	o1	5

o2	o21	5
o2	o22	3
o21	o2	5
o21	o22	4
o22	o21	4
o22	o2	3

o3	o31	3
o3	o32	5
o31	o3	3
o31	o32	4
o32	o31	4
o32	o3	5

(c) level 1 index

Figure 6.5: A simple two-level DJI and the index graph.

6.2.3.2 Hierarchical structure for shortest distance on a network

Interestingly, hierarchical DJI allows the shortest distance between two spatial objects be found efficiently, such as finding the shortest distance from a person's house to his/her friend's house. This can be realized by a simple modification of the data retrieval process presented in Algorithm-6.5. Notice that in this case the hierarchical

index structure $HDJI_k = \{\langle o_i, o_j, d_{i,j}\rangle\}$ indicates that for each pair of objects o_i and o_j, there exists a path between o_i and o_j, and $d_{i,j}$ is the distance of the shortest path between o_i and o_j.

The idea used for the shortest distance algorithm is as follows. Starting from the leaf level of the hierarchy, find the two inquired objects. If the two objects, o_i and o_j are directly related, the stored distance is returned and the search terminates. Otherwise, climb up the hierarchy in both directions. That is, starting at set T_i and T_j which contain objects o_i and o_j respectively, climb up the hierarchy by joining the hierarchical DJI file at one level higher than the current level and check whether the paths from the two sets reach a common node. At each level, newly generated records are stored in DT_i and DT_j, respectively. The paths which reach a common node form a complete path from o_i to o_j. For every pair of objects in the index construction and retrieval processes, only the one with the shortest path distance is kept.

Algorithm 6.6 Search for the shortest distance between two given spatial objects.

Input. (i) two spatial objects with identifiers o_i and o_j.
Output. The shortest distance between the two spatial objects in the spatial database.
Method.

1. Find o_i and o_j in the join index files using the primary indices.
2. $T_i := \{\langle o_i, o_k, d_{i,k}\rangle \mid \langle o_i, o_k, d_{i,k}\rangle \in HDJI_1\}$
   if there exists an o_k such that $o_k = o_j$,
    then return ($d_{i,k}$)
    else $\{T_j := \{\langle o_j, o_k, d_{j,k}\rangle \mid \langle o_j, o_k, d_{j,k}\rangle \in HDJI_1\};$
      $level_i := 1;$
      repeat
        $level := level + 1;$

$DT_i := \{\langle o_i, o_l, d_{i,l}\rangle \mid \langle o_i, o_k, d_{i,o_k}\rangle \in T_i \wedge \langle o_k, o_l, d_{k,l}\rangle \in HDJI_{level} \wedge$
$d_{i,l} = d_{i,k} + d_{k,l}\ \};$

$DT_j := \{\langle o_j, o_l, d_{j,l}\rangle \mid \langle o_j, o_k, d_{j,k}\rangle \in T_j \wedge \langle o_k, o_l, d_{k,l}\rangle \in HDJI_{level} \wedge$
$d_{j,l} = d_{j,k} + d_{k,l}\};$

$T_i := T_i \cup DT_i;\ T_j := T_j \cup DT_j;$

until there exists an o_k such that $\langle o_i, o_k, d_{i,k}\rangle \in T_i \wedge \langle o_j, o_k, d_{j,k}\rangle \in T_j$;

return $(d_{i,k} + d_{j,k})$ } □

Example 6.3 Shortest distance by hierarchical DJI.

Let the query be "*find the shortest distance between o_{11} and o_{22} in Figure 6.6*". The search proceeds as follows. At level one, $T_i = \{\langle o_{11}, o_{12}, 6\rangle, \langle o_{11}, o_{13}, 8\rangle, \langle o_{11}, o_1, 9\rangle\}$, and $T_j = \{\langle o_{22}, o_2, 7\rangle, \langle o_{22}, o_{21}, 8\rangle, \langle o_{22}, o_{23}, 9\rangle\}$. Clearly, the search should climb up the hierarchy. At level two, $DT_i = \{\langle o_{11}, o_2, 107\rangle\}$, $T_i = \{\ \langle o_{11}, o_{12}, 6\rangle, \langle o_{11}, o_{13}, 8\rangle, \langle o_{11}, o_1, 9\rangle, \langle o_{11}, o_2, 107\rangle\}$. Since $DT_j = \{\langle o_{22}, o_1, 107\rangle\}$ and $T_j = \{\ \langle o_{22}, o_2, 7\rangle, \langle o_{22}, o_{21}, 8\rangle, \langle o_{22}, o_{23}, 9\rangle, \langle o_{22}, o_1, 107\rangle\}$, the shortest distance between o_i and o_j is found, which is $107 + 7 = 114$. □

Notice that the above algorithm and the example find only the shortest distance between two spatial objects, but not the shortest path since the path information, which covers a sequence of intermediate nodes, is not maintained in the spatial join indices. Many applications need to find the shortest traversal path among a set of spatial objects. Such queries can be solved by modifying of the spatial join index record slightly. To find the shortest paths, an extra attribute *path* should be associated with the join index, which registers the shortest path between two spatial objects represented as a sequence of intermediate objects. That is, the join index record should be

$$HDJI = \{\langle o_i, o_j, d_{i,j}, p_{i,j}\rangle\},$$

where $p_{i,j}$ is the sequence of objects which form the shortest path from object o_i to object o_j. During join operations of hierarchical join indices, the two paths should be concatenated to form the shortest path between the spatial objects at different levels

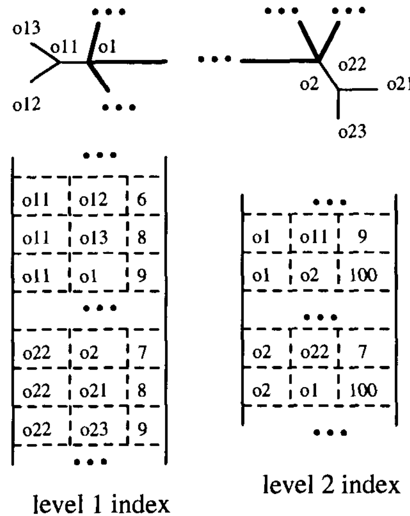


Figure 6.6: Search for the shortest distance between two spatial objects.

of hierarchy. Except for the data related to this path information, the remainder of the algorithm is the same as Algorithm-6.6.

6.2.4 Distance-associated spatial join index for nonzero-sized spatial objects

Although the previous discussions treat spatial objects as abstract points in their distance computation, distance-associated spatial join indices apply equally well to nonzero-sized spatial objects. For the basic DJI and the ring-structured DJI, the distances between two spatial objects (points/lines/polygons) can be defined according to particular applications, e.g. the shortest distance from a point to a line or the distance between the centers of two polygons. The algorithms developed for the basic DJI and ring-structured DJI can be applied directly to these cases.

It may not be so straightforward to apply the hierarchal DJI algorithm to construct and manipulate the hierarchically organized spatial join indexing structures for nonzero-sized spatial objects. A spatial hierarchy can be organized based on the

semantic structures of spatial data, such as administration hierarchy, or physical data structure, such as R-trees. For the hierarchical DJI, an additional spatial bounding information needs to be associated with each object.

At the leaf level, each nonzero-sized object o_i has a *minimum bounding rectangle* (MBR), denoted by $o_i.minx$, $o_i.miny$, $o_i.maxx$, $o_i.maxy$ referring to the $minimum_x$, $minimum_y$, $maximum_x$ and $maximum_y$ of its bounding rectangle. The MBR of a higher level object can in turn be constructed by finding the minimum bounding rectangle for those of all its child-nodes.

Given an inquired object o_i, the retrieval algorithm can be modified as follows.

Notice that the MBR of a parent(ancestor) node covers that of the inquire object. In the ascending phase, the criterion for stopping ascending a hierarchy at a parent node o_k is modified as follows,

$$minimum(o_i.minx - o_k.minx, o_k.maxx - o_i.maxx,$$
$$o_i.miny - o_k.miny,\ o_k.maxy - o_i.maxy) > D_{max}.$$

Similarly, in the descending phase, the condition for discard node o_j which cannot possibly satisfy the query consists of the following two parts.

1. Maximum distance is less than D_{min}

   Let $\triangle X$ and $\triangle Y$ be $maximum(o_i.maxx, o_j.maxx) - minimum(o_i.minx, o_j.minx)$ and $maximum(o_i.maxy, o_j.maxy) - minimum(o_i.miny, o_j.miny)$ respectively. The condition can be expressed as follows,

$$\triangle X^2 + \triangle Y^2 < D^2{}_{min}.$$

2. The minimum distance between the bounding rectangles is greater than D_{max}.

   Let x_{ij}, x_{ji}, y_{ij}, y_{ji} be defined as follows.

$$x_{ij} := \mid o_i.maxx - o_j.minx \mid;$$
$$x_{ji} := \mid o_j.maxx - o_i.minx \mid;$$
$$y_{ij} := \mid o_i.maxy - o_j.miny \mid;$$
$$y_{ij} := \mid o_i.maxy - o_j.miny \mid;$$

The second cutting criterion can be specified by the following Boolean expression

$$(minimum(x_{ij}, x_{ji}) > D_{max}) \ \vee \ (minimum(y_{ij}, y_{ji}) > D_{max}).$$

Substituting these algorithm control conditions into Algorithm 5, we have the hierarchical DJI retrieval algorithm for nonzero-sized objects.

6.3 Spatial-Information-Associated Join Index with the Orientation Component

Some spatial queries require information that are relevant to the relative orientation of spatial objects. For example, one may need to *find the schools in a district, located to the east of John's house.* Many other spatial queries, although they may not be directly indicating orientation values, may use the orientation information to reduce the search space. For example, to drive from your house to your friend's, it is important to have the direction information available. Therefore, in many applications, it is beneficial to construct **orientation-associated join indices**. Similarly to the case of distance-associated join indices, an index record should be in the form of $\langle o_i, o_j, angle_{i,j} \rangle$ where *angle* is the angle formed between the vector from o_i to o_j and a reference axis, such as the X-axis.

An orientation-associated join index can be constructed as follows. The range of the angular space for an orientation-associated join index, (usually $0°$ to $360°$), is divided into several zones defined by angle values. Index records are first sorted by the first attribute with a primary B+-tree structure. Objects with the same first attribute are then sorted by the angle attribute. A radix index can be used for fast

retrieval so that the secondary indexing retrieval time can be constant. The total retrieval time complexity for a database with N spatial objects is $O(\log N)$.

In general, these two frequently used pieces of information, *distance* and *orientation*, can be combined to form a relatively general **spatial-information-associated join index** . Many spatial queries are relevant to both of them, such as *finding all the restaurants within 1000 meters of the conference center and located to the east of it.* In such cases, the query is associated with two spatial predicates of the form

$$D_{min} < distance(o_i, o_j) \leq D_{max}, \ and \ A_{min} < angle(o_i, o_j) \leq A_{max},$$

where D_{min}, D_{max}, A_{min} and A_{max} are specified in the query.

