



Spatially-Aware Information Retrieval on the Internet

SPIRIT is funded by EU IST Programme
Contract Number: IST-2001-35047



Report on Spatial Indexing Methods

Deliverable number:	D12 2201
Deliverable type:	R
Deliverable nature:	PU
Contributing WP:	WP 2
Contractual date of delivery:	Jan 1, 2004
Actual date of delivery:	Dec 31, 2003
Authors:	Subodh Vaid and Christopher B. Jones
Keywords:	Indexing, Information Retrieval, Spatial Access Methods

Abstract: The design of efficient indexing structures is central to the performances of very large databases such as the web collections. Within the SPIRIT framework the challenge is to adapt ideas from the well-researched topics of text based information retrieval and geographical information systems for a spatially aware search engine. This document reports on the existing methods for both these areas and proposes a hybrid scheme for indexing and retrieving text documents.

Contents

1.	INTRODUCTION	5
2.	BACKGROUND	5
2.1.	Search Engine Architecture.....	5
2.1.1.	Architecture	6
2.1.2.	The Indexer Module.....	6
2.2.	Indexing Techniques	7
2.2.1.	Database Indexing Techniques	7
2.2.2.	Spatial Indexing Techniques.....	8
2.3.	Text-Based Information Retrieval.....	10
2.3.1.	Full-text scanning	10
2.3.2.	Signature Files.....	10
2.3.3.	Inverted Files.....	10
2.3.4.	Vector Space Model	11
2.3.5.	Latent Semantic Indexing (LSI).....	11
3.	RELATED WORK	11
4.	SPATIAL SEARCH ENGINE DESIGN	13
4.1.	The Spatial Perspective.....	13
4.2.	Query structure.....	14
4.3.	Document Footprints	15
5.	INDEXING OPTIONS	15
5.1.	Problem Definition.....	16
5.2.	Solution Approaches.....	16
6.	COST MODELS	17
6.1.	Spatial Cost.....	17
6.2.	Spatio-Textual Cost	18
6.2.1.	Pure Textual (PT)	19
6.2.2.	Text followed by Spatial (TS).....	19
6.2.3.	Spatial followed by Text (ST).....	21
6.2.4.	Spatio-Textual Index (SP).....	22
6.3.	Efficiency of Index types.....	22
7.	EXPERIMENTS.....	23

7.1. Data Generation	23
7.2. Discussion of Results	25
8. CONCLUSION.....	34
9. ACKNOWLEDGMENTS.....	35
10. REFERENCES	35

Executive Summary

The uniqueness of the SPIRIT project is its capability of handling spatial queries for a collection of web documents. This report looks at several design options for implementing the document indexing component of a spatially aware search engine. Starting with a review of popular methods for textual and spatial data, the discussion proceeds to the design issues in combining both for handling spatial queries. Here, the spatial perspective of the searching system is brought into focus and the structure of spatial queries and the concept of a document footprint are discussed. Having laid down basic assumptions for the queries and the spatially tagged documents possible indexing systems are discussed next.

In Section 6, some theoretical models are presented for estimating the behaviour of our intended searching system. This indicates that though spatial indexing introduces large storage overheads relative to pure textual indexing, it can be expected to be advantageous with regard to access times when compared to the alternative of a spatial filtering stage following pure text retrieval. Section 7 describes experiments with synthetic collections of spatial documents where it is revealed that in practice the results for spatial indexing are very promising with lower storage requirements than expected and fast access times. The document concludes by suggesting future work with real data using advanced spatial indices.

D12 2201

Spatially Indexing Text Documents

1. INTRODUCTION

The World Wide Web (Web) has become a popular resource for finding desired information. However, finding the right piece of information quickly is generally not that straightforward. This is primarily due to three main factors: very fast web growth rate; heterogeneity of information available and the user expectations from web based search. It is estimated that the present size of the web is in excess of 3 billion web pages [1], many of which have embedded images, audio and video objects in addition to popular formats for storing data and text. Mining this plethora of web based data has led to the emergence of a variety of search tools catering to diverse user needs. Examples of these are SciNet (Science), GeoIndex (Geology), iCrank (Mechanical Engineering) etc. These specialized search tools in a number of cases can be used as an alternative for the popular search engines such as Google, Lycos, AltaVista etc. On similar lines, the SPIRIT project undertakes this initiative of studying the design of a spatially aware search engine in which queries of the form find a set of things somewhere can be efficiently handled [2]. Using 1Tb of web data as a test bed, this research project will attempt to demonstrate that it is possible to effectively meet the requirement of spatial queries by devising specific retrieval techniques for textual data. This paper discusses the design issues for development of a spatially aware search engine with specific focus on spatially indexing text documents. The paper is organized as follows. In Section 2, we review the subject areas and the related techniques, which overlap, with our design objective. The motivation for this section is to introduce the reader to the inter-disciplinary diversity of this work. Section 3 follows this where similar and related efforts are discussed. In the remainder of this document, we discuss our proposal for a spatio-textual indexing scheme supported by theoretical models and experiments. Finally, conclusions and future work are discussed in Section 8.

