Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Sciences 167 (2004) 41–61
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

A web-based spatial data access system using semantic *R*-trees

Shu-Ching Chen [a,*], Xinran Wang [b,1], Naphtali Rishe [a], Mark Allen Weiss [a]

[a] *School of Computer Science, Florida International University, Miami, FL 33199, USA*
[b] *Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, USA*

Received 27 September 2002; received in revised form 8 May 2003; accepted 4 July 2003

Abstract

With the increasing use of geographical data in real-world applications, Geographic Information Systems (GISs) have recently emerged as a fruitful area for research. In order to provide information to a multitude of users, the World Wide Web (WWW) techniques have been integrated into GISs. A high-performance web-based GIS, called TerraFly, has been developed in order to provide web-based GIS accesses to the general public. The design of TerraFly considers two major aspects: (1) the system architecture including client, database server, proxy server and information server; and (2) the semantic *R*-tree data structure and semantic queries. The system architecture utilizes the existing resources to achieve maximum performance by using the "Internally Distributed Multithreading (IDMT)" technique. The spatial access method, semantic *R*-trees, is used to search for an object, based on both spatial and semantic information. System performance results are presented and analyzed. Reducing network traffic to achieve faster response to users is also discussed.

Keywords: Spatial data access; Web-based system; Semantic *R*-tree

* Corresponding author.
E-mail address: chens@cs.fiu.edu (S.-C. Chen).
[1] Research for this project conducted as a graduate student at School of Computer Science, Florida International University.

0020-0255/$ - see front matter
doi:10.1016/j.ins.2003.07.019

1. Introduction

Recently, the use of Geographic Information System (GIS) with spatial data has become more and more popular. There exist a number of GIS applications such as ArcInfo [23], Imagine [19], and ENVI [21] to manipulate and view different spatial data. All applications make use of the geographical data and the multimedia tools for animation, simulation, and visualization purposes. GIS applications have been used in a wide-range of domains such as architecture, telecommunication, urban planning, and entertainment industries.

With the exponential growth of the World Wide Web (WWW), more domains are open to the GIS applications. The Internet can spread information to a multitude of users, making GIS available to a wider range of public users. The corresponding change of domain users from GIS experts to general-public users requires the consideration of some new features and design issues for GIS. For instance, we have to refine the system architecture for Internet usage, taking into account factors such as differing platforms across the Internet, network speed, and so on. A major consideration is to make a GIS accessible to the general public users who have little knowledge of the spatial data, and to allow them to interact with the system to manipulate and retrieve information they need. In order to address these issues, we have developed a user-friendly web-based multimedia system called TerraFly [9]. The TerraFly system lets the remote users on various platforms interact with the system, exploit spatial data of their interest, and issue queries to retrieve information they need without the need for extensive training with the system, thus avoiding a painful experience.

Numerous GIS applications, both academic and commercial, have been developed [17–22,24,25]. The existing research is extensive and advanced, and focuses on the following areas:

(1) database systems to efficiently store and retrieve heterogeneous spatial data;
(2) spatial data analysis capabilities;
(3) spatial data indexing methods.

Even though there have been great achievements over the past decades on these areas, a web-based GIS application still suffers from the following drawbacks:

(1) Inability of databases to efficiently handle large and different spatial data sets.
(2) Tendency for complicated WWW technologies and distributed computing to add complexity to the system.
(3) Significant degradation of system performance for many GIS applications that are misconfigured.

The dramatically increased availability and usage of the spatial data require the use of advanced methodologies and technologies. In this paper, the methodologies and techniques used to improve performance for a web-based GIS application are discussed. Overall performance of a web-based GIS depends on the design of the system architecture. For a distributed GIS system, a desired design is to best utilize the existing resources to obtain maximum performance. To achieve this goal, a model called *Internally Distributed Multithreading Model* (*IDMT*) has been designed for the TerraFly system. In this model, the distribution of threads is based on the different functionality each thread may have, enabling the system to better utilize the server CPU and other resources.

In addition, it is necessary to have a good spatial data structure that allows efficient accesses of the spatial data in a GIS system. The *R*-trees or its variants are a widely-used data structure in GIS systems [1–5,13–15]. The *R*-tree is based on a heuristic optimization to minimize the area of each enclosing rectangle in the inner nodes, and is one of the most efficient methods that support range queries. However, in a GIS system, the users will be more interested in issuing semantic queries such as "Find the nearest airport." The current spatial data structures such as an *R*-tree cannot efficiently support this type of semantic query. To provide this functionality, in this paper, a data structure called the *Semantic R-tree* that is extended from the *R*-tree is proposed for the TerraFly system. Our experimental results show that the proposed Semantic *R*-tree can provide significant savings in response time and outperform the well-known *R*-trees in answering semantic queries that the users often request.

This paper is organized as follows. In Section 2, our proposed system architecture for the TerraFly system is presented. The information server system structure is discussed in Section 3, and the server computation algorithm is given in Section 4. Performance analysis is presented in Section 5. Section 6 concludes the paper.

2. System architecture

The TerraFly system design is based on a three-tier client/server architecture as shown in Fig. 1. The second tier handles all application logic: namely, it retrieves data requested by the clients and answers queries for the clients. Java 1.1 is used in developing the client side to generate byte codes running across different platforms. Each client is a data-less graphical user interface (GUI). The database server stores and retrieves multidimensional spatial data such as image data as well as alphanumeric data. Together, the client/server architecture forms a complete computing system with a distinct division of responsibility.