Clearly, a general spatial-information-associated join index facilitates the processing of such spatial range queries. With information pertaining to both distance and orientation dimensions, a user can specify any two-dimensional range. Furthermore, since such a structure maintains both pieces of information, it facilitates the processing of spatial range queries relevant to distance only, orientation only, or their combination. The shortest distance or the shortest path problem addressed above can be solved more efficiently with the help of the available spatial orientation information.

The information-associated join index is a general indexing data structure. Its scope of use is application-related and could be best suited for the situations where (i) a given geometric information is frequently inquired about, (ii) the spatial information is computationally expensive, and (iii) the solution is simple. For example, if the areas of intersection of objects in two thematic maps are frequently inquired about, an area-associated join index may be built. Only when two regions are overlapping and an index record need to be stored in the index, such a structure may facilitate finding the intersection of a pair of specified objects and its area. Clearly, a general spatial-information-associated join index can facilitate the processing of many spatial queries.

6.3.1 Basic spatial-information-associated join index

As an extension to the basic distance-associated join index, a *basic spatial-information-associated join index* (*basic SJI* or *BSJI*) can be defined as follows.

Definition 6.4 *Given two spatial object relations R_1 and R_2, the* **basic spatial-information-associated join index** *records are generated by coupling object identifier pairs in R_1 and R_2 respectively with the information about the distance between them and their orientation.*

$$BSJI = \{\langle o_i, o_j, d_{i,j}, a_{i,j}\rangle \mid o_i \in R_1 \wedge o_j \in R_2 \wedge d_{i,j} = distance(o_i, o_j) \ \wedge \ a_{i,j} = angle(o_i, o_j)\},$$

where the $angle(o_i, o_j)$ is the angle that the vector $o_i \rightarrow o_j$ forms with respect to the X-axis in the range between $0°$ and $360°$.

According to this definition, it is obvious that $distance(o_j, o_i) = distance(o_i, o_j)$, and that $angle(o_j, o_i) = (angle(o_i, o_j) + 180\ )\ mod\ 360$.

The distance and angular attributes in the join index record can be used to directly answer queries about distance between two spatial objects, their orientation, the predicates containing these constraints or their combinations. They can also be used to answer spatial range queries by satisfying the distance and/or orientation constraints provided in the queries.

Two pieces of spatial information lead to the organization of the spatial indexing structures in two dimensions. Multi-key indexing structures, such as grid files [105], can be applied in the construction of spatial-information-associated join indices. Here we present a two-level structure for the construction of such join indices.

In order to efficiently process distance- and/or orientation- related queries, index records should be clustered. One may select a preferable attribute as the primary clustering attribute. Based on the operational frequency, different indexing priorities may be established. A *distance preference* index structure sorts the distance

attribute before sorting the orientation attribute, whereas an *orientation preference* index structure sorts the orientation attribute first and then the distance attribute.

Taking a distance preference indexing structure as an example, the basic SJI structure can be constructed as follows. First, the indexing records are sorted by the first object identifier attribute (primary index). A B+-tree is constructed on the primary index. Second, each subset of records with a given first attribute value is then sorted by the preferable attribute, which is the distance attribute in this case. Another level of index is created on the value of this attribute. Finally, each subset of the records with a given object identifier and a given distance attribute can be further sorted according to the orientation. The detailed construction and retrieval algorithms for the basic SJI are similar to those for the basic DJI.

Similarly to the case of basic distance-associated join index, the size of the basic SJI file increases quadratically when the number of objects in the database increases. It will be impractical to construct and store such a huge index file for a relatively large size of spatial database. In the following subsection, an extension to ring-structured DJI based on orientation subdivision, called a zone-structured spatial-information-associated join index, is designed for improved performance.

6.3.2 Zone-structured spatial-information-associated join index

The motivation behind the construction of a "zone"-like structure is to take into consideration the combination of distance and orientation information in the index clustering. Many queries need to consider spatial objects within a zone which often crosses the boundaries of both a refined distance and a refined angle. The zone structure partitions a spatial-information-associated join index file into groups based on different distance and orientation ranges. Objects which are close to each other in space are clustered in the index file. For example, the objects whose distance from a given object o_i is smaller than 100 meters are stored in one ring, those whose with

the distance from o_i lies between 100 and 500 meters are stored in the second ring, etc. Each ring is further subdivided according to angle ranges. The rings close to the center may be divided into fewer zones than the ones which are farther from the center. For a query with a specified spatial range, the search can be confined to only those zones in the SJI file which overlap the inquired ranges.

Different standards can be used as the criteria in the partition or creation of zone-structured SJI's, such as equal-distance-and-equal-angle divisions, equal-area divisions, progressively increasing distance range, etc. Let the area set which specifies the zone $z_{i,j}$ be $Z(i,j)$. The key to the zone division is that it must be easy to determine whether it overlaps with an inquired range. For instance, consider an example with equal-distance (a given $\triangle r$) and equal-angle (a given $\triangle a$) division,

$$Z(i,j) = \{\langle d,a\rangle \mid \triangle r \times (i-1) < d \leq \triangle r \times i \wedge \triangle a \times (j-1) < a \leq \triangle a \times j\},$$

for i = 1, ..., n and j = 1, ..., $360° \div \triangle a$.

Definition 6.5 *Given a set of zones, $z_{i,j}$ specified by $Z(i, j)$, a set of* **zone-structured spatial-information-associated join indices** *can be constructed which corresponds to the zones specified by $Z(i,j)$, i.e.*

$$ZSJI_{k,l} = \{\langle o_i, o_j, d_{i,j}, a_{i,j}\rangle \mid \langle d_{i,j}, a_{i,j}\rangle \in Z(k,l)\}.$$

Figure 6.7(a) shows a set of concentric rings centered at o_1 with equal-distance radii: $r_1 = 10$, $r_2 = 20$, $r_3 = 30$ and $\triangle a = 45°$. Figure 6.7(b) illustrates a portion of the corresponding zone-structured SJI file. In this example, if a spatial query requires information about spatial objects whose distance from o_1 ranges between 10 and 30, and whose orientation relative to o_1 ranges between 0° and 45°, only zone $ZSJI_{2,1}$ is searched.

A zone-structured SJI is constructed as follows. First, all index records in a given zone are clustered with an index pointer pointing to the beginning of the indices of the zone. Inside the zone, the indices may be sorted first by the distance and then

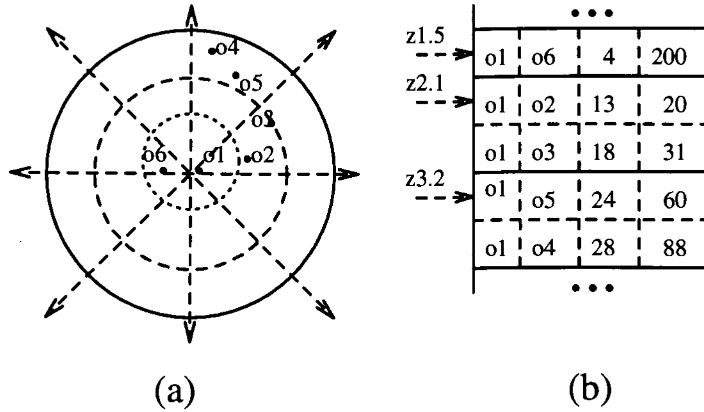


Figure 6.7: An example of zone-structured SJI.

by orientation. The size of the zone influences the efficiency of data retrieval. One index page can be used to hold the indices in each zone. Because of the regularity of the zone subdivision, given a distance range and/or an orientation range, the related zone(s) can be easily determined. There are four cases to consider for a selected zone with respect to a given query: a zone can be (i) covered completely by the query range, (ii) covered only by the query distance range, (iii) covered only by the query orientation range, and (iv) overlapping partially with distance and orientation ranges. Different retrieval methods can be applied to each case for efficient query retrieval. We present the data retrieval algorithm for a range query using a zone-structured index, based on the above four cases.

Algorithm 6.7 Range query search using the zone-structured join index structure.

Input. (i) an object identifier o_i; (ii) the lower and upper bounds of the distance range, D_{min} and D_{max}, and (iii) the lower and upper bounds of the orientation range, A_{min} and A_{max}.

Output. All spatial objects within the ranges.

Method.

1. Search for the object in the primary B+−tree.

2. Select the zones overlapping with the query range and collect data as follows:

   Case 1 (covered fully by the query range): collect all objects in the index of the given zone;

   Case 2 (covered only by the distance range): read the index records in the zone, check the angle values and collect objects satisfying the orientation condition;

   Case 3 (covered only by the orientation range): read the index records in the zone, check the distance values and collect objects satisfying the distance condition; and

   Case 4 (overlapping partially with distance and orientation ranges): read the index records in the zone and check both distance and angle attributes and collect objects satisfying the specified conditions. □

The computational complexity of solving a spatial range query using the zone SJI structure should be the same as that using the basic SJI. However, the zone structure partitions the basic SJI file into several zones, and the zones to be searched are determined before accessing the database if the distance and/or orientation range values are provided in the query. Therefore, it is expected that the zone structure improves the performance of query processing compared to the basic SJI.

An obvious advantage of the zone-structured SJI over the basic SJI is that it is not biased towards either the distance parameter or the orientation parameter. Zone-structured indices are also effective for distance-related range queries, orientation-related range queries and their combination. Moreover, if the query is to find all pairs within a certain range which fully covers the range of the zone, all objects in the zone are returned as part of the answer.

6.3.3 Hierarchical spatial-information-associated join index

As an logical enhancement following from the above discussion, a hierarchical distance-associated join indices can be augmented with an orientation measurement, which leads to a *hierarchical spatial-information-associated join index* (*hierarchical SJI* or *HSJI*) structure. The construction of hierarchical spatial-information-associated join index is similar to that of a hierarchical DJI. Distance scopes $S_1, \ldots, S_l$ and object classes $C_1, \ldots, C_l$ are defined in the same way as those in hierarchical DJI.

Definition 6.6 *Suppose that a set of* **hierarchical spatial-information-associated join indices** *has been constructed on the object classes* $\{C_1, \ldots, C_l\}$ *with different scale scope values* $S_1, S_2, \ldots, S_l$*, respectively. The join index on object class* C_k *is constructed based on the following formula:*

$$HSJI_k = \{\langle o_i, o_j, d_{i,j}, a_{i,j}\rangle \mid o_i \in C_k \wedge o_j \in C_k \wedge d_{i,j} \leq S_k\}, where\ k = 1, \ldots, l.$$

Given the orientation information stored in the index records, the new angle resulting from the join of $\langle o_i, o_j, d_{i,j}, a_{i,j}\rangle$ and $\langle o_j, o_k, d_{j,k}, a_{j,k}\rangle$ is derived by trigonometry as follows:

$$a_{i,k} = \arctan\left\{\frac{d_{i,j} \times sin(a_{i,j}) + d_{j,k} \times sin(a_{j,k})}{d_{i,j} \times cos(a_{i,j}) + d_{j,k} \times cos(a_{j,k})}\right\}$$

In contrast to the distance which increases as the path extends, the angular value may not increase monotonically as the path extends. Therefore, it is unreasonable to prune intermediate records which are out of the inquired zone with respect to the orientation constraints at each level of the hierarchy. A simple solution is to conduct the search without considering the orientation constraints and to verify the orientation for the records in the final result. A more efficient solution is to ignore the directional constraints at low levels based on the assumption that the highest level of the hierarchy in a query dominates the direction. Under this assumption, all objects satisfying the distance condition are calculated at the lower levels. At the top level,

the orientation constraints are verified to eliminate the objects whose orientation is out of the inquired range. In general, an angular buffering value $\triangle_l$ can be specified at each hierarchy level l for certain applications to allow angular deviation from the orientation range. Records beyond the range will be removed from the candidate path set. The higher the level, the more restrained the angular buffer should be set for efficient search.

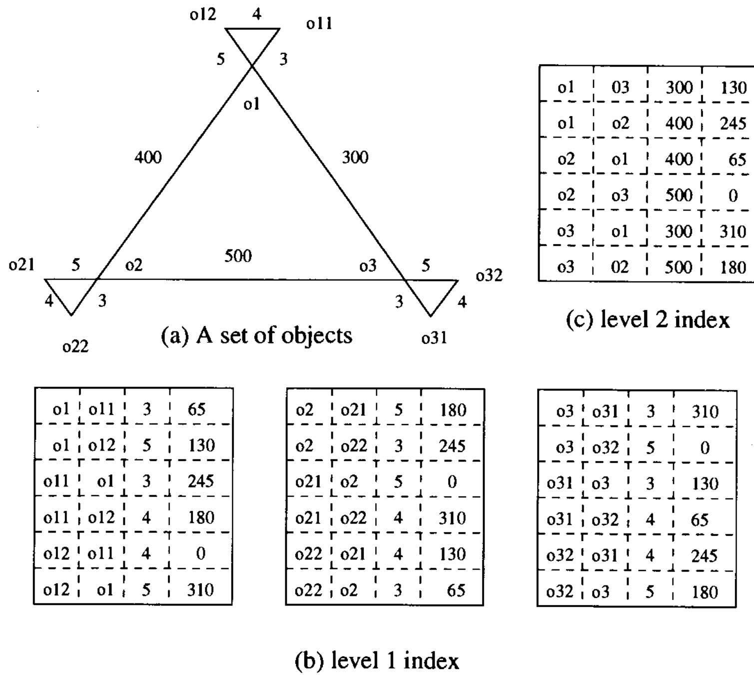
The retrieval algorithm is similar to that for the hierarchical DJI with additional operations for computing new angles and removing index records with conflicting orientations. The first step is to obtain the inquired object using the B+−tree. In the two phase hierarchy traversal, records satisfying the query conditions are stored in set T. Some temporary records are stored in set *Temp* for further ascent. At all lower levels, the orientation conditions are omitted. Only at the top level accessed by the query will the orientation condition be checked to eliminate those records which do not satisfy the constraints. At the end, the records in T are checked against the orientation condition. We present an example to illustrate the retrieval process.

Example 6.4 Retrieval using hierarchical SJI

Given the object o_1 in Figure 6.8, *"find all objects whose distance from o_1 ranges between 300 and 600 and whose orientation with respect to o_1 ranges between* 20° *and* 100°". The search sequence is as follows.

1. First, search for object o_{22} using the hierarchical SJI.
2. At the first step of the hierarchy ascent, the orientation constraints are not applied. Let T be an empty set because related object distances are all less than the lower distance bound. Let *Temp* contain $\{\langle o_{22}, o_{21}, 4, 130\rangle, \langle o_{22}, o_2, 3, 65\rangle\}$.
3. At the second step of the hierarchy ascent (the highest level involved in this query), o_3 is eliminated by the orientation condition and set T is $\{\langle o_{22}, o_1, 503, 62\rangle\}$.
4. By the hierarchy descent, we get $T = \{\langle o_{22}, o_1, 403, 62\rangle, \langle o_{22}, o_{12}, 408, 61\rangle, \langle o_{22}, o_{11}, 406, 63\rangle\}$.

The final object set returned is $\{o_1, o_{11}, o_{12}\}$. □



o1	03	300	130
o1	o2	400	245
o2	o1	400	65
o2	o3	500	0
o3	o1	300	310
o3	02	500	180

(c) level 2 index

o1	o11	3	65
o1	o12	5	130
o11	o1	3	245
o11	o12	4	180
o12	o11	4	0
o12	o1	5	310

o2	o21	5	180
o2	o22	3	245
o21	o2	5	0
o21	o22	4	310
o22	o21	4	130
o22	o2	3	65

o3	o31	3	310
o3	o32	5	0
o31	o3	3	130
o31	o32	4	65
o32	o31	4	245
o32	o3	5	180

(b) level 1 index

Figure 6.8: A two level hierarchical SJI for a set of objects

Additional information, such as the coordinates of specified objects may further speed up the search for the shortest distance between two objects, o_1 and o_2 on a road network. With the coordinate information, the orientation from one object to the other can be obtained. This orientation information can then be used to eliminate paths which do not follow the direction from obj_1 to obj_2. Thus it can significantly reduce the size of the intermediate sets T_i and T_j and speed up the processing. Generally speaking, the search space is reduced using orientation constraints.

6.4 Analysis and Simulation Results

In this section, we study the performance of the proposed spatial-information-associated join indices in a relatively large spatial database environment. The simulation results are presented, the results for three types of distance-associated join indices are analyzed, and the performance for different types of queries with respect to different spatial join indices are compared. The following notations are adopted in our analysis:

1. NDJI: for the case when no *distance-associated join indices* are used. That is, only regular join indexing structures are used and the distance information is computed at query processing time using geometric operators;
2. BDJI: for *basic distance-associated join index*;
3. RDJI: for *ring-structured distance-associated join index*;
4. HDJI: for *hierarchical distance-associated join index*;
5. BSJI: for *basic spatial-information-associated join index*, and
6. ZSJI for *zone-structured spatial-information-associated join index.*
7. HSJI for *hierarchical spatial-information-associated join index.*

6.4.1 Analytical model

An analytical model has been constructed to compare the performance of different schemes. The parameters illustrated in Table 6.1 are used in our analysis.

6.4.1.1 Storage requirement

We analyze the storage requirements for query processing using distance-associated join indices.

N	number of objects
N_i	average number of indices per page
N_b	maximum B-tree branches at any node
N_g	average number of geo-objects per page
N_{bf}	number of buffer pages
C_{io}	cost for one I/O operation
C_{dist}	cost for computing distance
C_{comp}	cost for one comparison operation

Table 6.1: Parameters for performance analysis.

1. NDJI: It requires no space for the join index.

2. BDJI: Let S be the scope distance for the distance-associated join index method and D be the a maximum object density over the area. Given an object o_i, the number of objects within the specified scope distance from o_i is the density multiplied by the area of the scope region, i.e. $D \times \pi \times S^2$. There are N objects in the database. Therefore, the basic DJI file size in pages is as follows.

$$\frac{D \times \pi \times S^2 \times N}{N_i}$$

When S is small, the size of the index file grows linearly with respect to the number of objects in the file.

3. RDJI: Given n rings, each file corresponding to an individual ring is $1/n$th of the basic DJI in average. The ring-structured DJI thus requires the same amount of space as that of basic DJI.

4. HDJI: As shown in Theorem 6.4, the space complexity S_{ind} of the hierarchical distance-associated join indices of a database with N objects is

$$\frac{N^2(K-1)}{2c^2(K^l-1)} - N \leq S_{ind} \leq \frac{2c^2N^2(k-1)}{k^l-1}$$

where the minimum and the maximum number of children of non-leaf nodes are k and K respectively, where c denotes $\lceil \frac{K}{k} \rceil$, and $l+1$ is the number of levels in the hierarchy.

Storage for spatial-information-associated join indices and that for distance-associated join indices are of the same order of magnitude.

6.4.1.2 Processing cost

Range queries will be chosen as an example for our analysis of processing cost. The curves of the cost for processing all-pairs queries will be presented later. The total processing cost is the sum of I/O operations and CPU costs.

1. NDJI: Two objects should be in the memory at the same time in order to compute the distance between them. When the number of buffer pages is smaller than the number of data pages, the total number of I/O's is greater than the number of data pages. In fact, let the number of pages of geometric objects N_p be N/N_g, and consider what happen when the *most-recently-used* page replacement strategy is used. The number of pages involved is then

$$N_{io} = \frac{N_p \times (N_p + N_{bf})}{2 \times N_{bf}}.$$

   For example, with $N_p = 100$ and $N_{bf} = 10$, the total number of I/O operations is 550. The CPU cost for computing distances between all pair objects and for checking if the distances lie in the query range is

$$CPU_{cost} = N \times (N - 1) \times (C_{dist} + C_{comp}).$$

2. BDJI: With the distance-associated join index where related objects are paired up, query processing never needs to read the same page into the buffers twice in the case of a single spatial join operation. Retrieval requires $\log_{N_b} N$ I/O operations to search the primary index and the same amount to search the secondary index. The sequential reading cost is the number of output records, k divided by N_i.

$$CPU_{cost} = (2 \times \log_{N_b} N + k/N_i) \times C_{comp}.$$

3. RDJI: If the query involves N_r rings, the ring method takes $N_r \times \log_{N_b} N$ I/O operations to search primary index and $\log_{N_b} \frac{N}{r}$ I/O operations to search the secondary index on average. Generally speaking, the ring-structured DJI takes more I/O than the basic DJI when N_r is large. The CPU cost for a range query on the ring-structured DJI is $2 \times \log(\frac{N}{N_r}) \times C_{comp}$, which is usually smaller than that required with the basic DJI.

4. HDJI: $\log_{N_b} N$ I/O operations are needed to search the leaf level primary index. The maximum number of join iterations is twice the number of hierarchy levels, denoted by l, which usually is a small integer. Because object associations are established only at consecutive levels, we can assume that the number of records in set T is bounded by a constant c_o. Hence, each join iteration takes $c_o \times \log_{N_b} N$ I/O operations. Thus, the overall I/O cost is $(2c_0 \times l + 1) \log_{N_b} N$. The cost for comparisons is therefore $((2c_0 \times l + 1) \log_{N_b} N) \times C_{comp}$.

Overall, by using distance-associated join indices, object retrieval for spatial range queries takes time logarithmic in the number of objects in the setting. The following are some comparisons for different types of spatial-information-associated join indices.

1. BSJI: The basic SJI takes $\log_{N_b} N$ I/O operations for finding the inquired object using the primary index. For queries on the preferable attribute, $\log_{N_b} N$ I/O operations are required to search the starting position of the attribute satisfying the condition. If the query is on both attributes, a check is needed for each resulting record to verify the second condition. If the query is on the nonpreferable attribute, a scan for all records related to the inquired object is needed, i.e. the cost is N/N_i.