2. BACKGROUND

Search engines are web-based tools, which allow users to search for documents on the Web. The search is performed on a database of web documents and the criteria of selection are based on the terms specified in the query. Ensuring that only relevant pages are returned is accomplished by using specialized algorithms falling in the domain of information retrieval. Information retrieval as applied in search engines or databases and organization of spatial data are widely researched topics. In order to discuss the design of a spatial search engine, we briefly discuss its basic building blocks and related concepts.

2.1. Search Engine Architecture

Broadly speaking two types of search methods are generally used in information retrieval over the web: Web Directories and Search Engines. Web Directories such as Yahoo and LookSmart rely on human editors to classify Web sites under various subject headings. The number of pages returned in response to a direct search may be much smaller in comparison to an output of a search engine but the results are aimed to be more specific. On the other

hand, search engines maintain an index or a catalog of Web sites in a massive database that the user can search by entering keywords. Generally a very large number of pages are returned to the user possibly ranked in the perceived order of relevance to the user. Some examples of search engines are Alta Vista, Excite and Google. We focus our attention mainly on search engines and briefly describe their essential components so as to lay foundation for the remaining sections of this document. For details on working of search engines, the reader is referred to other documents such as [3], [4] and [5].

2.1.1. Architecture

A query in a search engine is not performed directly on the web but on a pre compiled selection of documents gathered by specialist programs called spiders or web crawlers. After spiders find pages, they pass them on to another special computer program called indexer for indexing. The indexer identifies the text, links, and other content in the page and stores it in the search engine database's files. The final part of the search engine software is the relevance ranker, which attempts to rank results on the basis of information specified in the query and presents the top few to the user. Figure 1 shows the basic architecture of a typical search engine. The Data Collection Module using web crawling prepares the database or the document collection. The indexer prepares an index from the database. Both these activities are performed prior to a query session. In order to search through an index the user needs to build a query and submit it to the search engine. It can be seen that the user interacts with the search engine through the User Interface. The query can be quite simple, a single word at minimum. Complex queries can be constructed which require the use of Boolean operators (AND, OR, NOT etc.) to refine and extend the terms of the search. The query is received in the Retrieval Module by the indexer, which retrieves the relevant documents from the database and passes back the ranked results to the user interface. We will be mainly interested in the indexer module in the discussion to follow. In the next section, we briefly discuss some features of a typical indexing module.

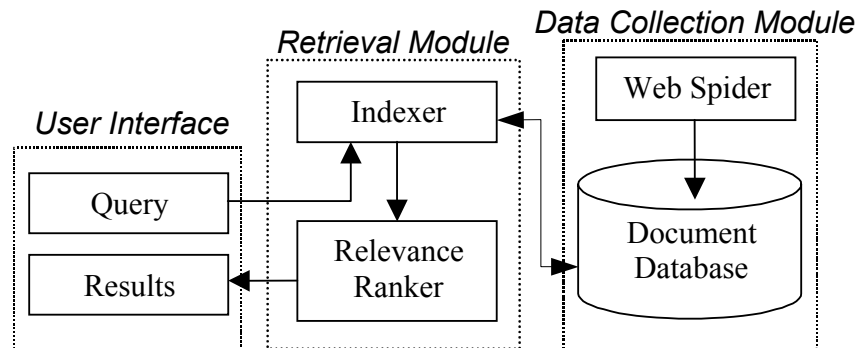


Figure 1. Components of a Search Engine

2.1.2. The Indexer Module

After the web crawling process has been completed, the information must be stored in an organized manner for efficient retrieval. It may be noted that in practice, the task of web crawling is never actually completed. In order to cope with Web dynamics and to tap the business opportunities thus created, search engines keep updating their databases. In managing these huge repositories of data (billions of web pages), the emphasis is on two key factors:

- The content of the web page
- The method used to index web pages

As the retrieval of documents is sensitive to the words specified in the query (query terms) a simple search procedure can be to store the word and the document names on which it was found for the entire collection. In practice such a search engine would be of limited use, as

merely finding a word on a page in no way reflects the relevance of the page as regard to given query. What is desirable is to know that: what is the importance of the word in describing the document page, how many times was it used on the page or whether the page contained links to other pages containing the word. Answers to above questions will surely help in deciding the rank of a document in the list of search results, which is not possible by the simple search procedure discussed above.

In order to arrive at a meaningful set of results, most search engines store more than just the word and URL. The emphasis is on storing the weight (importance) of the word which can be found by looking at its frequency, position(s) within the document, font, appearance in title, links or in meta tags. Each commercial search engine has a different formula for assigning weight to the words in its index. This is why for the same query different search engines produce different lists, with the pages presented in different orders.

While it is important to store as much information as possible about a word, the storage requirements in building the index can not be ignored. It is common for search engines to make use of compression techniques to store a large part of the index in the main memory. For example, in [39] Google report storing their lexicon comprising of 14 million words on a machine with 256 MB of main memory. This was possible by allocating just 2 bytes per word to store information on weighting – whether the word was capitalized, its font size, position, and other information to help in ranking the hit. If each individual piece of information is restricted to 2 or 3 bits within the 2-byte grouping, then it is possible to store a large amount of information in a very compact form. After the information is compacted, the next step is the building of an index for the collection.

The prime task of an index is to facilitate efficient retrieval of desired information. For example, the index at the end of a book contains a list of important words and the page numbers on which they appear for quick reference. A number of indexing techniques or access methods are employed in retrieving different types of information and are the subject of discussion of the next section. It would be appropriate to mention here that for accessing search engine databases, hash tables and inverted files are generally used.

2.2. Indexing Techniques

An index is a data structure that supports fast retrieval of information. Indexing plays an important role when an *item of interest* is to be retrieved from a (very) large collection of items (the database of web documents in our case). Specialized index structures are required for different database applications. There is an abundance of literature on Indexing methods or algorithms. We give below a brief review of indexing techniques that were studied here before undertaking the design of the spatial search engine. The overall objective of this review is to bring out the salient features of indexing methods employed in various search/application domains. We chose to categorize the review into conventional database (one dimensional) and spatial (multidimensional) domains, as it would focus better on the overall design objective.

2.2.1. Database Indexing Techniques

Database applications rely heavily on indexing techniques to increase the efficiency of retrievals. In particular, indexing ensures fast data retrieval by providing efficient address calculation of data in secondary storage based on the values of one or more attributes which uniquely specify data entities (records), called “keys”. Retrievals become much faster compared to those corresponding to an exhaustive file search. We reproduce some of the ideas generally covered in standard texts on algorithms or database concepts such as [6], [7], [8] and [9].

Indexing techniques fall into two broad categories: those making use of tables of tuple entries and those that employ tree structured indices. If table size becomes very large, then the table searching is extended to another level and this form of indexing is called hashing. A serious drawback with hashing schemes is that keys are stored unordered, thus making the treatment of “range queries” inefficient. In this case, all records having keys within a certain range of

values must be retrieved. Techniques capable of treating such queries are discussed in Section 2.2.

Tree based searching methods achieve logarithmic access times and are a common choice when the data collection is so large that the complete index can not be loaded into the main memory and is partially stored on the disks. Based on the binary search, binary trees were one of the first search structures to be studied under trees. A binary tree is either the empty tree or a node that has left and right subtrees that are binary trees. A common problem with all tree based structures is that with the advent of time, when the tree size grows its structure becomes unbalanced and requires balancing for effective retrieval. This observation led to the research of balanced trees such as AVL trees, 2-3 trees, red-black trees etc. However, in this category, the “B-tree” is one of the first and most important tree-indexing techniques proposed so far for disk accesses [14]. B-trees and its variants such as B+ trees have become a standard and a popular choice in a large number of database systems. B-trees ensure fast access times by allowing high “branching ratios” (i.e., up to a large number of entries can be stored in each node of a B-tree). Therefore, tree height is usually very small, which reduces the number of disk accesses needed for a query. A B-tree adapts its storage space to the amount of stored data and always remains balanced, that is the length of each path from the root to each leaf-node (and therefore the access time to each leaf-node) is the same.

The techniques discussed so far perform on single keys. However, techniques that perform on multiple keys also exist. These are discussed in the next section.

2.2.2. Spatial Indexing Techniques

Spatial access methods are generally studied under the category of multidimensional access methods. An excellent survey on this subject can be found in [21]. For multidimensional data, point, range and interval queries are common and for the scope of the SPIRIT project we focus primarily on region based queries i.e. finding all objects that overlap a given search region. Spatial indexing techniques can be divided into two broad categories: those based on grid files or “space filling curves” and those based on tree index structures (R tree and its variants). With the sole exception of the grid file technique [15], hashing techniques are not particularly well suited for region indexing and range queries since they store keys unordered. Tree indices avoid this problem, but result in slower retrieval times.

We use the classification of spatial access methods as described by Rigaux et al. in [12]. In this classification, spatial access methods are categorized as space driven and data driven structures. While the space driven structures are based on partitioning of the two-dimensional space into rectangular cells, independent of distribution of objects the data driven structures take the approach of partitioning of objects as opposed to space. We begin the discussion by describing the *fixed grid* data structure in which the search space is decomposed into equal sized rectangular cells. A two-dimensional array called the *directory* is then maintained to map grid cells to array elements. Using this directory index it is possible to know what objects are contained in which cell. When newer objects are added to a cell then either they are stored directly in the directory page corresponding to the cell or an overflow occurs. In this case a chain is maintained by linking to the initial page. However, this scheme can prove expensive for frequently occurring overflows for non-uniform distributions. This is overcome by the *grid-file structure* in which the cell is divided instead of adding an overflow chain [15]. A quadtree decomposes the search space into quadrants until the number of objects overlapping each quadrant is less than the page capacity. The quadrants are named North West (NW), North East (NE), South West (SW) and south East (SE). The index is represented as a quaternary tree (each internal node having four children, one per quadrant). Each leaf is associated with a disk page as in the case of the grid file. However, both these approaches suffer from the same problem – the duplication of the objects in the neighbouring cells when objects span more than once cell. This can have a serious effect on the query time if for retrieving some object available on multiple leaves requires accessing internal nodes that requires different pages to be loaded into the memory. The limitations of a quaternary tree can be overcome by employing a B-tree to maintain an index of quadtree leaves uniquely identified by location codes. These codes are typically based on *space-filling curves* [16], which preserve the proximity of neighbouring points. This approach is referred to as a *linear quadtree*.

Space filling curves can also be used on their own to reduce dimensionality of the search space. Space filling curves such as the *z-ordering* and the *Hilbert curve* can provide a linear ordering of multi-dimensional points that preserves spatial proximity. As in the case of grid based methods, the search space is represented by grid cells labeled according to the *z-ordering* (say). The data object is represented by a set of cells called *Peano regions* or *z-regions*, where each Peano region corresponds to a unique bit string code. Using those bit strings which describe the data object, the cells can be stored as a standard one-dimensional index, such as the B+tree. The number of disk accesses (and thus response time) required by range queries is then minimum and depends on how well the mapping which has been used preserves the distances in the one dimension.

Among the data driven category of spatial access methods, the majority of methods reported belong to the R-tree family, discussed initially in [10]. Over the years, a number of R-tree variants have been proposed, we discuss the R-tree structure (in two dimension) as representative of the family in describing the underlying concept, which can then be extended for other variants. R-trees behave much like B-trees. Their structure adapts itself to the distribution of rectangles (generally representative of a two-dimensional interval) in the two-dimensional plane. Similar to the B-tree, a R-tree is a depth balanced hierarchical structure, in which each node (internal or leaf) corresponds to a disk page. However, unlike B-trees that are based on ordering of single key values, R-trees organize rectangles according to a containment relationship. Every node in the tree corresponds to a rectangle, generally called the *directory rectangle*, which is the minimum bounding box of the rectangles of its child nodes. Again searching in the R-tree is similar to the B-tree. It begins at the root and all those children of the root are visited whose directory rectangle intersects or contains the search region specified in the query. The process is repeated at each level of the tree until the leaves are reached. Since the directory rectangles of the internal nodes may overlap, several paths from the root to the leaves may have to be traversed. Another reason for traversing more paths is that the search is done with a region and not a single point. In the worst case, answering a query may require visiting all the pages of the index. As traversing of multiple paths is related to overlapping of directory rectangles, this fact gave rise to a number of R-tree variants that intend to minimize this overlapping. One suggestion is to use clipping to avoid overlap of directory rectangles at a level. This variant is called the R+tree [13], in which the objects that intersect more than one directory rectangle at a particular level are clipped and stored on several different pages. While this scheme reduces the search to only one path, it has to cope with added storage requirements. Also, there can be cases when it is not possible to avoid overlap. Another variant of R-tree is the R*-tree [11], which provides several improvements to the insertion algorithm of the basic structure by introducing a special policy called *forced reinsert*. In this case if a node overflows, it is not split right away. A predefined number of entries are removed from the node and inserted back to the tree. Whenever a split is necessary, it is done by looking at all possible distributions of the objects on either side of the chosen vertical (or horizontal) split line.

While a number of variants of the R-tree have been proposed which claim better access times due to their insertion algorithms, it is difficult to assume in such methods that over the period of time optimal space and access organization of the structure will be maintained. Over time multiple insertions and deletions may lead to degradation of the performance. However, when the collection of rectangles is stable with time, it is possible to pre-process the data before creating the R-tree. Several algorithms called *packing algorithms* have been proposed for the R-tree. They build on the conjecture that if the dataset to be indexed is known beforehand, then it can be sorted for constructing an index bottom up. In the case of R-tree, the bounding rectangles of the data objects are sorted by their location. A simple algorithm for constructing a R-tree is the Sort-Tile-Recursive (STR) algorithm. In this algorithm the centroids of the rectangles are sorted first on their x-co-ordinates and then divided into P groups to get P partitions. Starting from left to right, for each partition P, rectangles are sorted in increasing order of their y-co-ordinates and then assigned to the leaves of the R-tree, by assigning first M to each node. Once the leaves are known, it is possible to calculate the directory rectangles for the nodes by calculating minimum bounding boxes, which can then be used for construction of the next level until we reach the root or one rectangle. This method seems to be appropriate for

constructing the R-trees as it closely resembles our data, a collection of web pages having rectangular foot-prints. However, if we were to select just one access method then the decision can be slightly tricky, as it is difficult to know how well would it suit our application. This is answered in Gaede and Guenther [21], who state that it is difficult to compare and rank different access methods as “*experimental benchmarks need to be studied with care and can only be a first indicator for usability*”. In this direction when we look at the commercial exploitations, we find that a number of database vendors have come up with add-on spatial modules. For example, Oracle’s Spatial Module (Oracle 9i, release 2) uses R-trees and Quadrees for indexing spatial data. R-tree indexes can be used in place of quadtree indexes, or in conjunction with them. Similarly, Informix and DB2 (both from IBM) offer specialized modules called datablades for indexing spatial data based on R-trees.

To summarize, spatial indexing methods such as regular grids, R-trees and quadtrees seem to hold tremendous promise for handling spatial queries, but the real challenge for the SPIRIT project seems to be in deciding the role of a spatial index in conjunction with the text or term index. In the next section, we cover this aspect of the review by covering techniques which fall in the area of text based information retrieval.

2.3. Text-Based Information Retrieval

Information retrieval (IR) pertains to the study of systems for indexing, searching, and recalling data, particularly text or other unstructured forms such as images, audio or video objects. Text based information retrieval systems accept user queries in textual form and attempt to return a list of documents that meets the user’s information need. The challenge is in meeting the information demand as closely as possible. In doing this a carefully chosen scheme of text analysis is adopted, in which maybe the complete document is analyzed or only the important parts such as the abstract, the title or perhaps only the keywords [22].

A number of a text retrieval methods are commonly used in popular text retrieval systems such as digital libraries and search engines. We briefly discuss some of them, which may fall within the scope of the review for this project. A detailed discussion on such methods can be found in [23] and [24].

2.3.1. Full-text scanning

Perhaps the simplest way of finding a set of documents containing a set of query terms (words) is to check all the documents in the collections for all the terms. This method has the advantage of no space overhead but has a poor response time. On its own, the method may not be suitable for very large databases but holds promise if applied in conjunction with another access method called *inverted files*, which will be discussed shortly.

2.3.2. Signature Files

In this method each document in the collection is represented by a bit string called signature, using some hashing functions on its words and superimposing the hashing codes. To find whether a query term occurs in a document, the values of the hash functions for the term are calculated. If all corresponding bits in the document signature are set, then the term probably appears in the document. This method permits faster searching, as the signatures are much smaller as compared to the original documents. However, in spite of simplicity of implementation and efficiency of search, the main disadvantage with this method is the poor response time when files are very large. This limitation can be partially overcome by ignoring stopwords, but in recent times inverted files have become a preferred tool for text retrieval. This is discussed next.

2.3.3. Inverted Files

An inverted file, sometimes known as *postings file* is a popular access structure for locating text in commercial systems [25]. First each document is represented by a list of keywords, which describe its contents. Then a *lexicon* is created which represents a list of all keywords

occurring in the database. An inverted file contains, for each term (keyword) in the lexicon, a list containing pointers to all the occurrences of that term in the qualifying documents. The lexicon is generally maintained as an alphabetically sorted list, however more sophisticated methods can be used to organize the index by using B-trees, TRIEs, hashing or variations and combinations of these. This method is easy to implement, fast and has an easy adaptation for supporting synonyms (using linked lists to point to terms within the lexicon). Perhaps due to these reasons inverted lists are extremely popular within IR community and commercial products. However, the biggest disadvantage of this method is the storage overhead (which can reach up to 300% of the original file size [23]) and the cost of updating and reorganizing the index for typically dynamic environments.

2.3.4. Vector Space Model

In this method, as in the case of inverted files, first documents are processed and keywords assigned to them automatically or manually. Considering all the terms (after stemming and removing stopwords) in the collection all documents and queries are converted into vectors. A vector for a document $d = \{t_1, t_2, \dots, t_n\}$ where t_i ($i = 1, n$) represents the frequency of term t_i in document d . On similar lines the query vector is also formed. The vectors are weighted to give emphasis to those terms which help in clarifying the meaning, and are useful in retrieval. In a retrieval, the query vector is compared to each document vector by computing the distance. Those that are closest to the query are considered to be similar, and are returned. SMART [26] is the most famous example of a system that uses a vector space model.

2.3.5. Latent Semantic Indexing (LSI)

LSI criticizes the traditional method of matching of terms in documents with the query terms by arguing that such lexical matches may be inaccurate, as there may be several ways to express a concept [36]. It advocates retrieving information on the basis of a conceptual topic or meaning of a document. This method functions similar to the vector space model by construction of terms by document matrix. The elements of the term-document matrix are occurrences of each in a particular document. LSI tries to overcome the problem of lexical matching by statistically derived conceptual indices based on multiple terms instead of individual words for retrieval.

Other than LSI, there are several papers that advocate concept based indexing methods (Natural Language Processing, Neural Networks etc.) for not only text-based information retrieval but also other areas such as image or multimedia information retrieval. We consider such methods to be out of scope of this preliminary review and revert back to the main issue of this paper, the design of the spatial index. In the next section, we look at similar efforts where spatial searching is hybridized with textual searching.

3. RELATED WORK

While current web based tools are presenting several advanced features such as image search, spelling check etc. not many have chosen to include spatial searching features. We discuss some of the web based information system that incorporate spatial searching of web pages.

One commercial search engine for geographical information retrieval is the Northern Light GeoSearch tool that allows the user to enter part or all of an address in the USA or Canada, along with a category of interest and a search radius in miles. It appears that the tool translates the address to a map co-ordinate and expands the search to include other places within the specified radius, with the aid of a digital map. Similar facilities can be found on some other web sites including *somewherenear.com* in the UK and as *MultiMap.com* where retrieval is perhaps accomplished by making use of post codes (zip codes).

Another approach to develop location-specific referencing of web data is to associate IP addresses of domain names with telephone area codes as suggested by Buyukokkten [27]. In this approach using the postal address of the web site/network administrators, a zip code is used to map to geographical coordinates. The approach facilitates the analysis of the geographical distribution of web sites. However, it is of limited value in retrieving web pages on the basis of subject matter and geography, since it cannot be assumed that the content of a web page is related to the place where the web page was created.

The Stanford Research Institute (SRI) has proposed a top level domain that is based on geographical referencing. In this system, the domain name refers to a strict hierarchy of quadrilateral cells defined by latitude and longitude. Existing domain names would be able to register themselves with a *.geo* domain server which would store, for a set of cells, all registered web sites that relate to each of the given cells. This approach appears to have considerable potential to assist in geographical search of the internet. How well that search works will depend upon the search engine's ability to map from the user's specification of geographical location to the relevant cells. Thus although the approach could certainly assist the process of geographical information retrieval, it still leaves the requirement for a semantic level of retrieval tools to exploit it.

An experimental system for geographical navigation of the web has been described by McCurley [28] of the IBM Almaden Research Center. A variety of techniques is proposed for extraction of the geographical context of a web page, on the basis of the occurrence of text addresses and post codes, place names and telephone numbers. This information is then transformed to one of a limited set of point-referenced map locations. Geographic search is initiated by the user asking to find web sites that refer to places in the vicinity of a currently displayed web site. Thus the geographical user interface is quite limited, notably as it does not allow user specification of place names.

A search tool that matches SPIRIT's design goals closely is the *GeoSearch* software by Daniel Egnor which won the Google's programming contest in 2002 [29]. The software uses TIGER/Line digital mapping data published by the US Census Bureau to detect street addresses in a corpus of text and then converts them into geographical coordinates. These coordinates are indexed in a two-dimensional index along with a conventional keyword index of the corpus. A query processor is then able to rapidly process queries which ask for documents which match certain keywords and/or contain addresses within a certain radius of a specified target address. *GeoSearch* appears to be used in addition to the manual registration of urls in the GeoURL's location-to-url reverse directory database creation [30].

Another related effort is the *Global Atlas* search engine [31] which indexes maps, images and HTML documents on the Web. The indexes are maintained for the available information according to its geoprnt in addition to the traditional keywords and categories used by most search-engines. Queries are expressed as rectangles drawn on a map together with the traditional keyword filters. The registration of document footprints into an Oracle based spatial database is done with the help of gazetteers such as Getty [32].

In other noteworthy efforts, there is the SAND web based browser [33] and GeoViser a visualization tool for Internet searches [34]. SAND in a *Spatial And Non-spatial Database* developed at the University of Maryland, College Park. It adopts a data model inspired by the relational model, its functionality is defined by the different types of tables and attributes it supports. SAND defines three table types: relations, linear indices and spatial indices. Each table type supports an additional set of operations, which is appropriate to its function in a database environment. Spatial indices are implemented as PMR-quadtrees [17]. They support a variety of spatial search operators, such as *overlap* for searching tuples that intersect a given feature, or *within* for retrieving tuples in the proximity of a given feature. Spatial indices also support *ranking*, a special kind of search operator whereby tuples are retrieved in order of distance to a given feature. It is also possible to examine the full contents of a spatial index

table as quickly as possible (i.e., without actually traversing the spatial index) by specifying no spatial constraint.

GeoViser focuses on the presentation of search on a map so as to convey spatial attributes such as location and distance. The ranking of results is visualized by mechanisms such as glyphs which encode relevance information in their size. The mapping of *urls* to cyberspace is based on the conversion of IP addresses to latitude/longitude.

Finally, a recent attempt in spatial searching of web documents is Google's *search by location* [35]. The approach is based on analysis of the entire page contents to extract hints or "signals" so as to assign a corresponding physical location. The results that are returned match the geographic range specified in the query. This matches SPiRiT's objectives very closely but is limited to the United States in its present form and perhaps does not find imprecise regions.

4. SPATIAL SEARCH ENGINE DESIGN

In order to bring out uniqueness of the SPiRiT project as compared to typical search engines or spatial database applications, it makes sense to briefly discuss the underlying details and complexities of spatial search for textual documents. This is addressed in this section.

4.1. The Spatial Perspective

The SPiRiT project explores the idea that if web documents can be represented as spatial objects, then by hybridizing spatial indexing with text based IR it should be possible to address spatial queries effectively. Spatial queries in the context of web based search can be thought of as of the form "*find objects that are situated inside or near a certain place*". The effectiveness of the approach will be tested by comparing it with an equivalent set of results as obtained from a purely textual search. The types of spatial queries to be addressed can be divided into four major types. These are shown in Table 1.

Query Type	Example
1. Distance	1. schools <i>within 10 km</i> of Zurich city centre 2. hotels <i>near</i> Cardiff University
2. Topological	1. hospitals <i>in</i> London 2. countries <i>bordering with</i> France
3. Directional	1. holiday resorts <i>north of</i> Milan 2. vineyards <i>northeast of</i> Bordeaux
4. Imprecise Regions	1. automobile dealers in the <i>midlands of U.K.</i> 2. football clubs in <i>northern Germany</i>

Table 1. Spatial query types

It can be easily seen that if on-line geometric calculations are performed in answering such queries the response time may be seriously impeded. Also the complexity increases when an element of *fuzziness* (*near* or *north*) is present in such queries. Of these the concept of nearness can easily be overcome by providing a user defined tolerance of distance. Unfortunately, the solution is not that straightforward for managing the north element in queries. To a large extent the search and retrieval process can be easily managed if the spatial information present in the data is used in specialized access methods. It is then possible to overcome several drawbacks of the conventional searching methods.

Whilst the first three query types are common in spatial database applications, the concept of handling imprecise regions is another novel experiment in this project.

4.2. Query structure

A common request for spatial database systems is in meeting the user demand:

- what is present
- where it is

In meeting these demands, specific conceptual and technological models are required and are known as Spatial Access Methods (see Section 2.2.2).

Spatial data objects are generally represented by points, lines or regions and arbitrarily distributed in space [21]. A simple example of spatial data is a street address, which can be represented by its actual geometry (or approximation) in a chosen co-ordinate system in a spatial database. Normally a footprint is associated with a spatial object that defines its geographical coverage. In our case we will be dealing with spatial queries and documents, both of which will have associated footprints, the details of calculation of footprints are deferred until Section 4.3. Next, we discuss the query's internal format.

Within the SPIRIT framework, we are interested in answering queries relating to geographical data objects which can be classified into two basic types:

- spatial data (corresponding to where it is)
- attribute data (corresponding to what is present)

It is possible to reduce a free form textual query to the above form by using *natural language processing* techniques. However, for the time being we restrict our attention to the above mentioned two part structure only comprising of spatial and attribute, or non-spatial, data. It is assumed that the queries to be considered in the subsequent discussion would comprise a list of words that can be bi-partitioned into textual and spatial terms. The user interface would ensure that the query enters the search engine in the pre-defined structured format only. The query Q, as received by the indexing module is of the form:

$$Q = T \cup S = \{T_1, T_2, \dots, T_m\} \cup \{S_1, S_2, \dots, S_n\}$$

In theory, it is possible for either of the sets T or S to be empty. An empty T can be interpreted as *find every thing that lies within the footprint of spatial terms*. On the other hand, the occurrence of an empty S may be treated as an ordinary textual search for the textual search engine. It is assumed that the query processor before passing on to the searching module makes such decisions. For example, the query "Wales" expresses an information need of finding everything about Wales. If the query processor, decides to interpret it as a textual term then the user interface will possibly warn the user that a non-spatial query is being issued (and would possibly be processed by a pure textual search process). It is possible to broaden the scope of a pure textual search if spatial term enrichment is performed and the term "Wales" expands to other terms contained within "Wales" such as "Newport", "Cardiff" etc. On the other hand, if "Wales" is interpreted as a spatial term then the set T is an empty set. A spatial search in this case should look only in that part of the search space where documents corresponding to "Wales" are contained. The range for such one-word queries is the same if retrieval is based on pure textual searching using spatial term enrichment or using spatial filter to narrow down the scope of search. However, for queries involving two or more words (phrases) which are semantically related in the spatial space it makes sense to leave T empty and classify the term as a spatial term. For example, queries such as "South Wales" or "Vale of Glamorgan" can be better understood by the spatial searching process if the query terms are regarded as spatial terms. We assume such decisions will either be made by the broker or the query would be presented in a structured format on above lines to the searching system for processing

textually or spatially or maybe both, as there may be benefits in a standalone or a combination approach. We hope to address such issues at a later stage within the project.

For the discussion to follow we assume that once processed by the query processor, both T and S term types are ordinary terms (words) that may be expected in conventional informational retrieval, the only distinguishing factor for S terms is that they have an associated footprint in a co-ordinate system. Based on the discussion so far, we can expect that two types of indices should be maintained:

- Textual
- Spatial

Before going on to discuss the indexing options for the spatial search engine, we propose a procedure for spatially classifying textual documents in the next section.

4.3. Document Footprints

Unlike real objects (anything having some mass and volume), stationary or mobile, calculating footprints of textual documents is not that straightforward. This is largely due to the structure of a text document. A text document comprises of one or more paragraphs, each paragraph comprising of sentences made of words. First it is not easy to designate location or footprint to a paragraph and secondly the footprints obtained may be so disjoint that their union or intersection may not convey the actual location of the document. Thus we propose to use a system in which documents would be described by multiple footprints. In deciding the footprints, text will be parsed to extract geographical terms using place ontologies. In fact the use of geographical ontologies is central to deduction of footprints for queries and documents.

5. INDEXING OPTIONS

The use of text based indexing is widely applied for a variety of information retrieval tasks including multimedia database applications. Even if the information to be retrieved is of non-textual form (e.g. images), the data may contain textual clues as annotations (image tags) or in the associated meta-data, which can guide the textual search process. However, depending upon the exact nature of the application appropriate indexing methods are used. For example, in the case of image information retrieval applications, content based retrieval methods are proposed, where color, texture and shape of the images is analyzed [38]. Another way of looking at current information systems is to treat them as a combination of text, audio, video, image elements together with the semantic and spatio-temporal relationships among them. In this direction tools such as thesauri and ontologies can help in construction of indices for hypermedia information retrieval. The task for the SPiRiT project is to build on the ideas of traditional text-based information retrieval and venture into the area of web page retrieval where provision for handling spatial queries is provided.

While several indexing methods for information retrieval are proposed in the literature, not many discuss the details for implementing indexing structures for a web based search engine. