Issues in Spatial Databases Indexing

CS641 Midterm (Fall 2002)

Ihab F. Ilyas
Department of Computer Sciences, Purdue University
ilyas@cs.purdue.edu

October 25, 2002

1 Introduction

Spatial data is any data with a location component. A spatial database is an efficient way to store vast amount of data that has spatial and non-spatial attributes. Efficient data structures are highly needed to facilitate access and to support the different spatial queries issued against this data. Since spatial objects can be very complex in shape and hence hard to present, processing queries on exact objects representation can be prohibitively expensive. *Generalization* techniques become essential requirement in processing large amount of data. Generalization aims at reducing the size and complexity of a spatial dataset with visual quality preserved. Spatial database system supports generalization by providing storage structures and a query processing strategies on the simplified representation of objects. Spatial indexes are data structures that organize the space hierarchically and establish a mapping between object approximation and sub-spaces. The main objective of a spatial index is to facilitate fast spatial data access to the stored objects.

There are numerous hierarchical data structuring techniques in use for retrieving spatial data. For example, the k-D tree [5], the point quadtree [18]. the trie index [9, 12], the fixed grid [20], the region quadtree [11], and other quadtree variants, are widely used in accessing spatial data. All of these structures depend on recursively partitioning the underlying space into disjoint partitions. One big advantage of these structures over the currently used R-tree is the *non-overlapping* search path. As the space is decomposed into disjoint partitions, these partition are guaranteed not to overlap and hence there is a single path from the root of the tree to the leaf that contains individual data items. This property is highly desirable in many applications especially in high-dimensional data types on the overlapping between bounding regions causes rapid deterioration to the query performance.

2 What Are We Trying To Solve?

For a database system to have a spatial support, two basic research challenges need to be addressed.

- Spatial data abstraction (spatial data model) that hides the details of object storage and representation. Exampled include defining new abstract data types (ADT) in object-relational databases. What is more important is a standardization of these ADTs. For example, The OGIS consortium has proposed a specification for incorporating 2-D geospatial ADTs in `SQL`. The proposed ADT consists of *Point*, *Curve* and *Polygon* classes [6].

- Developing spatial index structures and physical data organizations that provide efficient access paths for various spatial objects. Although many index spatial structures are proposed, each of them is suitable for a subset of the spatial data types and can handle a subset of the query types efficiently.

- Developing query processing algorithms for these data structures to be used by the `DBMS`. Range queries, nearest neighbor queries, bulk loading, dynamic insertion and deletion are the basic needs for most spatial applications.

3 Why It Is Not There!

Although many structures are introduced in the literature for more than two decades also, many searching, loading and updating algorithms have also been proposed, the adoption of these techniques by the commercial database systems is very limited. In the next section we will give examples on the current support for spatial processing in today's commercial `DBMS`s. There are two main obstacles that hinders this adoption [3].

- The *storage/structure characteristics* of spatial trees. Most of the *unbalanced* spatial tree structures are not optimized for I/O, which is a crucial issue for database systems. Unless the problem of appropriately clustering the tree nodes into pages is addressed properly, this would lead to many I/O accesses before getting the required query answer. Compare this to the B+-tree, that in most cases has a height of 2-3 levels.
- The *implementation effort* of building indexes. Hard wiring the implementation of a full fledged index structure with the appropriate concurrency and recovery mechanisms into the database engine is a non-trivial process. Repeating this process for each spatial tree that can be more appealing for a certain application requires major changes in the DBMS core code. After all, one may still need a new structure that will cause, rewriting/augmenting significant portions of the DBMS engine to add the new tree index.

To make these spatial access techniques feasible, the following issues need to be addressed:

- The I/O performance of these trees need to be enhanced to justify their benefits. Techniques like node clustering and minimizing maximum access path are well studied techniques and can be easily applied.
- Due to the larger number of these trees with each suits only a subset of the applications, there is no point of choosing any of them to implement in a commercial system. Of course, implementing all of them needs huge number of man-hours in an almost pain/no-gain process. What commercial systems need is a well-defined framework to accommodate for most of these trees with single core implementation. Whenever the application needs a new spatial index structure, a new tree can be defined without messing with the internal system code which is usually very hard to change.
- Along with the framework, scalable and efficient basic operations need to be defined. Insertion, deletion, single-item search, range queries and nearest neighbor queries need to be general yet efficient to be used as first-class operations. These algorithms need to take care of many practical issues such as concurrency control in order to be adopted commercially.

4 SP-GiST: An Example Framework

We developed a prototype spatial indexing framework that tries to address the points discussed earlier. The SP-GiST framework [3], is generalized index that supports the class of space partitioning trees (trees that recursively partition the space into finite number of disjoint partitions). We would like here to present some of the conducted experiments using SP-GiST to prove the feasibility of the ideas discussed in Section 3.

These experiments prove the conceptual idea of the SP-GiST. Using the SP-GiST interface parameters and methods [3] we implemented the basic version of the trie data structure and the quadtree. We change the interface parameters slightly to obtain different versions of the trie and the quadtree without any change to the core implementation. Figure 1 gives a comparison between three versions of the trie while Figure 2 gives the same experiment on the quadtree.

Input : The SP-GiST core implementation and the user interface parameters and methods.
Output: Three different implementations of the trie(the basic trie, the trie and the Patricia trie) and three different quadtrees (the PR-quadtree, the MX-quadtree and the Patricia quad) .
Parameters: We change the settings for the parameter *ShrinkPolicy* the limits the number of single-child nodes to change the tree shape. On the X-Axis we change the parameter *BucketSize* that limits the number of words per leaf node.

Metrics: We compare the resulting structures with respect to the maximum path length in pages and the total number of index pages used.
Dataset: A dataset of 100000 records with “string” keys for the trie and a dataset of 100000 points for the quadtrees.
Observation: Using the same implementation, different trie and quadtree versions can be implemented with different query performance.

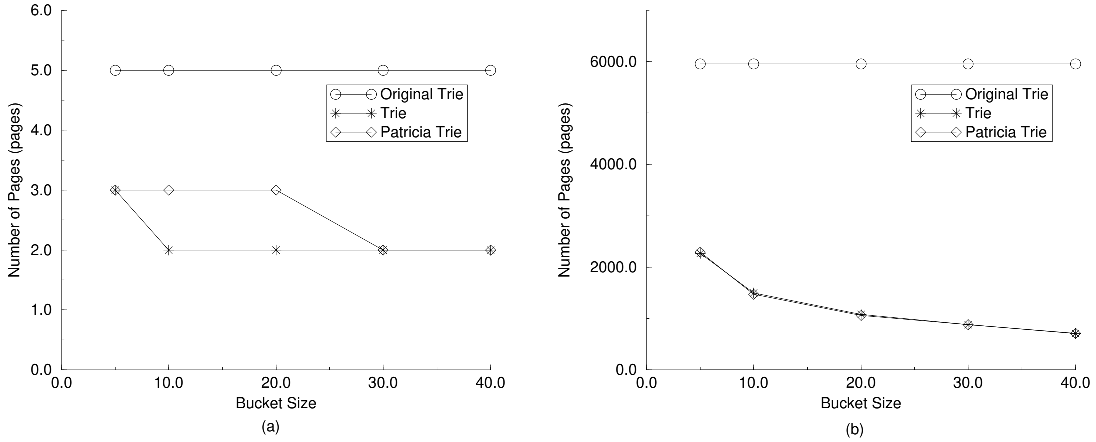

Figure 1: Implementing Different Versions of the Trie by Changing the Interface Parameters

One interesting observation from these experiments is the possibility of “tailoring” new data structures. There is no structure called *Patricia* quadtree (a quadtree with no single-child nodes) in the literature, yet, the implementation of such data structure was as easy as changing one or two parameters in the SP-GiST interface. That means different combination of the available interface can be set by applications to design new spatial indexes that best suit the applications needs.

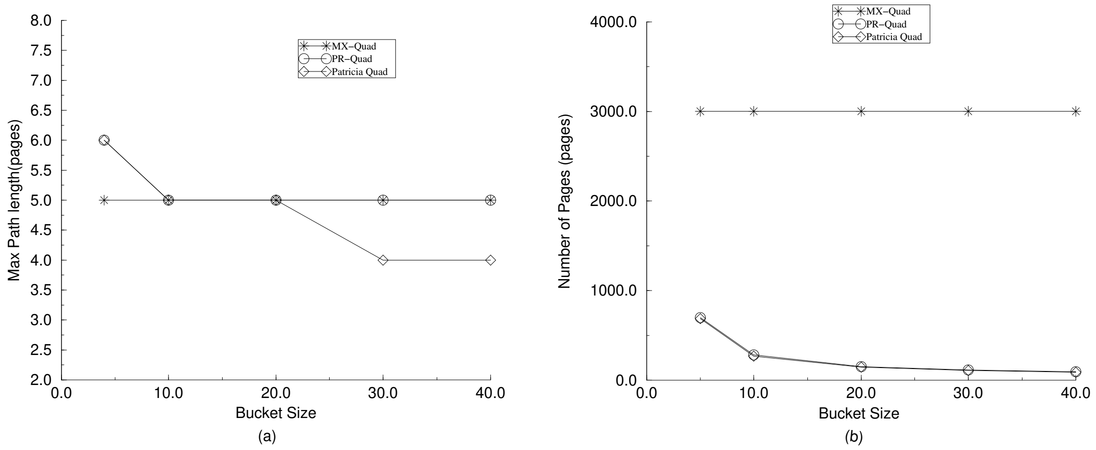

Figure 2: Effect of *BucketSize* on maximum path length for different settings of *ShrinkPolicy* for the quadtree.

5 Current Support In Commercial Systems

Due to the difficulties aforementioned in Section 3, the support of spatial index structures by commercial systems has been very limited. The support of current commercial database systems can take the following forms: The DBMS chooses one widely used spatial index structure and implement it as a core indexing mechanism besides the B-tree (and possible the R-tree). For Example, Oracle8 Spatial Cartridge [7, 8] provides an efficient implementation of the linear quadtree and implementation of the R-tree [13] to index spatial objects. The IBM Informix Spatial DataBlade [16] provides only R-tree implementation. The R-tree is usually chosen to be implemented because of its I/O characteristics that are similar to that of the B-tree although it may not be the best data structure to suit the application needs. Other spatial data structures like k-D tree, quadtree variants and the trie are not chosen by commercial systems because of their structure characteristics (they are unbalanced and have limited fanout)

Another form of spatial–support adopted by almost all commercial database systems–is to provide a *spatial extension* where application can define their own spatial indexes on top of the ordinary supported index structures such as B-tree and R-tree. An example of this approach is DB2 Spatial Extender [15] where application can create simple spatial indexes on top of the existing B-tree implementation. In this case the spatial index is in a intermediate layer between the core engine layer and the application layer.

A more promising approach is to implement a *Generalized Spatial Index* framework like the one described in this report (the SP-GiST) inside the core database system. Application can choose between a variety of of spatial indexes by only providing some interface parameters and methods. This approach is already implemented for non-spatial trees in Informix Dynamic Server with Universal Data Option (IDS/UDO) [19]. The idea is to use the *Generalized Index Search Tree (GiST)* [14] developed at Berkeley to support the class of B-tree like indexes (B-tree, R-tree, SR-tree [17], SS-tree [21] and other R-tree variants) with one core implementation.

6 Academic Efforts and Future Needs

There has been different prototypes developed by researchers to support spatial index structures not necessarily as a part of a database engine. The SAND project at the university of Maryland [4, 10]. Another interesting spatial database prototype is PostGIS [1]. PostGIS adds support for geographic objects to the PostgreSQL object-relational database [2]. In effect, PostGIS "spatially enables" the PostgreSQL server, allowing it to be used as a backend spatial database for geographic information systems (GIS), much like Oracle's Spatial extension. PostGIS follows the OpenGIS "Simple Features Specification for SQL". Here at Purdue we prototypes the Generalized spatial index framework, SP-GiST. Variants of the trie, the quadtrees and the k-D tree are implemented with the same core implementation and has been used in some data mining research projects.

What we need now is a set of generalized query processing algorithms to complete the picture. Together with an extensible spatial index structure, a real spatial extension can be incorporated into commercial database systems. This will allow for supporting a wider range of spatial access paths than the conventional R-tree with minimal implementation efforts. Many search algorithms (point, range and nearest neighbor search algorithms) have been developed for individual trees and we believe that their generalization is also possible due to the common structural features among these spatial indexes.

References

[1] *http://postgis.refractions.net/home.php.*

[2] *http://www.us.postgresql.org/index.html.*

[3] Walid G. Aref and Ihab F. Ilyas. Sp-gist: An extensible database index for supporting space partitioning trees. In *Journal of Intelligent Information Systems (JIIS)*, volume 17. IEEE Computer Society, 2001.

[4] Walid G. Aref and Hanan Samet. Optimization for spatial query processing. In *Proceedings of VLDB'97*, pages 81–90, Barcelona, Catalonia, Spain, September 1991.

[5] Jon L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 19:509–517, 1975.

[6] Open GIS Consortium. Opengis® Simple Features Specification for SQL, revision 1.1. *http://www.opengis.org/techno/specs/99-049.pdf*, May 1999.

[7] Oracle Corporation. Oracle Spatial (data sheet). *http://otn.oracle.com/products/oracle8/htdocs/xsdotwp3.htm*, March 1999.

[8] Oracle Corporation. Oracle spatial option: Location-based services for oracle9i. *http://otn.oracle.com/products/spatial/content.html*, 2002.

[9] Rene de la Briandais. File searching using variable length keys. In *Proceedings of the Western Joint Computer Conference*, pages 295–298, 1959.

[10] Claudio Esperanca and Hanan Samet. Spatial database programming using sand. *Proceedings of the Seventh International Symposium on Spatial Data Handling*, May 1996.

[11] R. A. Finkel and J. L. Bentley. Quad trees: a data structure for retrieval on composite key. *Acta Informatica*, 4(1):1–9, 1974.

[12] E. Fredkin. Trie memory. *Communications of the ACM*, 3:490–500, 1960.

[13] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proceedings of SIGMOD'84*, pages 47–57, June 1984.

[14] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized search trees for database system. *Proceedings of VLDB'95*, 1995.

[15] IBM. IBM DB2® Spatial Extender: User's guide and reference. *ftp://ftp.software.ibm.com/ps/products/db2/info/vr7/pdf/letter/db2sbe70.pdf*, March 1999.

[16] IBM Informix. The IBM Informix Spatial Datablade. *http://www-3.ibm.com/software/data/informix/blades/spatial/*.

[17] Norio Katayama and Shin'ichi Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. *SIGMOD Record*, 26(2), 1997.

[18] A. Klinger. Pattern and search statistics. *In S. RUSTAGI Ed., Optimizing Methods in Statistics*, pages 303–337, 1971.

[19] Marcel Kornacker. High-performance extensible indexing. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *Proceedings of the Twenty-fifth International Conference on Very Large Databases, Edinburgh, Scotland, UK, 7–10 September, 1999*, pages 699–708, Los Altos, CA 94022, USA, 1999. Morgan Kaufmann Publishers.

[20] J. Nievergelt, H. Hinterberger, and K. Sevcik. The grid file: an adaptable symmetric multi-key file structure. *ACM Transactions on Database Systems*, 9(1):38–71, 1984.

[21] D. A. White and R. Jain. Similarity indexing with the SS-tree. In *Proceedings of ICDE'96*, pages 516–523, Washington - Brussels - Tokyo, February 1996.