2. ZSJI: In order to find the inquired object, a cost of $\log_{N_b} N$ I/O operations are required. Generally speaking, further accessing cost is proportional to the inquired area A. Assuming that the maximum object density is D, the number of objects searched is approximately $\frac{A \times D}{N_i}$ pages. Testing for the overlap can be done using a simple mathematical comparison. If the zone is completely covered

by the inquired area, no checking is required. If the zone is completely contained in the inquired area over one of the dimensions, checking is only needed on the other attribute. If the zone is partially overlapped with the inquired zone in both dimensions, checking for both attributes is necessary. Overall, the zone structured SJI allows queries related to distance and/or orientation to be processed efficiently.

3. HSJI: Similar to the cost analysis of HDJI, the I/O cost for HSJI retrieval is $(2c_0 \times l + 1)\log_{N_b} N$. The cost for comparisons is $((2c_0 \times l + 1)\log_{N_b} N) \times C_{comp}$.

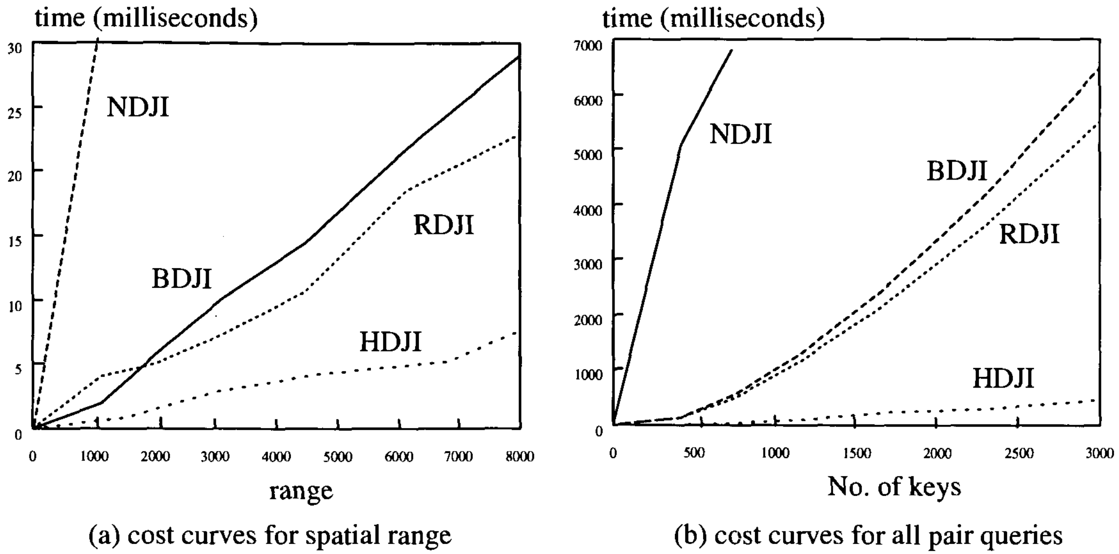
The hierarchical SJI contains more orientation information than the hierarchical DJI. The retrieval cost of hierarchical SJI is essentially the same as that for hierarchical DJI.

6.4.2 Simulation results

The simulation was performed on a SUN/4 SPARC/14.28MHz-workstation with 7 MIPS under the SunOS. The simulation program was written in Sun C compiler without optimization. A set of randomly generated points was used for simulation. The simulation was performed for two types of queries: (i) range queries, and (ii) queries for all pairs of objects with a specified distance constraint. In Figure 6.9(a), the curves represent the processing cost for different methods as the query range increases, given a fixed number (4000) of randomly generated objects. In Figure 6.9(b), the curves illustrate the relationship between the processing cost and the number of objects in the spatial database. A three-level hierarchy was built for the simulation, with the scope values specified as $S_1 = 25000$, $S_2 = 2500$ and $S_3 = 250$, respectively. Some other simulation parameter values are as follows:

1. Ring radii r_k = k × 2000, k = 1 ,..., 6;

2. $l = 3$, $N_{bf} = 40$, $N_g = 50$, $N_i = 50$, $N_b = 50$,
   $C_{io} = 4000$, $C_{dist} = 500$, $C_{comp} = 1$.



(a) cost curves for spatial range

(b) cost curves for all pair queries

Figure 6.9: Cost curves

The curves show the effectiveness of the distance-associated join indexing mechanism for reducing query processing cost in spatial queries.

In Figure 6.10, the simulation curves shows how zone-structured SJI improves efficiency for queries on distance and/or orientation. The simulation was based on the distance-preference structure. The horizontal axis is the inquired area and the vertical axis is the time to process the queries. Distance and orientation constraints were randomly generated. There were three groups of queries, (i) with constraints on both information attributes, (ii) with constraints on the preferable attribute (distance), and (iii) with constraints on the nonpreferable attribute (orientation). In the figure, ZSJI stands for zone-structured SJI, and $BSJI_b$, $BSJI_p$, $BSJI_u$, and $BSJI_a$ represent queries on basic SJI with constraints on both attributes, with constraints on the preferable attribute, with constraints on nonpreferable attribute without auxiliary index, with constraints on the nonpreferable attribute with an auxiliary index respectively.

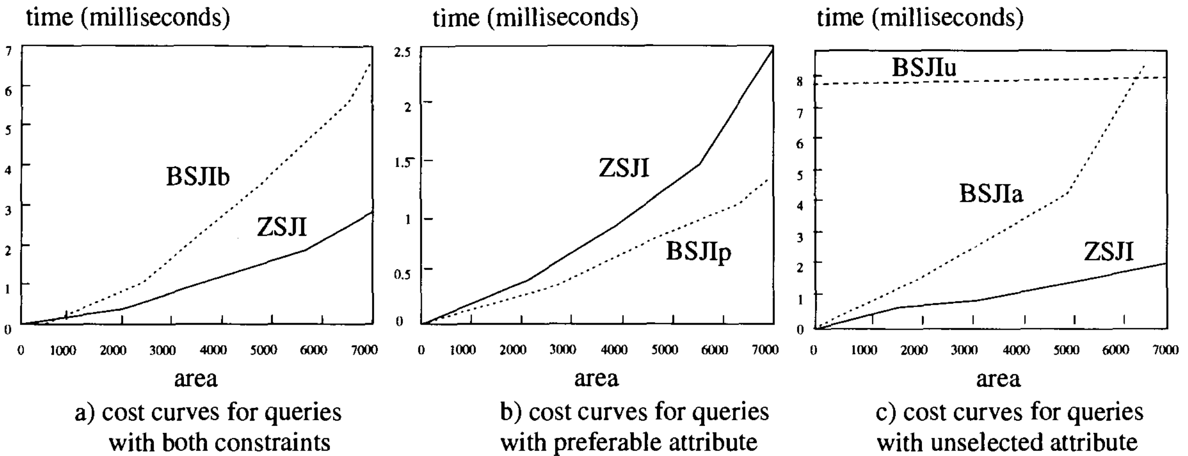


Figure 6.10: Cost curves of spatial-information-associated.

From the curves in Figure 6.10, it is obvious that the zone-structured SJI performs reasonably well in all cases whereas the performance of the basic SJI depends on the constraints specified in the query.

For the same set of objects, a performance comparison was conducted for K-D trees, range-trees and distance-associated join indices for a circular range search. Given a distance range, a circumscribing box was formed to perform a rectangular search. Figure 6.11 shows the comparison curves.

6.4.3 Analysis of the simulation results

As shown by the simulation results, the distance-associated join index improves the range query performance significantly. The three types of distance-associated join indices have advantages in different situations.

- The basic DJI is simple and results reasonably good performance when the database is small. However, when the size of the data relation grows, the performance of the basic DJI deteriorates.

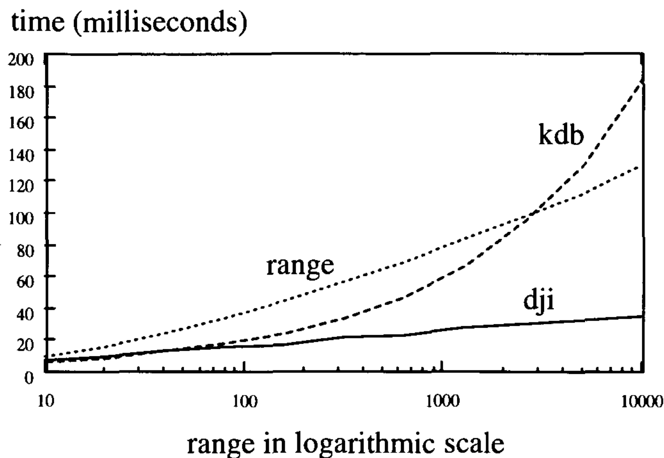


Figure 6.11: Comparison with other structures

- The ring-structured DJI decomposes one basic DJI file into several ring index files. For queries relevant to one particular ring, the ring structure reduces the search space and therefore enhances performance. For range queries, the ring-structured DJI reduces the search cost when the range is confined to a small number of rings. When the range expands, that is when it covers many ring structures to be referenced, the ring-structured DJI performs slightly worse than the basic DJI structure. For all pair queries (queries to find all pairs of spatial objects satisfying a distance constraint), the ring structure performs slightly better than the basic DJI because fewer comparison operations are required.

- Since the hierarchical indexing structure substantially reduces the number of index records which must be directly associated in the database, it substantially reduces the storage space and hence the access cost in a relatively large database. Although spatial objects need to refer to their higher level spatial objects for some spatial queries, the cost of accessing several higher level reference points only increases the access cost linearly. Since such an organization may substantially reduce the storage and access cost, the hierarchical DJI offers the best performance in most cases.

The curves presented in Figure 6.10 show the performance difference between the basic SJI and the zone-structured SJI. Figure 6.10(a) indicates the query processing time for queries with constraints on both distance and direction. The zone structure accesses only the zones overlapping with the inquired range; whereas the basic structure selects all records satisfying the distance constraint and then eliminates those that fail the orientation test. Therefore, the zone-structured index is more efficient than the basic one. Figure 6.10(b) indicates the query processing time for queries with constraints on one preferable information attribute, such as distance in our experiment. The ZSJI still performs reasonably well. The basic structure is a little more efficient than the zone-structure because zone mapping takes time, because partially overlapped zones may contain records which are out of the inquired range. Figure 6.10(c) contains the curves with the sharpest contract, where queries are only on the nonpreferable attribute. There is no obvious negative effect on the performance of the zone-structured SJI. When the query constraints are on the nonpreferable attribute, the basic structure performs poorly. Without auxiliary index, all records related to the inquired object have to be read and checked to see whether they satisfy the conditions; curve $BSJI_u$ shows the result. With an auxiliary index, only the records satisfying the condition are accessed. However, since these records are scattered all over the index file, retrieval cost is still quite high as shown by the curve labeled $BSJI_a$. Hence, overall, the zone-structured SJI performs much better than the basic structure.

In comparison with K-D-trees and range trees, distance-associated join indices performs better than both methods. For N objects, retrieval using a K-D tree structure takes $O(\sqrt{N})$, retrieval using range-trees takes $O(\log^2 N)$, while retrieval using distance-associated join indices takes only $O(\log N)$. As the search range increases, range trees perform better than a K-D-tree because of its range structure.

6.5 Chapter Summary and Discussions

In this chapter, a general and flexible spatial indexing structure, the spatial-information-associated join index, has been developed. Two important pieces of spatial information between pairs of spatial objects, *distance* and *orientation*, are precomputed at join index construction time and stored for efficient query processing. Information-associated join index structures include but are not limited to distance and/or orientation information. For example, if intersection of regions are frequently required, the associated information may be the intersection. Since costly geometric computations of spatial relationships are performed before query processing time, queries using this spatial information can be processed fairly efficiently according to our analysis.

To facilitate the step-by-step analysis for different applications, different distance-associated join index structures have been investigated and compared. Three distance-associated join index structures have been proposed for the optimization of spatial range queries and other spatial queries. By associating distance information with the join index, part of the cost involved in processing the query can be reduced or eliminated by performing these computations at index construction time.

Each kind of spatial-information-associated structure has its own application domain. The basic DJI is concise and efficient in simple and small environment. The ring-structured DJI performs well when the query references only a small spatial range in a moderate size database. By adjusting the ring radii, a reduction of the size of the file to be processed and an improvement of the I/O cost can be achieved. The hierarchical DJI performs best among the three in a complex and large spatial database. For a city map database, the hierarchical DJI can help finding the shortest path between objects at a reasonable processing cost. All three join index mechanisms are simple, flexible, and easy to create and maintain. Our primitive simulation-based performance study demonstrates the high promises of this approach.

As an extension to the distance-associated join index, a spatial-information-associated join index has been proposed which utilize both the distance and orientation information of related object pairs to facilitate spatial query processing. The zone-structured SJI partitions the join records into nonoverlapping zones each of which is specified by mathematical boundaries so that overlapping tests with the inquired range are simple. The zone-structured SJI accesses only indices in the zones which overlap with the inquired zone, and is not biased to either spatial attribute. The experimental results show that this mechanism is efficient.

Not only spatial join information such as distance and orientation, but also other pieces of information, such as driving speed, can be associated with the record when necessary. For example, in the query *"find a driving route which reaches B from A in the shortest time"*, the driving speed could be a useful asset. The spatial-information-associated join index illustrates the effectiveness of associating important information with join indices. The information-associated join index can be used to precompute and store any piece of computationally intensive information (not necessarily spatial information) related to two objects to facilitate the efficient implementation of join operations related to such piece of information.

However, this idea should not be pushed to an extreme. First, it is important to clarify the role of spatial join indices in spatial databases. As it has been demonstrated in [140], the role of join indices in relational database systems is not to replace the common relational indexing structures, such as B+-trees, but to provide additional indexing support to speed up certain relational operations, such as joins. In this respect, the role of spatial join indices in spatial databases is analogous to that of join indices in relational databases. The goal of information-associated spatial join indexing structure is not to replace the commonly used spatial indexing structures but to speed up certain spatial join operations. It is not our intention to substitute the traditional spatial indexing structures by join indices. However, our study shows that it may provide good support for certain spatial operations, which could be used relatively frequently in spatial database applications. Also, in a situation where the locations of objects change from time to time, it is expensive to recompute these

indices and to keep all associations consistent and up-to-date. Finally, there are many geometric operations which may involve combinations of different objects. Since there are a large number of possible combinations, it is unrealistic to explore and store all such combinations. For example, the geometric constructor *union* may involve different combinations and create new geometric objects. It is practical to determine all such combinations and to precompute and store them as join indices before query processing.

This chapter proposed an information-associated spatial join indexing structure for spatial query optimization. The complexity analysis and preliminary simulation-based performance study have demonstrated the good performance of this interesting data structure. Further studies, development and experiments should be performed on the implementation of spatial-information-associated join indices and on their application to spatial query optimization in large spatial databases.

Chapter 7

Conclusion

We summarize our research work in this chapter. Discussion on future directions in a general intelligent spatial database follows the summary of the research.

7.1 Summary

In this thesis, a deductive and object-oriented paradigm for spatial database design has been studied. A deductive and object-oriented spatial database system (DOOSDB) provides an effective modeling facility for spatial data. Object hierarchies are used to provide property inheritance and type consistency checking. Spatial relationships can be defined using high-level deduction rules. Spatial queries can be posed using either an SQL-like syntax or a Prolog-like syntax. The query interface with dual syntaxes provides users with both first-order query power and ad-hoc query user friendliness. In this system, spatial and nonspatial data can be treated uniformly in a query. The system can be extended using user-defined object classes, rules and methods.

To achieve processing efficiency for high-level queries in a large spatial database, optimization is essential. We investigated set-oriented query processing and optimization in deductive and object-oriented spatial (DOOS) databases and proposed

an integrated paradigm and a set of useful techniques for DOOS query optimization.

The integrated paradigm of spatial query optimization in DOOS databases provides us with the following advantages:

1. It promotes a structured design and an integrated, high-level view of spatial databases, which leads to high-level query interfaces, uniform handling of complex spatial object structures using object-oriented storage management, and flexibility in the specification and use of spatial computation routines (methods) together with deduction rules.

2. It facilitates the exploration of various aspects of query optimization to achieve high efficiency, including the compilation of deductive queries, the simplification of query expressions by relational and spatial optimization rules, the access plan generation by analysis of the query processing costs of different candidate plans, and set-oriented processing and optimization of spatial computation routines (methods).

3. Set-oriented spatial method computation overcomes the weakness present in tuple-oriented spatial computations, reduces redundant spatial computation and provides an interesting solution to the impedance mismatch problem.

4. The spatial-information-associated join index, has been developed for spatial join operations. Important pieces of spatial information involving spatial object pairs, such as *distance* and *orientation*, are precomputed when the spatial join index is constructed, and stored for efficient query processing. Since costly geometric computations of spatial relationships are performed before the query is processed, queries that require such spatial information can be processed fairly efficiently according to our analysis.

The proposed approach for spatial query processing supports a high-level user interface and promotes compilation-based, set-oriented, efficient query processing in deductive and object-oriented spatial databases. Implementation of such a DOOS system is in progress.

7.2 Discussion

An intelligent spatial system should not only perform data retrieval but also be able to discover interesting knowledge from spatial data, to perform spatial deduction and to help decision making [38, 58, 95]. We will discuss one of the interesting research directions in this area, i.e. learning in a large spatial database.

Another very important aspect of spatial databases is its temporal factor. Historical database information is often used in GIS and engineering database systems [87, 8, 147]. Problems and research directions in this area will also be addressed.

7.2.1 Knowledge discovery in large spatial databases

Spatial reasoning using data and knowledge stored in large spatial databases is a crucial task in the development of geographical information systems, medical imaging systems and robotics systems. Because of the huge amount (usually tera-bytes) of spatial data obtained from satellites, video cameras, medical equipment, etc., it is costly and often unrealistic for users to examine the spatial data in detail to extract interesting knowledge or general patterns from spatial databases.

Knowledge discovery in spatial databases is the extraction of interesting spatial patterns and features, of general relationships between spatial and nonspatial data and of other general data characteristics not explicitly stored in the database. This discovery may play an important role in understanding spatial data, capturing intrinsic relationships between spatial and nonspatial data, presenting data regularity in a concise manner and reorganizing spatial databases to accommodate data semantics and achieve high performance.

In a preliminary study [95], the attribute-oriented induction technique [58] was extended to knowledge discovery in spatial databases. Two kinds of concept hierarchies, thematic concept hierarchies and spatial hierarchies, were constructed for the learning process. Induction was performed by traversing these hierarchies and summarizing

general relationships between spatial and nonspatial attributes at a high concept level. Two algorithms were developed based on the priority set for performing generalization on the nonspatial concept hierarchy or the spatial hierarchy.

Nonspatial-data-dominated generalization first generalizes non-spatial attributes to a specified high level and then performs spatial consolidation, which consists of the following steps:

1. Collect related data;
2. Perform attribute-oriented induction on the collected nonspatial data by (i) concept-hierarchy-ascending, (ii) attribute-removal, (iii) merge of identical tuples until either the number of tuples is within the generalization threshold or every attribute has been generalized to the desired concept level; and
3. Perform spatial generalization, i.e. merge neighboring areas with the same high-level attribute using the relationship *is_adjacent_to*.

Spatial-data-dominated generalization first climbs up the spatial hierarchy and for each resulting spatial object, generalizes non-spatial-attributes, in the following major steps:

1. Collect task-relevant data,
2. Generalize the spatial database by clustering spatial object according to their regions and merging them until the desired concept level is reached or until the number of generalized spatial objects is smaller than a threshold, and
3. For each region, perform generalization on non-spatial objects until a minimal concept set subsumes all of the concepts in the subregions.

This method can discover interesting relationships between spatial and nonspatial data and can be applied to the analysis of correlations between different spatial features based on different thematic maps. Our preliminary study shows that knowledge

discovery can be performed efficiently in spatial databases by extending the techniques used for knowledge discovery in relational databases.

Beside the two primitive generalization techniques, i.e. *nonspatial-data-dominated generalization* and *spatial-data-dominated generalization*, presented in [95], more sophisticated algorithms are required for complex spatial environments, which may require extension of these techniques in many ways. We discuss several possible extensions of the algorithms discussed above.

1. Interleaved generalization between spatial and nonspatial data.

   The nonspatial-attribute-oriented algorithm generalizes nonspatial data before generalizing spatial data, whereas the spatial-attribute-oriented algorithm proceeds in reverse order. In some cases, one may wish to interleave generalization between spatial and nonspatial data to achieve satisfactory results with reasonable performance. A spatial-data-dominated algorithm could be costly to evaluate. It is often preferable to perform non-spatial (relational) generalizations to a certain level and then perform a high-level spatial merge/join or approximation. Further generalization may depend on the number of distinct spatial objects or appropriate concept levels. The concrete algorithm integrating the above two algorithms to achieve interleaved generalization can be developed.

2. Generalization on multiple thematic maps.

   The generalization algorithms in [95] involve only one thematic map. In some applications, a learning task may require generalization on more than one thematic map. Similar spatial generalization techniques, such as spatial merge and approximation, can be applied on the overlay of the two maps to find the regions in each class. Generalization may also derive relationships between nonspatial attributes. For example, a correlation between temperature and precipitation can be discovered by learning.

3. Learning in a dynamic/temporal spatial database.

One application of learning in spatial databases is to analyze satellite data, in which temporal factors play an important role. The learning process may be performed on a sequence of data maps. Differentiation of spatial features at different times may enable the system to detect geographical events, such as a quake on the Moon, based on images from the Moon.

4. Probability in learning.

   Impure data occurs in spatial and real life applications, e.g. 80% of the trees in a given region are pine trees, while 20% of trees in that forest may be of different types. Probabilistic learning and fuzzy logic are useful tools for applications in spatial knowledge discovery.

Learning in spatial databases is a challenging and promising research area in spatial databases. As an emerging topic for integration of spatial databases and machine learning technologies, knowledge discovery in spatial databases will have applications in spatial knowledge discovery, spatial reasoning, spatial query optimization, the construction of multiple resolution spatial data models, etc. More investigations should be performed in this direction, especially regarding its integration with statistical methods, the development of customized spatial generalization operators, as well as, additional studies should be done on knowledge discovery in spatial databases under different assumptions.

7.2.2 Spatiotemporal databases

Spatiotemporal data captures the movement and changes of objects over time in a dynamic database system. Medical image record analysis, physical experiment analysis, urban development monitoring and other geographic problems involve both spatial and temporal data. Historic spatial databases are also important for decision making. A spatiotemporal database can be constructed by adding temporality to spatial databases [8] or by merging spatial data types with temporal components in a historical database.

Some major issues related to temporal databases include data representation models, huge amount of data, temporal integrity constraints, granularity, different versions of objects and spatiotemporal indexing mechanism. We discuss some of the problems.

Nested relations have been proposed for modeling temporal databases [42, 120]. Nested relations used for modeling spatial data in DOOSDB can be extended for modeling spatiotemporal data. Langran proposed a framework for temporal geographic information [87]. Two data structures recommended for storing spatiotemporal data are "base maps with overlays" and "space-time composition". The former uses an initial data map and records the change of the map over the time, whereas the later essentially decomposes also the image database into subsections so that each component can be expressed in terms of a time-property list.

Spatiotemporal indexing and object clustering add a new dimension to those in spatial databases and therefore represent a more challenging task. An extension of spatial index R-tree for spatiotemporal purposes has been presented using [147], in which an RT-tree incorporates temporary information in spatial objects and index nodes. It represents an elegant merge of multiple R-trees with different time stamps, thus saving storage space and improving performance.

The challenge of spatiotemporal databases comes from the combination of the large volumes of data in temporal database with the difficulty of optimizing geometric functions in spatial databases and with the spatiotemporal index. A lot of research work still needs to be performed in this direction.

In summary, spatiotemporal databases and knowledge discovery in spatial databases are important research topics in spatial database research. These two topics will be the focusing point of our future research.

Appendix A

BNF of the Query Language DOOSQL

The following meta-symbols are used for language definition:

- ::= defines non-terminal symbol
- ⟨ ⟩ for non-terminal symbol
- [ ] for an optional component of language which may appear at most once.
- { } for an optional component of language which may appear any number of times.

DOOSQL ::= ⟨*definitions*⟩ | ⟨*queries*⟩

⟨*definitions*⟩ ::= ⟨*schema*⟩ | ⟨*procedure*⟩ | ⟨*rule*⟩

⟨*schema*⟩ ::= **schema** ⟨*schema_body*⟩

⟨*schema_body*⟩ ::= ⟨*name*⟩ (⟨*attri_def*⟩ {, ⟨*attri_def*⟩ })

⟨*attri_def*⟩ ::= ⟨*atom_attri_def*⟩ | **setof** ⟨*subschema*⟩
| **sequenceof** ⟨*subschema*⟩

⟨*subschema*⟩ ::= ⟨*type*⟩ | ⟨*schma_body*⟩

⟨*atom_attri_def*⟩ ::= ⟨*name*⟩ : ⟨*type*⟩

⟨*procedure*⟩ ::= **procedure** ⟨*name*⟩ (⟨*para_def*⟩ {, ⟨*para_def*⟩ })

⟨*para_def*⟩ ::= ⟨*Name*⟩ : ⟨*type*⟩ ⟨*mode*⟩

⟨rule ⟩ ::= ⟨predicate ⟩ : – ⟨predicate ⟩{, ⟨predicate ⟩}.

⟨predicate ⟩ ::= ⟨name ⟩(⟨Name ⟩{,⟨Name ⟩})

⟨*queries*⟩ ::= ⟨*sql_query*⟩ | ⟨*logic_query*⟩

⟨*sql_query*⟩ ::= **select** ⟨*result*⟩ {, ⟨*result*⟩ }
from ⟨*name*⟩ {, ⟨*name*⟩ }
where ⟨*predicate_expression*⟩

⟨*result*⟩ ::= ⟨*attribute*⟩ | ⟨*func_name*⟩ (⟨*attribute*⟩ {, ⟨*attribute*⟩ })

⟨*logic_query*⟩ ::= ? – ⟨*predicate*⟩ {, ⟨*predicate*⟩ }.

⟨*predicate*⟩ ::= ⟨*name*⟩ (⟨*parameter*⟩ {, ⟨*paremeter*⟩ })

⟨*parameter*⟩ ::= ⟨*constant*⟩ | ⟨*Variable*⟩

⟨*predicate_expression*⟩ ::= ⟨*pred_term*⟩ { ⟨*logic_link*⟩ ⟨*pred_term*⟩ }

⟨*logic_link*⟩ ::= **and** | **or**

⟨*pred_term*⟩ ::= [**not**] ⟨*predicate*⟩
| [**not**] (⟨*math_expression*⟩ ⟨*comp*⟩ ⟨*math_expression*⟩)
| [**not**] ⟨*geo_obj*⟩ ⟨*geo_predicate*⟩ ⟨*geo_obj*⟩

$\langle geo_predicate \rangle$ ::= *is_inside* | *is_adjacent_to* | *geo_intersect* | $\cdots$

$\langle func_name \rangle$ ::= $\langle geo_func \rangle$ | $\langle geo_val_func \rangle$ | $\langle aggreg_func \rangle$

$\langle geo_val_func \rangle$ ::= *area* | *length* | *distance* | $\cdots$

$\langle geo_func \rangle$ ::= *geo_union* | *geo_intersection* | $\cdots$

$\langle aggreg_func \rangle$ ::= *maximum* | *minimum* | *count* | *sum* | *average*

$\langle geo_obj \rangle$ ::= $\langle attribute \rangle$ | *geo_func* ($\langle geo_obj \rangle$ {, $\langle geo_obj \rangle$ })

$\langle comp \rangle$ ::= $= \mid \neq \mid < \mid > \mid \leq \mid \geq$

$\langle type \rangle$ ::= [$\langle group \rangle$] $\langle atom_type \rangle$

$\langle atom_type \rangle$ ::= *INT* | *REAL* | *STRING* | *BOOLEAN* |
POINT | *LINE* | *POLYGON* | *GEO*

$\langle group \rangle$ ::= **setof** | **sequenceof**

$\langle mode \rangle$ ::= **in** | **out** | **all**

$\langle name \rangle$ ::= $\langle char_num_string \rangle$

$\langle Variable \rangle$ ::= $\langle Name \rangle${. $\langle name \rangle$ }

$\langle Name \rangle$ ::= $\langle Char_num_string \rangle$

$\langle attribute \rangle$ ::= $\langle name \rangle$ {. $\langle name \rangle$ }

Bibliography

[1] A.I. Abdelmoty, M.H. Williams, and N.W. Paton. Deduction and deductive databases for geographic data handling. In *Advances in Spatial Databases (Proc. 3rd Symp. SSD'93)*, pages 443–464, Singapore, June 1993.

[2] S. Abiteboul. Towards a deductive object oriented language. In *Proc. 1st Int'l Conf. Deductive and Object-Oriented Databases (DOOD'89)*, pages 453–472, Kyoto, Japan, December 1989.

[3] S. Abiteboul and S. Grumbach. *COL: A Logic-Based Language for Complex Objects.* ACM Press Frontier Series, Addison-Wesley, New York, 1990.

[4] R. Agrawal and H. Jagadish. Direct algorithms for computing the transitive closure of database relations. In *Proc. 13th Int'l Conf. Very Large Data Bases*, pages 255–266, Brighton, England, Sept. 1987.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, 1983.

[6] T. Andrews and C. Harris. Combining language and database advances in an object-oriented development environment. In *Proc. 2nd Conf. Object-Oriented Programming Systems, Languages and Applications*, 1987.

[7] W. G. Aref and H. Samet. Optimization strategies for spatial query processing. In *Proc. 17th Int'l Conf. Very Large Data Bases*, pages 81–90, Barcelona, Spain, Sept. 1991.

[8] M. Armstrong. Temporality in spatial database. In *Proc. GIS/LIS'88*, pages 880–889, San Antonio, Dec. 1988.

[9] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object oriented database systems manifesto. In *Proc. 1st Int'l Conf. Deductive and Object-Oriented Databases (DOOD'89)*, pages 40–57, Kyoto, Japan, Dec. 1989.

[10] F. Bancilhon, C. Delobel, and P. Kanellakis. *Building an Object-oriented Database System: The Story of O2.* Morgan Kaufmann Publishers, 1992.

[11] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proc. 5th ACM Symp. Principles of Database Systems*, pages 1–15, Cambridge, MA, March 1986.

[12] J. Banerjee, W. Kim, H. J. Kim, and H. F. Korth. Semantics and implementation of schema evolution in object-oriented databases. In *Proc. 1987 ACM-SIGMOD Int'l Conf. Management of Data*, pages 143–159, San Francisco, 1987.

[13] L. Becker and R. Güting. Rule-based optimization and query processing in an extensible geometric database system. *ACM Trans. Database Systems*, 17:247–303, June 1992.

[14] C. Beeri. Formal models of object oriented databases. In *Proc. 1st Int'l Conf. Deductive and Object-Oriented Databases (DOOD'89)*, pages 405–429, Kyoto, Japan, Dec. 1989.

[15] C. Beeri and R. Ramakrishnan. On the power of magic. In *Proc. 6th ACM Symp. Principles of Database Systems*, pages 269–283, San Diego, March 1987.

[16] J. L. Bentley. Decomposable searching problems. *Info. Proc. Lett.*, 8:244–251, 1979.

[17] W. Bohm. A survey of curve and surface methods in CAGD. *Computer Aided Geometry Design*, 1:1–60, 1986.

[18] O. P. Buneman and M. P. Atkinson. Inheritance and persistence in database programming languages. In *Proc. 1986 ACM-SIGMOD Int'l Conf. Management of Data*, pages 4–15, Washington, DC, 1986.

[19] P.A. Burrough. Principles of geographical information system for land resources assessment. *Monographs on Soil and Resources Survey*, 12, 1986.

[20] M. Carey, D. DeWitt, D. Frank, G. Graefe, M. Muralikrishna, J. Richardson, and E. Shekita. The architecture of the EXODUS extensible DBMS. In *Proc. Int'l Workshop of Object-Oriented Database System*, pages 52–64, Pacific Grove, Sept. 1986.

[21] U. S. Chakravarthy, J. Grant, and J. Minker. Foundations of semantic query optimization for deductive databases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 243–274. Morgan Kaufmann, 1988.

[22] C.Y. Chan, B. C. Ooi, and H. Lu. Extensible buffer management of indexes. In *Proc. 18th Very Large Data Base*, pages 444–454, Vancouver, Aug. 1992.

[23] N. S. Chang and K. S. Fu. Picture query languages for pictorial database systems. *Computer*, 25:23–33, 1981.

[24] S. K. Chang and S. H. Liu. Picture indexing and abstraction techniques for picture databases. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 6:475–483, 1984.

[25] B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. In *1st ACM Symp. Computational Geometry*, pages 135–146, Baltimore, MD, June 1985.

[26] W. Chen, M. Kifer, and D.S. Warren. HiLog as a platform for a database language (or why predicate calculus is not enough). In *Proc. 2nd Int'l Workshop Database Programming Languages*, pages 315–329, Gleneden Beach, OR, June 1989.

[27] J.-B. Cheng and A. R. Hurson. Effective clustering of complex objects in object-oriented databases. In *Proc. 1991 ACM-SIGMOD Int'l Conf. Management of Data*, pages 22–32, 1991.

[28] D. Chimenti, R. Gamboa, R. Krishnamurthy, S. Naqvi, S. Tsur, and C. Zaniolo. The LDL system prototype. *IEEE Trans. Knowledge and Data Engineering*, 2:76–90, March 1990.

[29] A. Choi and W. S. Luk. Using an object-oriented database system to construct a spatial database kernel for GIS applications. *Computer System Science and Engineering*, 7:100–121, April 1992.

[30] E. F. Codd. Extending the relational database model to capture more meaning. *ACM Trans. Database Systems*, 4:397–434, 1979.

[31] W. J. Coffey. *Geography, Towards a General Spatial System Approach.* Methuen and Co. Ltd, London and New York, 1981.

[32] C. J. Date. *An Introduction to Database Systems, 5th edition.* Addison-Wesley, 1990.

[33] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. In *1st ACM Symp. Computational Geometry*, pages 251–262, Baltimore, MD, June 1985.

[34] M. J. Egenhofer. *Spatial Query Languages.* UMI Research Press, Ann Arbor, MI, 1989.

[35] M. J. Egenhofer. Reasoning about binary topological relations. In *Advances in Spatial Databases (Proc. 2nd Symp. SSD'91)*, pages 143–160, Zurich, Switzerland, Aug. 1991.

[36] M. J. Egenhofer and J. Sharma. Topological consistency. In *Proc. 5th Int'l Symp. Spatial Data Handling*, pages 335-343, Charleston, S.C., Aug. 1992.

[37] C. Faloutsos, T. Sellis, and N. Roussopoulos. Analysis of object-oriented spatial access methods. In *Proc. 1987 ACM-SIGMOD Int'l Conf. Management of Data*, pages 426–439, San Francisco, 1987.

[38] D. Fisher and P. Langley. Approaches to conceptual clustering. In *Proc. 9th Int'l Joint Conf. AI*, pages 691–697, Los Angeles, Aug. 1985.

[39] J. D. Foley. *Computer Graphics : Principles and Practice.* Addison-Wesley, 1990.

[40] A. U. Frank. Requirements for database systems suitable to manage large spatial databases. In *Proc. 1st Int'l Symp. Spatial Data Handling*, pages 38–59, Zurich, Switzerland, 1984.

[41] J. Freytag. A rule-based view of query optimization. In *Proc. 1987 ACM-SIGMOD Int'l Conf. Management of Data*, pages 173–180, San Francisco, May 1987.

[42] S. K. Gadia. A homogeneous relational model and query language for temporal databases. *ACM Trans. Database Systems*, 13:418–448, Dec 1988.

[43] H. Gallaire, J. Minker, and J. Nicolas. Logic and databases: A deductive approach. *ACM Comput. Surv.*, 16:153–185, 1984.

[44] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Leaning.* Addison-Wesley, 1989.

[45] R. Gonzalez and P. Wintz. *Digital image processing.* Addison-Wesley Pub. Co., 1977.

[46] A. M. Goodman, R. M. Haralick, and L. Shapiro. Knowledge-based computer vision – integrated programming language and data management system design. *Computer*, 22:43–58, Dec. 1989.

[47] G. Graefe and D. DeWitt. The EXODUS optimizer generator. In *Proc. 1987 ACM-SIGMOD Int'l Conf. Management of Data*, pages 160–172, San Francisco, 1987.

[48] D. Greene. An implementation and performance analysis of spatial data access methods. In *Proc. 5th Int'l Conf. Data Engineering*, pages 606–615, Los Angeles, Feb. 1989.

[49] S. Grumbach. Integration of functions defined with rewriting rules in datalog. In *Proc. 1st Int'l Conf. Deductive and Object-Oriented Databases (DOOD'89)*, pages 349–368, Kyoto, Japan, December 1989.

[50] O. Günther. The design of Cell Tree: An object-oriented structure for geometric databases. In *Proc. 5th Int'l Conf. Data Engineering*, pages 598–605, Los Angeles, Feb. 1989.

[51] O. Günther. Efficient computation of spatial joins. In *Proc. 9th Int'l Conf. Data Engineering*, pages 50–60, Vienna, Austria, April 1993.

[52] O. Günther and H. Noltemier. Spatial database indices for large extended objects. In *Proc. 7th Int'l Conf. Data Engineering*, pages 520–526, Kobe, Japan, 1991.

[53] R. H. Güting. *Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems.* Springer-Verlag, Venice, Italy, March 1988.

[54] R. H. Güting. Gral: An extensible relational database system for geometric applications. In *Proc. 15th Int'l Conf. Very Large Data Bases*, pages 33–44, Amesterdam, Sweden, Aug. 1989.

[55] A. Guttman. R-Tree: A dynamic index structure for spatial searching. In *Proc. 1984 ACM-SIGMOD Int'l Conf. Management of Data*, pages 47–57, Boston, June 1984.

[56] L. A. Haas, W. Chang, G. M. Lohman, J. McPherson, P. F. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M. J. Carey, and E. Shekita. Starburst Mid-Flight: As the dust clears. *IEEE Trans. Knowledge and Data Engineering*, 2:143–160, 1990.

[57] R. W. Haddad and J. F. Naughton. Counting methods for cyclic relations. In *Proc. 7th ACM Symp. Principles of Database Systems*, pages 333–340, Austin, March 1988.

[58] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: An attribute-oriented approach. In *Proc. 18th Int'l Conf. Very Large Data Bases*, pages 547–559, Vacouver, Canada, Aug. 1992.

[59] J. Han and L. J. Henschen. Handling redundancy in the processing of recursive database queries. In *Proc. 1987 ACM-SIGMOD Int'l Conf. Management of Data*, pages 73–81, San Francisco, 1987.

[60] J. Han, L. J. Henschen, and N. Zhuang. Derivation of magic sets by compilation. In *Proc. 1989 Int'l Conf. Software Engineering and Knowledge Engineering*, pages 164–171, Chicago, June 1989.

[61] J. Han and Z. Li. Deductive-ER: A deductive entity-relationship data model and its data language. *Information and Software Technology*, 34:192–104, 1992.

[62] J. Han and W. Lu. Asynchronous chain recursions. *IEEE Trans. Knowledge and Data Engineering*, 1:185–195, 1989.

[63] J. Han and Q. Wang. Evaluation of functional linear recursions: A compilation approach. *Information Systems*, 16:463–469, 1991.

[64] M. Hardwick. Why ROSE is fast: Five optimizations in design and experimental database system for CAD/CAM applications. In *Proc. 1987 ACM-SIGMOD Int'l Conf. Management of Data*, pages 292–298, San Francisco, 1987.

[65] A. Henrich, H. Six, and P. Widmayer. The LSD tree: Spatial access to multidimensional point and non-point objects. In *Proc. 15th Int'l Conf. Very Large Data Bases*, pages 45–53, Amsterdam, Aug. 1989.

[66] A. Henrich, H.-W. Six, and P. Widmayer. The R-file: An efficient access structure for proximity queries. In *Proc. 7th Int'l Conf. Data Engineering*, pages 372–379, Los Angeles, 1990.

[67] A. R. Hurson, S. H. Pakzad, and J.-B. Cheng. Object-oriented database management systems: Evolution and performance issues. *Computer*, 37:48–60, Feb. 1993.

[68] T. Ibaraki and T. Kameda. On the optimal nested order for computing n-relational joins. *ACM Trans. Database Systems*, 9:483–502, Sept. 1984.

[69] Y. E. Ioannidis and Y. Kang. Randomized algorithms for optimizing large join queries. In *Proc. 1990 ACM-SIGMOD Int'l Conf. Management of Data*, pages 312–321, Atlantic City, NJ, May 1990.

[70] Y. E. Ioannidis, R. T. Ng, K. Shim, and T. K. Sellis. Parametric query optimization. In *Proc. 18th Int'l Conf. Very Large Data Bases*, pages 103–114, Vancouver Canada, Aug. 1992.

[71] H. V. Jagadish and L. O'Gorman. An object model for image recognition. *Computer*, 22:33–42, Dec. 1989.

[72] M. Jarke and J. Koch. Query optimization in database systems. *ACM Comput. Surv.*, 16:111–152, 1984.

[73] B. Jiang. A suitable algorithm for computing partial transitive closures. In *Proc. 6th Int'l Conf. Data Engineering*, pages 264–271, Los Angeles, Feb. 1990.

[74] R. Kasturi, R. Fernandez, M. L. Amlani, and W. Feng. Map data processing in geographic information systems. *Computer*, 22:10–21, Dec. 1989.

[75] W. Kent. Limitations of record-based information models. *ACM Trans. Database Systems*, 4:107–131, 1979.

[76] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented database. In *Proc. 1992 ACM-SIGMOD Int'l Conf. Management of Data*, pages 393–412, San Diego, June 1992.

[77] M. Kifer and G. Lausen. F-Logic: A higher order language for reasoning about objects, inheritance, and scheme. In *Proc. 1989 ACM-SIGMOD Int'l Conf. Management of Data*, pages 134–146, Portland, OR, June 1989.

[78] W. Kim. On optimization of a SQL-like nested query. *ACM Trans. Database Systems*, 7:443–469, Sept. 1982.

[79] W. Kim. *Introduction to object-oriented databases.* MIT Press, Cambridge, MA, 1990.

[80] W. Kim. Object-oriented databases: Definition and research directions. *IEEE Trans. Knowledge and Data Engineering*, 2:327–341, 1990.

[81] W. Kim and F. H. Lochovsky. *Object-Oriented Languages, Applications, and Databases.* Addison-Wesley, 1989.

[82] W. Kim, J.-M. Nicolas, and S. Nishio, editors. *Deductive and Object-Oriented Databases.* North-Holland, 1990.

[83] D. Knuth. *The Art of Computer Programming.* Addison-Wesley Publishing Company, 1973.

[84] R.A. Kowalski. Directions for logic programming. In *Proc. 2nd Symp. Logic Programming*, pages 2–9, Boston, July 1985.

[85] R. Krishnamurthy, H. Boral, and C. Zaniolo. Optimization of nonrecursive queries. In *Proc. 12th Int'l Conf. Very Large Data Bases*, pages 128–137, Kyoto, Japan, Aug. 1986.

[86] R. Krishnamurthy and C. Zaniolo. Safety and optimization of Horn clause queries. In *Workshop on Foundation of Deductive Databases and Logic Programming*, 1986.

[87] G. Langran. *Time in Geographic Information Systems.* Taylor and Francis, 1992.

[88] R. Lanzelotte and P. Valduriez. Extending the search strategy in a query optimizer. In *Proc. 17th Int'l Conf. Very Large Data Bases*, pages 363–374, Barcelona, Spain, Sept. 1991.

[89] B. Lindsay, J. McPherson, and H. Pirahesh. A data management extension architecture. In *Proc. 1987 ACM-SIGMOD Int'l Conf. Management of Data*, pages 220–226, San Francisco, 1987.

[90] J. W. Lloyd. *Foundation of Logic Programming.* Springer-Verlag, Berlin, 1984.

[91] D. Lomet and B. Salzberg. The hB-Tree: Guaranteed performance index method with a robust multiattribute search. *ACM Trans. Database Systems*, 15:625–658, 1990.

[92] W. Lu and J. Han. Decomposition of spatial database queries by deduction and compilation. In *Proc. 4th Int'l Symp. Spatial Data Handling*, pages 579–588, Zurich, Switzerland, July 1990.

[93] W. Lu and J. Han. Deductive spatial query optimization by dynamic connection graph transformation. In *Proc. 5th Int'l Symp. Spatial Data Handling*, pages 323–334, Charleston, SC, Aug. 1992.

[94] W. Lu and J. Han. Distance-associated join indices for spatial range search. In *Proc. 8th Int'l Conf. Data Engineering*, pages 284–292, Phoenix, AZ, Feb. 1992.

[95] W. Lu, J. Han, and B. C. Ooi. Knowledge discovery in large spatial databases. In *Proceedins of Far East Workshop on Geographic Information Systems*, pages 279–288., Singapore, June 1993.

[96] D. J. Maguire, M. Goodchild, and D. W. Rhind. *Geographical Information Systems: Principles and Applications.* Lonman Scientific and Technical, 1991.

[97] D. Maier, J. Stein, A. Otis, and A. Purdy. Development of an object-oriented DBMS. In *Proc. 1st Conf. Object-Oriented Programming Systems, Languages and Applications*, pages 355–392, 1986.

[98] D. Maier and D. S. Warren. *Computing With Logic.* Benjamin-Cummings, 1987.

[99] D. Mandelkern. Special section on graphic user interfaces: The next generation. *Communications of ACM*, 36:36–109, April 1993.

[100] J. Minker. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, 1988.

[101] L. Mohan and R. L. Kashyap. An object-oriented knowledge representation for spatial information. *IEEE Trans. Software Engineering*, 14:675–681, May 1988.

[102] S. Morehouse. ARC/INFO: A geo-relational model for spatial information. In *Proc. Digital Representations of Spatial Knowledge (AUTO-CARTO 7)*, pages 388–397, Washington D.C., March 1985.

[103] M.E. Mortenson. *Geometric Modeling.* John Wiley and Sons Inc., 1985.

[104] W. M. Newman and R. F. Sproull. *Principles of Interactive Computer Graphics.* McGraw-Hill, 1979.

[105] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The Grid File: An adaptable, symmetric multikey file structure. *ACM Trans. Database Systems*, 9:38–71, 1984.

[106] Object Design Obc. *ObjectStore Technical Overview.* Object Design Inc., Burlington, MA, 1992.

[107] B. Ooi, R. Sacks-Davis, and K. Mcdonell. Extending a DBMS for geographic applications. In *Proc. 5th Int'l Conf. Data Engineering*, pages 590–597, Los Angeles, Feb. 1989.

[108] B.C. Ooi. *Efficient Query Processing in A Geographic Information System.* Springer-Verlag, 1990.

[109] J. Orenstein. Strategies for optimizing the use of redundancy in spatial databases. In *Design and Implementation of Large Spatial Databases (Proc. 1st Int'l Symp. SSD'89)*, pages 115–133, Zurich, Switzerland, 1989.

[110] J. A. Orenstein and F. A. Manola. PROBE spatial data modeling and query processing in an image database application. *IEEE Trans. Software Engineering*, 14:611–629, May 1988.

[111] T. Pavlidis. *Algorithms for Graphics and Image Processing.* Computer Science Press, Rockville, MD, 1982.

[112] D. J. Peuquet. Representation of geographic space: Toward a conceptual synthesis. *Annals of the American Geographers*, 78:375–394, 1988.

[113] P. Pistor and R. Traunmuller. A database language for set, list and tables. *Information Systems*, 11:323–336, 1986.

[114] T. K. Poiker and N. Chrisman. Cartographic data structures. *The American Cartographer*, 2:55–69, 1975.

[115] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[116] R. Ramakrishnan, D. Srivastava, and S. Sudarshan. Coral - control, relations and logic. In *Proc. 18th Int'l Conf. Very Large Data Bases*, pages 547–559, Vancouver, Canada, Aug. 1992.

[117] R.L. Read, D.S. Fussell, and A. Silberschatz. A multi-resolution relational data model. In *Proc. 18th Int'l Conf. Very Large Data Bases*, pages 139–150, Vancouver, Canada, Aug. 1992.

[118] J.T. Robinson. The K-D-B tree: A search structure for large multidimensional dynamic indexes. In *Proc. 1981 ACM-SIGMOD Int'l Conf. Management of Data*, pages 10–18, Ann Arbor, MI, April 1981.

[119] Doron Rotem. Spatial join indices. In *Proc. 7th Conf. Data Engineering*, pages 500–509, Kobe, Japan, 1991.

[120] M. A. Roth. Extended algebra and calculus for nested databases. *ACM Trans. Database Systems*, 13:389–417, 1988.

[121] M. A. Roth and H. F. Korth. The design of ¬1NF relational databases into nested normal form. In *Proc. 1987 ACM-SIGMOD Int'l Conf. Management of Data*, pages 143–159, San Francisco, 1987.

[122] M. A. Roth, H. F. Korth, and D. S. Batory. SQL/NF: A query language for ¬1NF relational databases. *Information Systems*, 12:99–114, March 1987.

[123] N. Roussopoulos, D. Leifker, and T. K. Sellis. An efficient pictorial database system for PSQL. *IEEE Trans. Software Engineering*, 14:639–650, May 1988.

[124] D. Sacca and C. Zaniolo. Magic counting methods. In *Proc. 1987 ACM-SIGMOD Int'l Conf. Management of Data*, pages 49–59, San Francisco, 1987.

[125] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

[126] J. H. Saunders. A survey of object-oriented programming languages. In *J. Object-Oriented Programming*, pages 5–10, Mar/Apr. 1989.

[127] H. J. Schek, H.-B. Paul, M. H. Scholl, and G. Weukum. The DASDBS project: Objectives, experiments, and future prospects. *IEEE Trans. Knowledge and Data Engineering*, 2:25–43, March 1990.

[128] B. Seeger and H. Kriegel. Techniques for design and implementation of efficient spatial access methods. In *Proc. 13th Int'l Conf. Very Large Data Bases*, pages 360–371, Brighton, England, 1988.

[129] P. Selinger, D. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access path selection in a relational database management system. In *Proc. 1979 ACM-SIGMOD Int'l Conf. Management of Data*, pages 23–34, Boston, May 1979.

[130] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A dynamic index for multi-dimensional objects. In *Proc. 13th Int'l Conf. Very Large Data Bases*, pages 3–11, Brighton, England, 1987.

[131] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities. *Comm. ACM*, 34:94–109, 1991.

[132] H. Six and P. Widmayer. Spatial searching in geometric databases. In *Proc. 4th Int'l Conf. Data Engineering*, pages 496–503, Los Angeles, Feb. 1988.

[133] M. Stonebraker, editor. *Readings in Database Systems.* Morgan Kaufmann, 1988.

[134] M. Stonebraker, L. A. Rowe, and M. Hirohama. The implementation of the POSTGRES. *IEEE Trans. Knowledge and Data Engineering*, 2:125–141, 1990.

[135] D. Straube and T. Ozsu. Execution plan generation for an object-oriented data model. In *Proc. 2nd Int'l Conf., DOOD'91*, pages 43–67, Munich, Gemany, Dec. 1991.

[136] A. Swami and A. Gupta. Optimization for large join queries. In *Proc. 1988 ACM-SIGMOD Int'l Conf. Management of Data*, pages 8–27, Chicago, 1988.

[137] R. E. Tarjan. *Data Structures and Network Algorithms.* Philadephia Pa: Society for Industrial and Applied Mathematics, 1983.

[138] R. F. Tomlinson. *Geographic Information Systems and Cartographic Modeling.* Prentice Hall, 1990.

[139] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Vols. 1 & 2.* Computer Science Press, Rockville, MD, 1989.

[140] P. Valduriez. Join indices. *ACM Trans. Database Systems*, 12:218–246, June 1987.

[141] P. van Oosterom and J. van den Bos. *An Object-Oriented Approach to the Design of Geographic Information Systems.* Springer-Verlag, Berlin, 1990.

[142] K. Wilkinson, P. Lyngbaek, and W. Hasan. The Iris architecture and implementation. *IEEE Trans. Knowledge and Data Engineering*, 2:63–75, 1990.

[143] A. Wolf. The DASDBS GEO-Kernel: Concepts, experiments, and the second step. In *Design and Implementation of Large Spatial Databases (Proc. 1st Int'l Symp. SSD'89)*, pages 67–88, Santa Barbara, July 1989.

[144] E. Wong and K. Youssefi. Decomposition – a strategy for query processing. *ACM Trans. Database Sys.*, 1:223–241, 1976.

[145] L. Wong. Inference rules in object oriented programming systems. In *Proc. 1st Int'l Conf. Deductive and Object-Oriented Databases (DOOD'89)*, pages 493–510, Kyoto, Japan, December 1989.

[146] J. Woodwark, editor. *Geometric Reasoning.* Clarendon Press, 1989.

[147] X. Xu, J. Han, and W. Lu. RT-Tree: An improved R-Tree indexing structure for temporal spatial databases. In *Proc. 4th Int'l Symp. Spatial Data Handling*, pages 1040–1049, Zurich, Switzerland, July 1990.

[148] K. Yokota. Deductive approach for nested relations. In F. Fuchi and L. Kott, editors, *Programming of Future Generation Computer II*, pages 461–481. Elsevier Science Pub., 1988.

[149] C. Zaniolo. Object identity and inheritance in deductive databases – an evolutionary approach. In *Proc. 1st Int'l Conf. Deductive and Object-Oriented Databases (DOOD'89)*, pages 7–24, Kyoto, Japan, Dec. 1989.