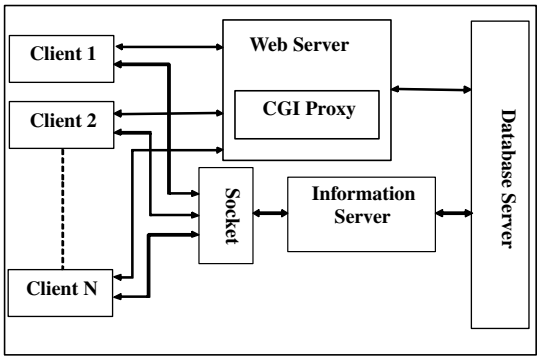

Fig. 1. TerraFly system architecture.

The Java clients use a connection wrapper to synchronize data transmission. The clients send requests to either a web server or an information server, and receive data (image data and textual data) from different servers. When the web server receives a request from a client, the proxy server will parse the request, retrieve the image and textual data related to the fly-over request from the database server, and send the data back to the client. The information server will search the semantic *R*-tree spatial data structure to retrieve information related to range queries and nearest neighbor queries [14] from the database server, and send the answers to the client.

2.1. Java client

The client is written using Java 1.1 to achieve platform independence. A snapshot of the client GUI is shown in Fig. 2. Along the top center, there is a choice box to select the desired image format source. For example, selecting *Landsat* will make the client show the *Landsat* images. Next to this data source selection are three drop down choice boxes for selecting which bands of *Landsat* to use to create a color composite image. Depending on the user's selection, different bands will be retrieved from the server. A user can also click on the "Feature Highlight" button to select features to highlight. The main features of the client include:

(1) Capability to fly over different data sets such as the *Landsat* TM data and Digital Orthophoto Quad (DOQ) at different directions by positioning the mouse within the image. The *Landsat* data, currently stored in the database, covers an area of 57 miles $\times$ 50 miles and corresponds to the Florida State map region. This data has a resolution of 30 m.
(2) Capability to fly over the Digital Orthophoto Quad (DOQ) merely by positioning the mouse within the image. The aerial photography data cur-

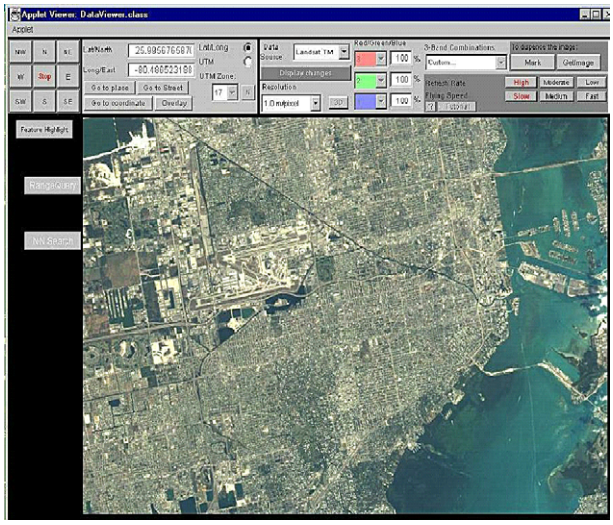

Fig. 2. A TerraFly client.

rently stored in the database has a resolution of 1 m and covers an area of 1400 square miles.

(3) Customized three-band (sensor) combination: the users can select some predefined and useful three-sensor combinations to view false color images from a drop down menu. Each time a new combination is selected, a different set of images/bands is retrieved from the database and the selected false color image is computed and displayed within the window.
(4) Advanced three-band color composite: this application allows a scientific user to enter any three-band combinations (RGB) that the user is interested in studying or analyzing. After the combination is entered and the query button is pressed, the data is retrieved from the database, and the image is computed and displayed. For the *Landsat*, the users are allowed to select from a list of seven possible sensors. However, for aerial photography data, there are only three possible sensors.
(5) A Help menu with instructions on how to use the system.
(6) Capability to display the latitude and longitude of the center point of an image.
(7) RGB intensity control: this option allows the users to increase or reduce the intensity of any of the bands that represent the colors (red, green, or blue).
(8) Capability to issue semantic range queries and nearest neighbor queries.
(9) Capability to obtain feature extraction of an image.
(10) Capability to display online information (latitude, longitude, regions, etc.) of the images that users are viewing.

2.2. Database server

The database server contains a multimedia spatial database system built by HPDRC (High-Performance Database Research Center) [12] using the Semantic Object-Oriented Database Management System (Sem-ODB) [10,11] based on the Semantic Binary Object-Oriented Model [8]. In the Semantic Binary Object-Oriented Model, information is represented by logical associations (*relations*) between pairs of objects and by the classification of objects into *categories*. Unlike the traditional database systems that store only alphanumeric data, the Sem-ODB has not only alphanumeric data but also the data that cover multidimensional spaces such as image data (maps). Currently, the database contains semantic/textual, spatial/remote sensed (*Landsat*) and digital data including Digital Orthophoto Quad (DOQ) Arial photograph data. The Sem-ODB provides an efficient data storage and manipulation methodology. The database engine by itself is implemented in approximately 75,000 lines of code.

2.3. Proxy server

A proxy server is used to relay data requested by the clients to the database server, and to transmit the data retrieved by Sem-ODB back to the clients. This proxy needs to use two different protocols, one to interface with the clients, and another to interface with the Sem-ODB server. We use Common Gateway Interface (CGI) as the first protocol and Sem-ODB's APIs as the second protocol. Upon receiving a request from a client, the proxy server must decode the request and retrieve data from the database server. The proxy process accomplishes this by using the Sem-ODB APIs. What the proxy server needs to do is to find the location of the database server [16]. After the database connection is established, all queries can be performed using the APIs provided by the Sem-ODB.

2.4. Information server

The TerraFly system as a whole provides an integrated view of the spatial and associated data along with the capability to display and manipulate spatial images. Also, being a GIS system, it provides the capabilities to issue range queries and nearest neighbor queries to satisfy particular interests of the scientists and public users. Range queries are used to find spatial objects in a specific area around a location specified by the user. A sample query is "Find all rental car companies around Miami International Airport within six miles." Nearest neighbor queries are used to find the nearest spatial object to the object that the user specifies, for instance, "Find the car rental company nearest to Miami International Airport." The information server is designed for an-

swering these two types of queries and it is a multithreaded application. It receives requests from the clients through a Unix socket. The configuration of the information server based on the IDMT model is shown in Fig. 4, which is discussed in detail in Section 3.

3. Information server structure

The server is designed in a multithreaded environment (using POSIX threads [6]). The basic concepts and the research of multithreaded programming have existed for several decades, but the emergence of this concept in industry as an accepted standardized programming technique is relatively recent. The greatest push is from the emergence of shared memory symmetric multiprocessors (SMP) [6]. Multithreading provides exactly the right programming paradigm to obtain greater performance by making maximal use of these machines. Using threads has several significant advances over using processes. Each process must maintain a complete process structure, including a full virtual memory space and kernel state. It takes time and memory to create a new process and maintain it. A kernel call is needed to create a process and system's context switching mechanism may be involved. Interprocess communication and synchronization of shared data may involve kernel calls. Alternatively, creating threads is much easier compared to creating a process. A thread can be created without replicating an entire process. The threads, their stacks, the code they run, the global data they share and the thread structures are in the user space, rather than in the kernel. Thread synchronization does not involve system calls trapped into the kernel.

In the TerraFly Information Server, we use POSIX threads to enhance performance. The server is based on the IDMT model that will be described next.

3.1. Internally distributed multithreading model

In a general model, the server is the main application logic for the whole GIS system, and this server is further divided into three sub-logic components: (1) communication module, (2) computation module, and (3) data structure (as shown in Fig. 3). The communication module is used to communicate with the clients and to perform certain first phrase computations when it receives requests from the clients. Major computation work is processed in the computation module that searches the data structure to find the results and to retrieve the information that the clients need.

In this general model, when a client sends a request to the server, the main thread will generate a thread to handle this request. That is, this worker thread takes over the communication channel, receives the request query, performs

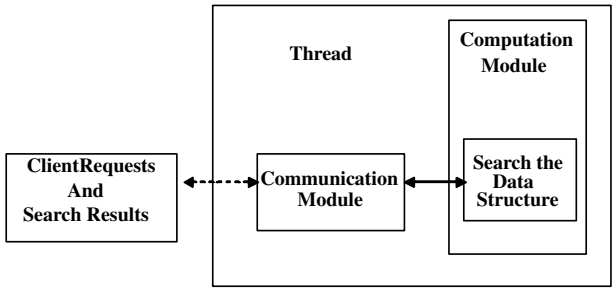

Fig. 3. A general model—a server using multithreading with each thread having three modules.

the computation (searches the data structure), and then transmits the results back to the clients. It is evident that the use of threads improves the overall performance, but the overhead of using threads is still quite high.

In order to improve system performance, an IDMT model is proposed. Fig. 4 shows the proposed server structure. The backbone of this structure is a thread pool [7] containing a number of threads that do computation in the

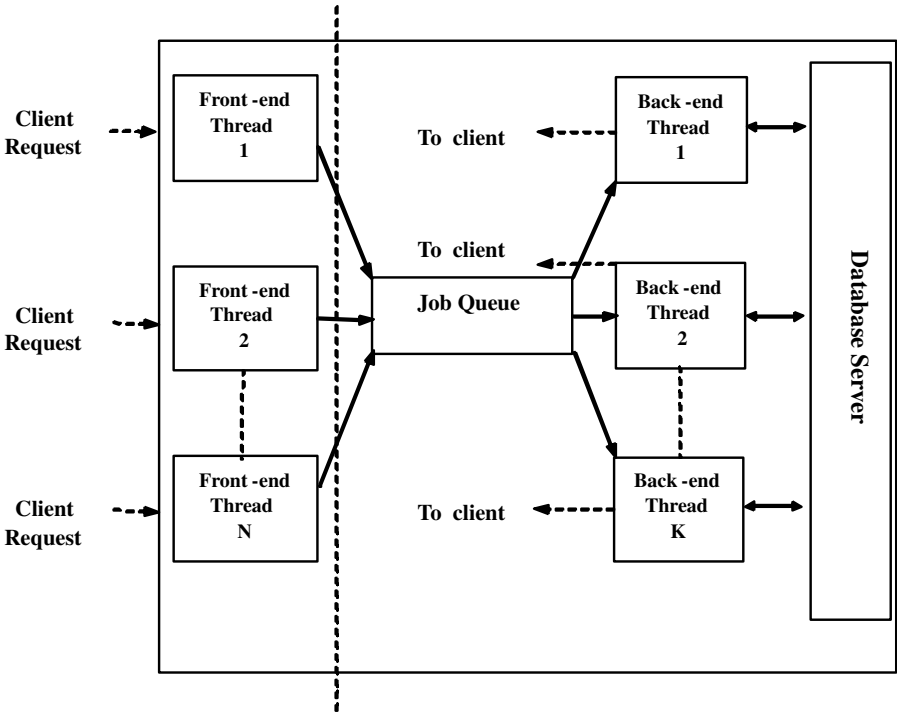

Fig. 4. The internally distributed multithreading (IDMT) model design.

back end. We call these working threads the back-end threads that are dedicated to communicate with the clients. The experimental results will show that this structure not only achieves better performance than the general model, but also has a better scalability property.

3.2. Analysis of internally distributed multithreading

As mentioned earlier, the front-end communication threads are dedicated to communicating with the clients and performing first-phrase computations, such as converting the data types and getting query results. All requests from the client-side are sent to the front-end threads at the server. When these threads receive the requests, they analyze the requests based on the agreed-to protocols between the server and the client, and post the analyzed requests to the job queue, where the back-end threads take over the job.

Normally, there are two types of requests from a client: a single request and several continuous requests. For a single request, a client sends the request and gets the results from the server. On the other hand, for the continuous requests, a client continuously sends the requests (for instance, a video stream) to the server. Sometimes, the clients send a large number of requests in a short period of time. In the general model, one thread dedicated for each client is needed to handle the communication and computation. If there are many requests from one client, the thread dedicated to the client will have a heavy load, and thus performance degenerates. However, in the proposed IDMT model, the server is capable of dealing with these kinds of requests efficiently. When a server gets the requests, the front-end threads distribute these jobs to the back-end threads that do the computation concurrently. This distribution improves the server in the following aspects:

(1) *Throughput*: If a client sends multiple requests, the response time in the IDMT model is much less because the server distributes the requests to the back-end threads that run concurrently to solve the requests from the same client. The best case is that these threads compute almost simultaneously to get the results. Heavy load is distributed to several threads, making the system more robust and achieving a better response time.
(2) *Internal load balancing*: The back-end threads in the server thread pool perform the major computations. When the server detects a heavy load, such as too many requests from the clients, it determines that there are not enough back-end threads to handle the situation. The server can then generate some extra back-end threads on demand to relieve the heavy load. This dynamic thread generation can balance the server performance with a minimal cost of resources needed.
(3) *Scalability*: When we separate a big process into several upgradeable components, we actually gain a benefit from scalability. Each component

is independent from the other components and has its own functionality. If one component is upgraded, there is little impact on the other components. In our design, the communication module is separated from other modules. If another communication method is used, there is no effect on the server computation at all. The computation module consists of a number of back-end threads, which can actually be distributed to different powerful workstations. Thus, updating will have the minimal impact on the whole system.

4. Semantic *R*-trees for semantic queries

R-trees are an extension of *B*-trees for multidimensional objects. The details of the *R*-tree are discussed in [3]. An object in an *R*-tree is represented by its minimum bounding rectangle (MBR). The nodes are allowed to overlap and thus an *R*-tree can guarantee at least 50% space utilization and remain balanced. For completeness and clarity, the *R*-tree data structure is briefly described here.

Let M be the maximum number of entries of one node and $m = \lceil M/2 \rceil$ be a parameter specifying the minimum number of entries in a node. An *R*-tree structure specifies the following properties:

(1) All leaf nodes appear at the same level.
(2) Each entry in a leaf node is a 2-tuple of the form (*MBR*, *oid*) such that *MBR* spatially contains the object pointed to by *oid*.
(3) Each entry in a non-leaf node is a 2-tuple of the form (*MBR*, *ptr*) such that *MBR* spatially contains the *MBR* in the child node pointed to by *ptr*.
(4) Parameters m and M indicate that each node in the tree contains between m and M entries, with the exception of the root, which has at least two entries unless it is a leaf node.

There are various *R*-tree variants, such as R^+-trees [15], R^*-trees [1], packed *R*-trees [4,13]. R^+-trees avoid overlapping at the expense of space, which indirectly increases the height of the tree. The R^+-tree is motivated by a desire to avoid overlapping among the bounding rectangles. Each object is associated with all the bounding rectangles that it intersects. All bounding rectangles in the tree are non-overlapping. The result is that there may be several paths starting at the root to the same object. R^+-trees exhibit good searching performance, especially for point queries, at the expense of some extra space. An R^*-tree incorporates a combined optimization of area, margin and overlap of each enclosing rectangle in the directory. An R^*-tree also uses a different split algorithm. All entries are first sorted by the lower values, and then sorted by the upper values of their rectangles. After finding a good split, the R^*-tree

forces re-insertions to reorganize the tree structure. Beckmann et al. [1] found that this approach improves performance up to 20–50%. The Packed *R*-tree was proposed to achieve better space usage, where the *R*-tree is built bottom-up from a sorted collection of rectangles. In [4], the authors proposed the Hilbert value, and the rectangles are sorted by the Hilbert curve to achieve better ordering of the rectangles and eventually better packing. The performance of the Packed *R*-tree is up to 36% better than the best known *R*-tree variant.

An *R*-tree is based on heuristic optimization to minimize the area of each enclosing rectangle in the inner nodes. Most of the research on the *R*-tree focuses on minimizing overlapping MBR, and optimizing storage utilization. However, for a GIS system, improvements to the *R*-tree are needed to allow the *R*-tree to efficiently answer all kinds of nearest neighbor queries and range queries, of the types that are typical on a GIS system. In the next section, we show how to improve the *R*-tree (or other spatial data structure) for a GIS system, especially a GIS with static data.

4.1. Semantic query

Semantic queries are the queries to find the specified objects according to their relations to the base object. An example semantic query is "Find the rental car company nearest to Miami International Airport (MIA)." In this query, the base object is "MIA," the specified object (i.e., the semantic object) is "the rental car company" and the relationship is "nearest to." On the other hand, the query such as "Find an object nearest to MIA," where no attribute is specified for this object, belongs to a general query. A semantic object is defined as an object with some information that a user asks for. In the above example, a user wants to find a rental car company. This "rental car company" is one type of semantic information a user may specify. If a spatial data structure can support this type of information, query performance would be enhanced greatly.

The difference between the semantic queries and the general queries is subtle but significant. A general query formula is shown as follows.

$$Q = G_{i=1}^{K}(O_i R_i OB_i) \tag{1}$$

where

- G is a set operator that is either Union ($\bigcup$) or Intersect ($\bigcap$);
- K is the number of union and/or intersect operations;
- O_i is a set of objects to be found;
- OB_i is a base object; and
- R_i is a relation.

For each GIS system, there are a number of semantic subsets. Let S_j be one semantic subset in a GIS semantic set S ($S_j \subseteq S$) and H be the number of semantic subsets in S, where $S = \cup_{j=1}^{H}(S_j)$. A semantic object is $O^s = O(s)$, where s is the specified information ($s \in S_j$). A semantic query is of the following form:

$$Q_s = G_{i=1}^{k}(O_i(s)R_iOB_i) \tag{2}$$

A user provides a semantic query Q_s to request the information that he/she wants. For each semantic subset S_j, there are a limited (fixed) number of elements. For example, in a map GIS system, $S = \{S_1, S_2, S_3\}$, where

- $S_1 = \{\text{Dade county}, \text{Broward County}, \text{Orange county}, \ldots\}$ by *county*;
- $S_2 = \{\text{river}, \text{bridge}, \text{route}, \ldots\}$ by *type*; and
- $S_3 = \{\text{school}, \text{building}, \text{shopping center}, \ldots\}$ by *category*.

The semantic subset information can be used to construct a spatial data structure to answer the semantic queries more efficiently. For example, if a user specifies that he wants to find a rental car company nearest to him, the server will search the data structure not only by the spatial specifications, but also by this semantic information to find a car rental company. In order to optimize (at least partially) a data structure based on the semantic subsets, these subsets need to be identified first. Actually, this is not a problem at all in reality. Finding the semantic subsets is a practical issue, which can be accomplished by searching the data sets to identify different properties and to categorize these properties into different subsets. Some unnecessary subsets can be further pruned according to the features of each GIS system so that only a few subsets need to be taken into consideration.

4.2. Semantic R-trees

To better answer the semantic queries, a spatial data structure with built-in semantic information, called a Semantic *R*-tree is proposed. Without such built-in semantic information, a spatial data structure has difficulty answering a query such as "Find all schools within 20 miles of Florida International University" efficiently. Searching the proposed semantic *R*-tree will find all the objects having the specified relation (i.e., "within 20 miles" in this example) with the base object (i.e., "Florida International University"), and then further processing is required to get the desired objects (i.e., the "schools") for the query. With the built-in semantic information, some sub-trees containing unrelated information can be pruned, which makes semantic searching quite efficient.

In our design, the "category" semantic subset is used to build a semantic *R*-tree and the algorithm used is based on the *R*-tree's packing technique. For

each node, its semantic information is categorized. Then, in each category, the MBRs are sorted on the x or y coordinates of one of the corners of the rectangles. Sorting the MBRs is similar to the method proposed by Roussopoulos and Leifker [13]. In each category, the sorted list of rectangles is scanned and assigned to one R-tree leaf node until this leaf node is full or there is no data left for this category. The algorithm is shown in Table 1. Let M be the maximum number of entries of one node.

Because the semantic information is packed into an R-tree, there might be some underflow nodes (less than $M/2$ children). However, since only a fixed number of elements exist in one semantic subset and usually this number is small, there might be only a few underflow nodes. For instance, there are 37 elements in the "category" semantic subset including "park", "church", "river", and so on. At the leaf node level, each category probably will have one underflow node. If (fanout) M is 40 and there are 500 parks, then totally there are 13 leaf nodes, 12 of which are full. The last one (i.e., the 13th leaf node) has 20 pointers to the park objects. Since each category has at most one underflow node, and the number of categories is only a small number, the underflow should not be a problem in a semantic R-tree. Also, a semantic R-tree is packed in the first place, so it can have a better space utilization than a non-packed R-tree. Therefore, better performance for the semantic query can be achieved. As for a general query, since a semantic R-tree is not generated by fully optimizing

Table 1
Pseudo-code of the Semantic-packing algorithm

Semantic-packing algorithm

Step 1: Select one semantic subset that is best for one GIS system.
Step 2: Categorize the data according to the selected semantic subset.
Step 3: For each category {
$l = 0$;
Step 4: Sort the data in this category based on the x or y coordinate.
Step 5: /* create the child node at level l (leaf nodes are at level 0) */
While (there are data in the sorted list) {
If (the remaining P data in the same category $==$ 1) Stop;
Generate a new R-tree node.
If ($P >= M$)
Assign the next M data into this node.
Else
Assign the next P data into this node.
}
Step 6: /* create the parent node at level $(l + 1)$ */
While (there are nodes at level l)
Sort the nodes at level l based on their generation times.
$l = l + 1$;
Go to **Step 5**.
}

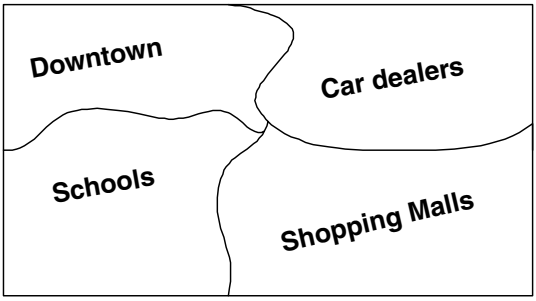

Fig. 5. Each section has high density of specific data.

the spatial factors, there may be some degree of degeneration of the performance when answering a general query. However, this is not always the case, depending on the features of the static data. Suppose a map GIS has several regions with the same semantic data (e.g., "car dealers" in Fig. 5). Then, a semantic *R*-tree will have a tree structure that is more condensed than an *R*-tree. In this case, a semantic *R*-tree performs better than an *R*-tree even in answering general queries.

We state two observations. Experimental results that justify these observations are discussed in Section 5.

- *Observation* 1: Even if a semantic *R*-tree is partially optimized by semantic information, there are still many instances where a semantic *R*-tree outperforms an *R*-tree in answering a general query. If in some cases, the performance of a semantic *R*-tree degenerates in answering a general query, the degeneration is minimal.
- *Observation 2*: A semantic *R*-tree data structure has better performance in answering semantic queries.

4.3. Semantic range query and nearest neighbor query examples

As mentioned in Section 2.1, the TerraFly Java client supports semantic range queries and nearest neighbor queries. A user can click on the "Range Query" button to issue a range query based on his/her interest. For example, in Fig. 6, the user wants to find all "parks" around the location specified as a (longitude, latitude) pair (−79.558, 25,8802) within a 15 mile range. In this example, the user is searching the category "park" around the specified location. The default category, "ALL," finds everything around the user's location. After the user clicks the "Search" button, the information server gets the client's request and searches the *R*-tree. The server will find the result(s) and transmit information back to this client. After the client obtains the informa-

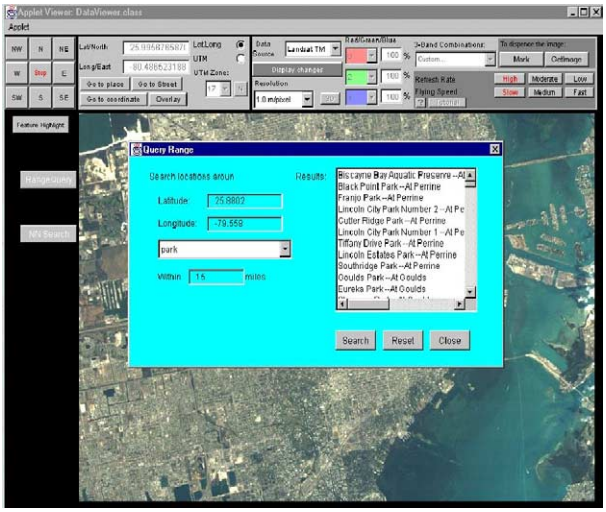

Fig. 6. Range query searching and its results listed in *Results* List Box.

tion from the information server, it displays the result(s) in the "Result" list box. If one result is double-clicked (e.g., "Black Point Park"), this particular region will be moved and displayed at the center of the display window in TerraFly.

The operation of nearest neighbor searching is similar to the range query searching. That is, the user can click "NN search" to find a nearest neighbor as shown in Fig. 7.

5. Performance analysis

5.1. Experimental results for internally distributed multithreading model

A number of experiments were performed to demonstrate the effectiveness of the IDMT model. The procedure used was that the client sends a number of continuous requests to the server, and the response time of the server was measured. To test the relative performance of the IDMT, a server with the IDMT model and a server with the original thread pool model [7] were compared. A thread pool model is a very common and very important design technique to improve the performance. In a thread pool model, the main thread (boss) creates a fixed number of worker threads when the system boots up. The number is specified by the designer in order to achieve the best overall performance. All worker threads survive for the duration of the program as the boss does. When the main program receives a new request, it places the request

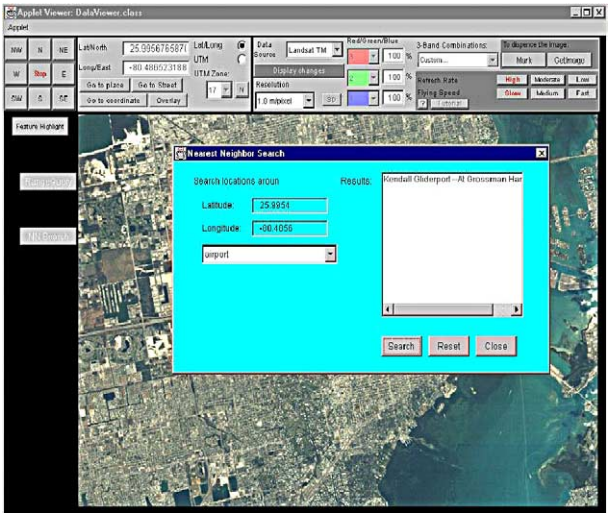

Fig. 7. Nearest neighbor searching.

in a queue. Workers remove the requests from this queue and process them. When a worker completes a request, it returns and gets another request from the queue. If this queue is empty (no jobs for this program), all workers wait for the new coming requests. The results are shown in Figs. 8 and 9.

In Fig. 8, the X-axis is the number of continuous requests sent and the Y-axis is the relative response time: the response time of the server with the original thread pool model divided by the response time of the server with the IDMT model, where the critical value is 1.0. If the test value (the ratio) is greater than 1.0, it means that the server with the original thread pool model uses more time to find the results than does the server with the IDMT model. On the other hand, if the value is less than 1.0, it means that the server with the original thread pool model outperforms the server with the IDMT model. In the experiment, a range of requests (from 1000 to 30,000) were sent by a single client to the server. From Fig. 8, it is clear that the response time of the server

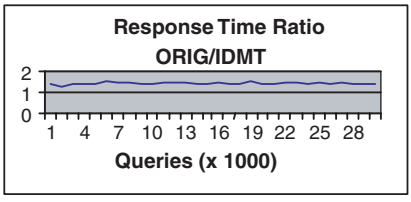

Fig. 8. Response time ratio of the server with the original thread pool model and the server with the IDMT model, in the case of that one client sends continuous requests.

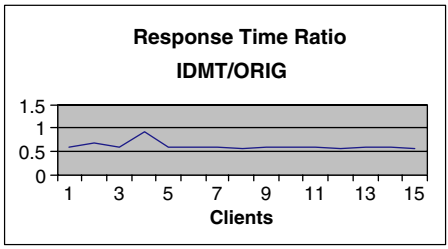

Fig. 9. Relative response time in the case of multiple clients.

with the original thread pool model is much over 140% of the response time of the server with the IDMT model, which means that the response time of the server with the IDMT model achieves at least 40% better performance than that of the server with the original thread pool model.

In Fig. 9, the X-axis is the number of multiple clients and the Y-axis is the ratio of the response time of the server with the IDMT model divided by the response time of the server with the original thread pool model handling multiple clients. Similarly, if the ratio is less than 1.0, then the server with the IDMT model spends less time answering the requests from multiple clients, which indicates better performance. In this experiment, to avoid the scenarios that the server finishes the request from one client very soon, each client sent a huge job to the server. The server processed all requests from all clients together when they sent their requests. In Fig. 9, the result is very similar to the result shown in Fig. 8. The response time of the server with the IDMT model is nearly 57% of that of the server with the original thread pool model. That is, the performance of the server with the IDMT model is more than 40% better than that of the server with the original thread pool model.

5.2. Experimental results for semantic R-trees

The semantic R-tree was implemented and several experiments were conducted to demonstrate the capability and usefulness of the proposed semantic approach when it is applied to the GIS semantic queries. The tested data represented a Florida map, consisting of 32,108 segments. The page size for data was chosen to be 1024 bytes. Also, the real data from the GNIS (Geographic Names Information System) [26] was used in this study. This data has been compiled by the United States Geological Survey in conjunction with the US Board on Geographic Names (BGN) and contains information about nearly 2 million physical and cultural geographic features in the United States, its territories as well as Antarctica. GNIS data primarily consists of names and types of places along with associated coordinate point information. Each feature described in the database is identified by its federally recognized name,

and a feature's location is referenced by the state, county and geographic coordinates.

We compared a semantic *R*-tree with an *R*-tree that used the quadratic split algorithm. The quadratic split algorithm was chosen because there is no essential performance gain resulting from the linear split algorithm [3]. The performance metric used is the response times of the general queries and semantic queries. In the general queries, the degree of degeneration, if any, of a semantic *R*-tree compared to an *R*-tree can be obtained. For the semantic queries, experiments were conducted to show how semantic queries are answered using the semantic *R*-tree.

The sample range sizes of the rectangles were large size (200 miles) and small size (20 miles) in the GNIS file. The sample areas chosen were northwest (NW) region, northeast (NE) region, southwest (SW) region, southeast (SE) region, and center (CEN) region in the data set. The number of nodes tested ranged from 5000 to 30,000.

Figs. 10 and 11 show the experimental results of the general queries. The *X*-axis is the number of nodes. The *Y*-axis represents the relative response time, which is the ratio between (response time using an *R*-tree) and (response time using a semantic *R*-tree). If the result value is greater than 1.0, it means that the semantic *R*-tree outperforms the original *R*-tree, and vice versa, if the value is less than 1.0, it means that the original *R*-tree outperforms the semantic *R*-tree. For the queries with a large rectangle size, the semantic *R*-tree outperforms an *R*-tree for some of the queries. When an *R*-tree outperforms a semantic *R*-tree, most of results show that the relative response time is very close to 1.0, which supports our first observation in Section 4.2 (as shown in Fig. 10). As can be seen from Fig. 10, the response time ratio is very close to 1.0. In this case, SE and CEN regions are more category-dense relative to the large range size, so the result is very close to 1.0 and even is much better than the results from an *R*-

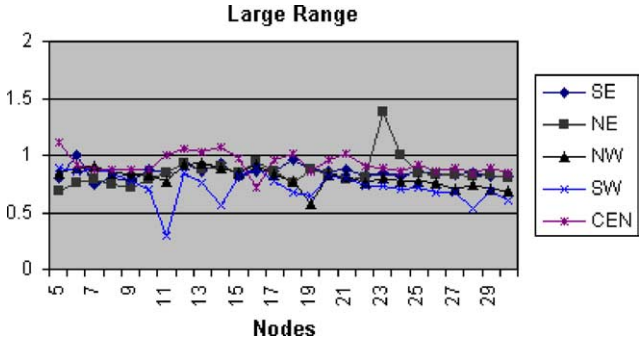

Fig. 10. Range query results with large size range.

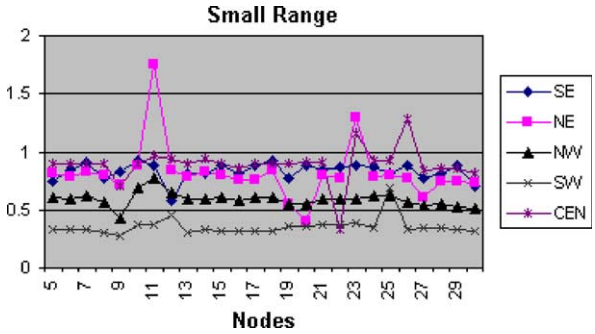

Fig. 11. Range query results with small size range.

tree. Similar experimental results are obtained for small rectangle size (as shown in Fig. 11).

The experimental results for semantic queries are shown in Fig. 12, where the X-axis is the number of nodes and the Y-axis represents the ratio between (response time using an R-tree) and (response time using a semantic R-tree). If the test value is greater than 1.0, then it means that the semantic R-tree outperforms the original R-tree, and vice versa. The results support observation 2 in Section 4.2, too. The semantic R-tree performs much better than the R-tree. If no objects are found for the queries, the relative response time may be higher than 1.0.

All these experimental results show that the semantic R-tree is a very efficient spatial data structure for answering semantic queries. As expected, the

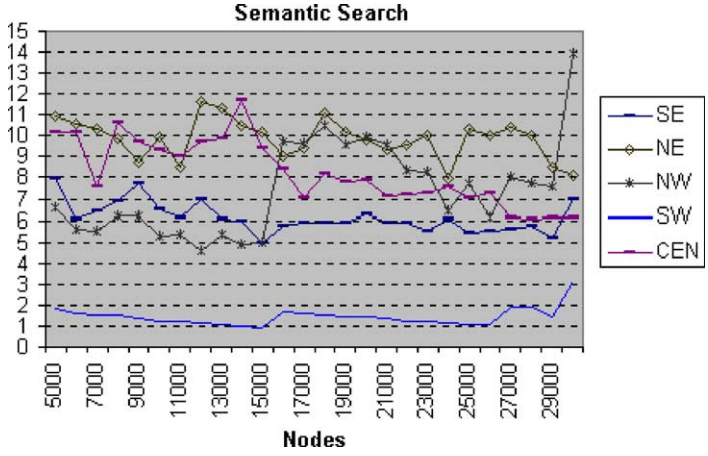

Fig. 12. Results of finding all objects with category "Bank".

performance gain of the semantic *R*-tree over an *R*-tree in answering semantic queries is considerably high. Since the semantic *R*-tree is a variant of the *R*-trees, both precision and recall values are very close to those of the *R*-trees. In many GIS applications, the support for semantic queries is more desirable since this kind of query typically corresponds to information that users request. Hence, a semantic *R*-tree is more suitable in such cases.

6. Conclusion

In this paper, a GIS system called TerraFly is studied. The TerraFly system is a multimedia application that allows the users to view images, manipulate the retrieved data, and issue range queries and nearest neighbor queries. A spatial data structure, the Semantic *R*-tree, is proposed to search for objects based on both the spatial and semantic information. A technique called "IDMT Model" is proposed to achieve better performance. A semantic object-oriented database management system is developed to meet the database requirements. Spatial data such as maps can be stored and retrieved from this database. Several experiments were conducted to compare the Semantic *R*-tree with the regular *R*-tree based on general queries and semantic queries. The experimental results show that the Semantic *R*-trees perform better than the *R*-trees for semantic queries, and have similar performance for general queries.

Acknowledgements

This research was supported in part by NASA (under grants NAGW-4080, NAG-5095, NAS5-97222, and NAG5-6830), NSF (EIA-0220562, CDA-9711582, IRI-9409661, HRD-9707076, and ANI-9876409), ARO (DAAH04-96-1-0049 and DAAH04-96-1-0278), AFRL (F30602-98-C-0037), BMDO (F49620-98-1-0130 and DAAH04-0024), DoI (CA-5280-4-9044), and State of Florida. Mark Allen Weiss was supported in part by NSF (EIA-9906600).

References

[1] N. Beckmann, H. Kriegel, R. Schneider, B. Seeger, The *R**-tree: An efficient and robust access method for points and rectangles, in: Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 322–331.

[2] S. Berchtold, D.A. Keim, H.-P. Kriegel, T. Seidl, Indexing the solution space: A new technique for nearest neighbor search in high-dimensional space, IEEE Transactions on Knowledge and Data Engineering 12 (1) (2000) 45–57.

[3] A. Guttman, *R*-tree: A dynamic index structure for spatial search, in: Proc. ACM SIGMOD, June 1984, pp. 47–57.
[4] I. Kamel, C. Faloutsos, On packing *R*-trees, in: ACM CIKM, 1993, pp. 490–499.
[5] S.T. Leutenegger, M.A. Lopez, The effect of buffering on the performance of *R*-trees, IEEE Transactions on Knowledge and Data Engineering 12 (1) (2000) 33–44.
[6] B. Lewis, D.J. Berg, Multithreaded Programming With Pthreads, Sun Microsystems Press, 1998.
[7] B. Nichols, D. Buttlar, J.P. Farrell, Pthreads Programming, O'Reilly & Associates, Inc., 1996.
[8] N. Rishe, A Semantic Approach to Database Design: The Semantic Modeling Approach, McCraw-Hill, 1992.
[9] N. Rishe, S.-C. Chen, et al., TerraFly: A high-performance web-based digital library system for spatial data access, in: The 17th IEEE International Conference on Data Engineering (ICDE), Heidelberg, Germany, April 2001, pp. 17–19.
[10] N. Rishe, J. Yuan, R. Athauda, X. Lu, X. Ma, A. Vaschillo, A. Shaposhnikov, D. Vasilevsky, S.-C. Chen, SemanticAccess: Semantic interface for querying databases, in: International Conference on Very Large Databases, Cairo, Egypt, September 2000, pp. 591–594.
[11] N. Rishe, A. Vaschillo, D. Vasilevsky, A. Shaposhnikov, S.-C. Chen, A benchmarking technique for DBMS's with advanced data models, in: ACM ADBIS-DASFAA Symposium on Advances in Databases and Information Systems, Prague, Czech Republic, September 2000, pp. 138–149.
[12] N. Rishe, W. Sun, et al., Florida International University, High Performance Database Research Center, SIGMOD Record 24 (3) (1995) 71–76.
[13] N. Roussopoulos, D. Leifker, Direct spatial search on pictorial database using packed *R*-trees, in: Proc. ACM SIGMOD, May 1985.
[14] N. Roussopoulos, C. Faloutsos, T. Sellis, Nearest neighbor queries, in: Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 71–79.
[15] T. Sellis, N. Roussopoulos, C. Faloutsos, The R^+-tree: A dynamic index for multi-dimensional objects, in: Proc. 13th Int'l. Conf. on Very Large Databases, 1987, pp. 507–518.
[16] Semantic Object-Oriented Database management System: C++ Interface Programmer's Guide, Ciotek Version, High Performance Database Research Center, Florida International University, Miami, FL, 1998.
[17] E. Tanin, F. Brabec, H. Samet, Remote access to large spatial databases, in: The Tenth ACM International Symposium on Advances in Geographic Information Systems, 2002, pp. 5–10.
[18] http://elib.cs.berkeley.edu/gis/index.html.
[19] http://gis.leica-geosystems.com/.
[20] http://terraserver.microsoft.com/default.aspx.
[21] http://www.envi-sw.com.
[22] http://www.esri.com.
[23] http://www.esri.com/software/arcgis/arcinfo/index.html.
[24] http://www.gvu.gatech.edu/gvu/virtual/VGIS/.
[25] Intergraph, Mapping and GIS solutions. Available from <http://www.intergraph.com/software>.
[26] USGS mapping Information: Geographic Names Information System (GNIS). Available from <http://mapping.usgs.gov/www/gnis/>.