

A Study on the Indexing of Satellite Images at NASA Regional Application Center

Chiara Ghirardini Soon Ae Chun, Vijay Atluri
University of Milan CIMIC-Rutgers University

Ibrahim Kamel
Panasonic Research Lab

Nabil R. Adam
CIMIC-Rutgers
University

chiara@cimic.rutgers.edu {soon,atluri}@cimic.rutgers.edu ibrahim@research.panasonic.com adam@adam.rutgers.edu

Abstract

This paper presents a performance analysis of indexing structures for the satellite image database. The unique features of this database are that the images vary in resolution level and time of capture, but cover the same geographic area. Each image belongs to one of the following types: raw, processed, vegetation, temperature, water, and color. We are particularly interested in comparing the space requirements, insertion and creation time, and query response time.

We have implemented a web-based information system for Satellite images using Oracle DBMS. We implemented different indexing structures on top of the system for our experiments. We found that SS-tree gives the best performance for our application.

1. Introduction

The NASA Regional Application Center (RAC) at the Center for Information Management, Integration and Connectivity (CIMIC) [12], Rutgers University, Newark is a joint project between CIMIC-Rutgers, NASA Goddard Space Flight Center (GSFC) and the Hackensack Meadowlands Development Commission (HMDC). As a RAC, CIMIC maintains a large collection of satellite images acquired through various sources.

The goal of this database is to promote timely access to satellite images to large user groups including scientists, urban planners, teachers and students for research, planning and education purposes. As our image database grows with daily downloads of satellite images and with acquisitions from multiple existing sources as well as from new generation satellites, search time became an imminent problem. For example, HMDC uses this database in a wide variety of application areas, including environmental monitoring, such as disaster (fire, flood) identification, urban planning and permitting processes and business decision making. These value added application products require efficient retrieval of satellite images, which undergo further processing.

Our satellite image database has the following unique characteristics: (a) *All images cover the same geographic area:* Our region of interest (ROI) is the HMDC area.

(b) *They have different resolution levels:* The database includes images of 4 different resolutions depending on the sensors: 'AVHRR' (NOAA 12, 14 and 15), 'Landsat', 'Radar' and 'Orthophoto'; (c) *Each image is of a distinct type:* 'raw' (i.e. the original downloaded format), 'processed' (i.e. a cut image or an image generated from appropriate combinations of its spectral bands) or 'vegetation', 'temperature', 'water' or 'color composites'; (e) *Each image is time stamped with date and time the capture time;* (f) *Typical user queries are based on retrieving raw bands; looking at the condition of water, vegetation or land usage; searching for images that fall into a particular time interval, etc.* Thus our satellite image comprises of the metadata comprising of the following attributes: level of resolution, type, and the download time and date. Since almost all of the images cover the same area, the spatial attributes of the image are not considered important.

This experimental study considers three indexing structures and compares their performance to find a suitable and efficient structure for our satellite image database. Particularly, our performance evaluation study analyzes the behavior of the following structures: R*-Tree, SS-Tree and SR-Tree. These index structures have been proven to be efficient for a wide variety of database and GIS applications.

The goal of our study and of this paper is to present benchmarking results for all the above structures and to draw conclusions about which would be the "best" structure for CIMIC's Satellite Image database. We have compared the three structures using data sets that reflect realistic distributions of our image database. Furthermore, each experiment is aimed at evaluating the efficiency of those queries that are most frequently asked by our user community.

The paper is structured as follows. Section 2 briefly describes the index structures we considered in this study. Section 3 describes our application environment. The experiment setup and the results are presented in Section 4. Section 5 provides some discussion on our findings. Section 6 provides conclusions and outlines our future work.

2. Multi-dimensional Indexes

Several multi-dimensional indexes have been proposed over the years, e.g., Quadrees[4], K-D-B-Tree [8], R-tree[2], SS-tree[11], SR-tree[6]. One of the most successful index is the R-tree which originally proposed by Guttman [2]. Several variants have been proposed like R+-Tree [9], Packed R-tree[5], R*-tree[1], Hilbert R-tree[3]. R-tree is a hierarchy of nested d-dimensional rectangles (also called minimum bounding regions, MBR). The index is completely dynamic and allows overlapping directory rectangles. R*-tree implements the concept of deferred splitting to achieve higher space utilization. When a node or leaf is full, R*-tree reinserts a portion of its entries rather than splitting it. It is likely that split is not needed, unless reinsertion has been made on the same tree level. SS-tree and SR-tree also use the concept of forced reinsert.

The SS-tree is a structure designed for similarity indexing of multidimensional point data. It is similar to the R-tree, however with a striking difference. Instead of minimum bounding rectangles, it uses minimum bounding spheres (MBS), whose center is the centroid of the points contained in a given subtree. SS-tree modifies the forced reinsertion mechanism of R*-tree. SS-tree reinserts entries unless reinsertion has been made at the same node or leaf.

The SR-tree is a structure based on that of R*-tree and SS-tree. However, its distinctive feature is that it specifies a region by the intersection of the bounding sphere and bounding rectangle of underlying points. By making use of both spheres and rectangles, SR-tree leads to less overlap between the indexed regions but is most costly to build.

Space requirement for the index is critical for our application because of the size and number of images added everyday. This is why the “forced reinsert” feature is useful. In this paper we are comparing the performance of SS-Tree [7, 8] and SR-Tree [3] with R-tree. We chose R*-tree to represent the R-tree group because it supports the “forced reinsert” feature.

3. Application Requirements

Recall that all the images in the CIMIC’s database cover the same geographical area; typical user submits queries using a combination of the resolution, type and timestamp attributes of the image rather than the spatial location. In fact, a range query that searches the database using a particular region that covers Hackensack District would probably return the whole database.

By taking into consideration the type of queries that are of interest, we have come to the conclusion that the best way to represent the images of CIMIC’s database is by transforming each image to a point in the parametric space. This space is composed of the resolution, type, and timestamp attributes of the images in the native space. The

three attributes chosen to represent the images are selective enough to narrow down the search and consequently to obtain the desired set of images in a reasonable time. The types of queries considered for the study are:

Exact match query. The user defines a specific value for all three attributes. All the images that match the specified values are returned.

Time range query. The user defines the type and resolution of the images acquired during a time. All the images that match the first two values and fall into the time interval are returned.

N Nearest neighbor query. The user defines a specific value for all three attributes and a number N. All $k, k \leq N$, images that match the specified values and N-k most similar images are returned.

The benchmarking results have been obtained using a dataset of 200,000 3-dimensional points. This dataset was generated so as to maintain the same distribution of the satellite images contained in CIMIC’s database as well as the same temporal ingest patterns.

In order to provide a fair comparison among the structures we used the implementation made available by GiST (Generalized Search Tree) [13], libgist v.2.0, which includes code for R*-tree, SS-tree and SR-Trees.

We ran our experiments on a dedicated Sun Ultra 10 Workstation with 640 MB of RAM, 10 GB of hard disk space and a Unix Solaris 2.7 operating system. The physical disk page size in the system is 8192 bytes.

We compared the three indexing structures by analyzing the results obtained from various experiments, based on *properties of the trees* (space requirement and number of nodes); *time* (creation time, insertion time and exact match query time); *number of I/Os* (exact match query I/Os and range query I/Os). In order to study the effect of the page size on the performance of the index, we used page sizes of 2, 4 and 8 KB.

4. Performance Results

(a) **Properties of the Trees:** The space requirement, i.e. the amount of memory, in MB, required to store the index, is shown in Figure 1. The size of the index in terms of the number of nodes is shown in Figure 2.

For disk page size of 2 KB, the R*-tree requires less space and has fewer nodes than SS-tree and SR-tree. This means that for page size 2 KB R*-tree is more packed and has higher node utilization. On the contrary, for page sizes of 4 and 8 KB, SS-tree and SR-tree require less space and have fewer nodes than R*-tree. SS-tree and SR-tree have their minimum values at 4 KB disk page size, i.e. the two lines representing SS-tree and SR-tree do not show linear growth. This dip shows that the combination of the number of nodes and the node size is optimal for those trees.

Moving from 2 KB to 4 KB, the number of nodes reduces to less than half, resulting in lower space requirements, while moving from 4 KB to 8 KB, the number of nodes is more than half, resulting higher space requirements. These results show that with 2 KB page size gives minimum space requirements and maximum page utilization. The next best choices in terms of the space requirements are the SS-tree and SR-tree with 4 KB page sizes respectively. It is clear that we should avoid using R*-tree with 8 KB page size since it gives the worst space utilization.

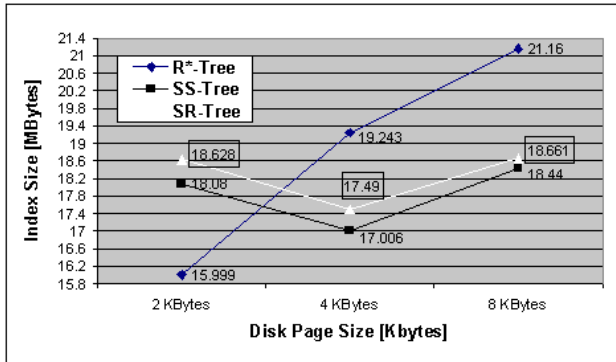


Figure 1: Index Size versus Disk Page Size

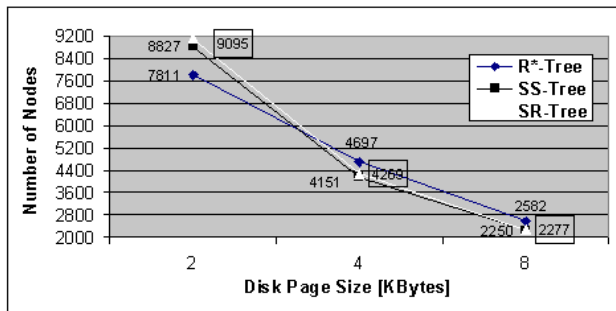


Figure 2: Number of Nodes versus Disk Page Size

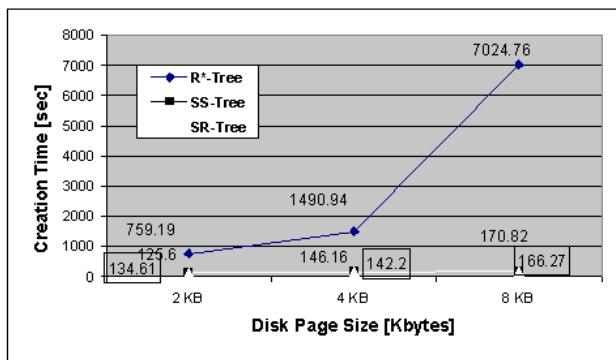


Figure 3: Creation Time versus Disk Page Size

(b) **Time:** The time required to create the index, add a new record to the index and execute an exact match query is measured and shown in figures 3, 4 and 5, respectively.

(b1) **Creation Time:** With “creation time” we refer to the actual CPU processing time required to insert all data points into each index. The creation time takes into account the time needed to perform several operations, such as finding the right node in which to insert the new record, performing all the necessary node splits and possible rearrangements of the tree structure to preserve its original properties.

All three indexing structures were built by inserting a set of 200,000 images one record at a time. The insertion order was based on the date of acquisition of the image (ordered by ascending date).

As shown by the graph of Figure 3, the creation time required for SS-tree and SR-tree is almost the same regardless of the page size. On the other hand, the creation time required by R*-tree is much higher especially with larger page size.

As we observed earlier, R*-tree requires more nodes than the other two structures for page sized of 8 KB. This means that R*-tree will need to perform more node splits and structure rearrangements, which are very costly operations for R*-tree.

(b2) **Insertion Time:** Insertion time is defined as the actual CPU processing time required to add a new record to the index structure. Having measured an average nodes utilization of 70%, the insertion of the extra record was guaranteed not to cause any node splits. This means that the insertion time refers to the insertion operation only and does not include node splits or rearrangements of the structure. Figure 4 shows that SS-tree and SR-tree are faster than R*-tree, as expected from the results on creation time. The creation time (total insertion time of 200,000 points) can be viewed as the sum of the insertion time required by every point.

By averaging the creation time over the total number of points (creation time of one tree/200,000) we notice that this average insertion time of a point (35 msec for R*-tree, 0.85 msec for SS-tree and 0.83 msec for SR-tree) is higher than the actual single point insertion time.

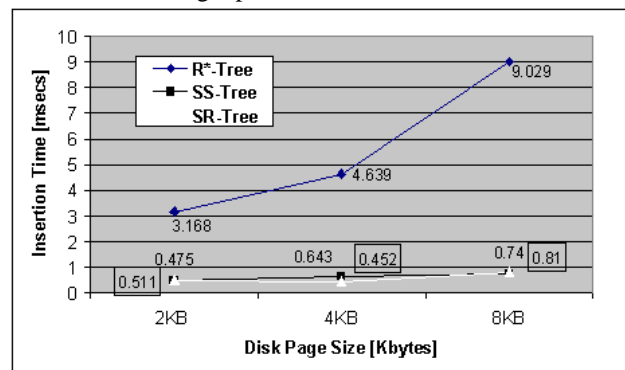


Figure 4: Insertion Time versus Disk Page Size

This is due to the fact that the creation time takes into account the time required to perform all node splits. Thus the average insertion time consists of the actual single point insertion time plus the average split time. This means that the difference between average insertion time and actual insertion time corresponds to the average split time, 25.97 msec for R*-tree, 0.11 msec for SS-tree, and 0.02 msec for SR-tree. As shown by these numbers, the node split algorithm of R*-tree is slower than that of SS-tree and SR-tree. These results confirm what was already expected, i.e. that the linear split algorithms of SS-tree and SR-tree are much more efficient than the quadratic ones implemented in R*-tree.

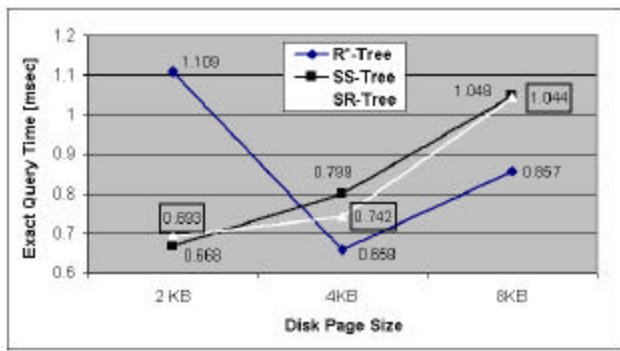


Figure 5: Exact Query Time versus Disk Page Size

(b3) Exact Match Query Time: The query time refers to the time required to search and retrieve the record that matches the specified query point. We ran 5000 exact match queries and averaged the response time. Of the 5000 points, 80% were actual data points (i.e., taken from the data file) while 20% were generated randomly. The results shown in Figure 5 indicate that the exact query time is strongly dependent on the properties of the trees, especially on the node utilization.

For disk page size 2 KB, the response time for exact match retrieval in R*-tree is much higher than those for SS-tree and SR-tree. This is in part because the R*-tree structure has less nodes than SS-tree and SR-tree. R*-tree has a higher node utilization, which means that every node contains more records and therefore describes a larger region, which leads to highly overlapping regions. The exact match query will have to follow multiple paths and will require more time to retrieve the specified query point. Similarly for disk page sizes of 4 KB and 8 KB the higher node utilization of SS-tree and SR-tree leads to higher response time for the two indexing structures.

(b4) Nearest Neighbor Query Time: The query time refers to the time required to search and retrieve 20 records that were most similar to the specified query point. We ran 5000 nearest neighbor queries and averaged the response time. Even though the differences in the response times were not so significant, the results still showed that SS-tree outperforms the other two structures. The results

graphs are not shown for lack of space.

(c) Number of I/Os: The number of I/Os indicates the total number of I/Os, i.e. internal nodes and leaf nodes retrieval.

(c1) Exact Match Query I/Os: We ran 10,000 exact match queries. Recall that the set of points used to run exact match queries were split to have 80% actual data points and 20% randomly generated data. Out of the 10,000 exact match queries we got 8320 hits. The results shown in Figure 6 indicate that for disk page size of 2 KB, R*-tree requires twice the number of I/Os of SS-tree and SR-tree due to the high node utilization of R*-tree. The higher node utilization leads to higher overlapping among the regions described by each node and, consequently, increases in the number of paths that will be traversed by the search algorithms. On the other hand for disk page sizes of 4 and 8 KB, R*-tree, which has the lowest node utilization, needs similar or even less I/Os than the other two structures.

These results are consistent with those shown in Figure 5. In fact, the higher/lower number of I/Os performed by one indexing structure corresponds to a higher/lower retrieving time.

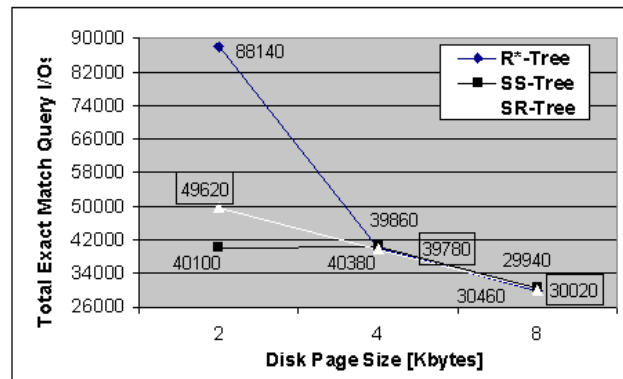


Figure 6: Total Exact Query I/Os versus Disk Page Size

(c2) Range Query I/Os: For every disk page size, we ran 100 range queries varying the range size of the date of acquisition to 3, 16 and 96 months. Independently of the disk page size, the set of 100 range queries retrieved: 9219 points, 48419 points and 292251 points, respectively for 3, 16 and 96 months ranges. The results are shown in the graphs of Figure 7, 8 and 9, which correspond to disk page sizes of 2, 4 and 8 KB respectively.

Unlike exact match queries, range queries are performed more efficiently (i.e. require a lower number of I/Os) by the structure that has higher node utilization (more packed data). In Figure 7 shown for page size of 2 KB, R*-tree minimizes the number of I/Os. R*-tree with the highest node utilization for 2 KB requires less nodes for retrieving. On the other hand, for disk page sizes of 4 and 8 KB, SS-tree and SR-tree which have higher node

utilization perform better than R*-tree as shown in Figures 8 and 9.

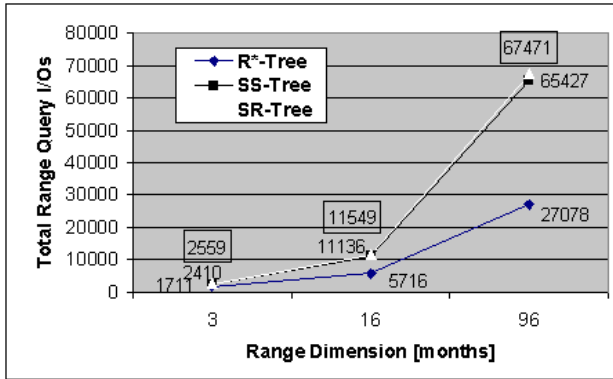


Figure 7: Total Range Query I/Os versus Range Dimension – Disk Page Size = 2 KB

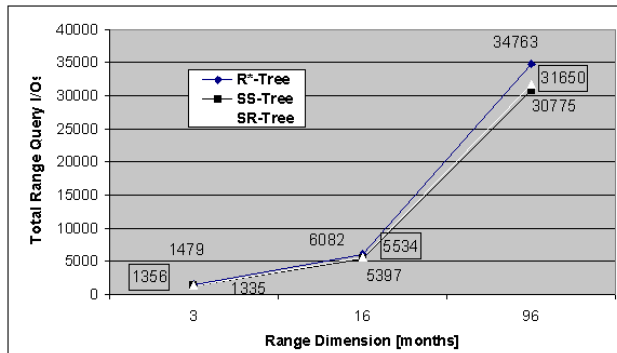


Figure 8: Total Range Query I/Os versus Range Dimension – Disk Page Size = 4 KB

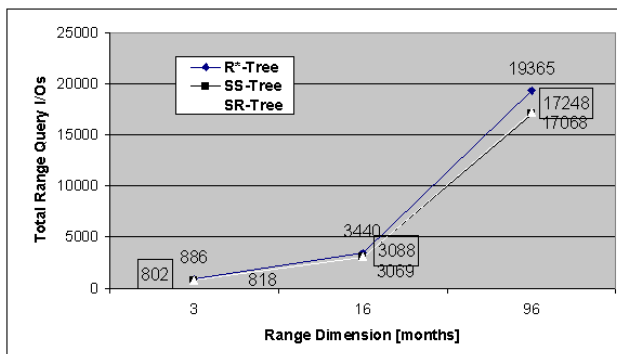


Figure 9: Total Range Query I/Os versus Range Dimension – Disk Page Size = 8 KB

	Index Size	Number of Nodes	Creation Time	Insertion Time	Exact Query Time	Exact Query I/Os	Range Query I/Os
2 KB	R*	R	SS	SS	SS	SS	R*
4 KB	SS	SS	SS(SR)	SS(SR)	R*	SR(R*)	SS
8 KB	SS	SS	SS(SR)	SS	R*	R*	SS

Figure 10: Performance evaluation results

5. Discussion

Figure 10 shows a summary of our findings in the previous Section. The row refers to the disk page size (2, 4 and 8 KB); each column refers to the performance criteria used in the experiment. Each cell indicates the structure that gives the best performance for particular measure and page size. (Note: the second best performing structure has been indicated in parenthesis when its results turned out to be very similar to those of the first/most efficient structure.)

Given the application requirements of our system at CIMIC (disk page sizes of 8 KB, minimizing the size of the index, minimizing the insertion time and the range query response time), we found SS-tree is our best choice.

We have implemented the web-based image retrieval system with SS-Tree index and the Oracle based image database. The user queries are sent first to the SS-tree where it identifies relevant image ids. These retrieved image ids are used for Oracle database image retrieval. We also conducted an experiment to compare the query time with and without the use of SS-tree based on 1,000,000 records. The retrieval time with SS-tree was significantly better than that of sequential search. Exact match queries with SS-tree took 137.3853 msec while the sequential search took 19311.49 msec. Range queries with SS-tree took 19899.51 msec while the sequential search took 34081.24 msec.

6. Conclusion and Future Work

In this study, we have shown experimental results conducted on a satellite image database using R*-tree, SS-tree, and SR-tree index structures. Based on the query types and their processing time, we have identified SS-tree structure most efficient for our needs. We showed that SS-tree has a superior retrieval time for the case of the exact match and range queries.

Our future work includes comparing other operation like spatial join.

Acknowledgement

This work was supported in part by Meadowlands Environmental Research Institute from the Hackensack Meadowlands Development Center, and in part by the University of Milan thesis grant. We acknowledge Dr. Francisco Artigas, CIMIC-Rutgers University, for information on satellite imagery and discussion on user requirements, Dr. Richard Holowczak, City University of New York for his guidance in Oracle satellite image database.

References

[1] N. Beckmann, H. P. Kriegel, R. Schneider, B. Seeger. The R*-Tree: An Efficient and Robust Access Method For Points and Rectangles. In *Proceedings of the ACM-SIGMOD International*

Conference on Management of Data, pages 322-331, Atlantic City, NJ, May 1990.

[2] A. Guttman. R-Trees: A Dynamic Index Structure For Spatial Searching. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 47-57, Boston, MA, June 1984.

[3] I. Kamel, C. Faloutsos. Hilbert R-tree: An Improved R-tree using Space Filling Curve. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 500-509, Santiago, Chile 1994.

[4] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1989.

[5] I. Kamel, C. Faloutsos. On Packing R-trees. In *Proceedings of 2nd International Conference on Information and Knowledge Management*, pages 490-499, Washington, November 1993.

[6] N. Katayama, S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 369-380, Tucson, AZ, May 1997.

[7] M. Nascimento, N. Colossi. Benchmarking Access Structures for High-Dimensional Multimedia Data. In *IEEE International Conference on Multimedia and Expo (ICME'2000)*, New York, USA, July 2000.

[8] J. T. Robinson. The K-D-B-Tree: a Search Structure for Large Multidimensional Dynamic Indexes. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pages 10-18, Ann Arbor, MI, April 1981.

[9] T. K. Sellis, N. Roussopoulos, C. Faloutsos. The R⁺-Tree: A Dynamic Index for Multidimensional Objects. In *Proceedings of the 13rd International Conference on Very Large Data Bases*, pages 507-518, Brighton, England, September 1987.

[10] D. A. White, R. Jain. Similarity Indexing: Algorithms and Performance. In *Proceedings of the SPIE*, Vol. 2670, pages 62-73, San Diego, CO, Jan. 1996.

[11] D. A. White, R. Jain. Similarity Indexing with the SS-tree. In *IEEE International Conference on Data Engineering*, pages 516-523, New Orleans, LA, Feb. 1996.

[12] CIMIC Home Page – <http://cimic.rutgers.edu/>

[13] GiST Home Page – <http://gist.cs.berkeley.edu/>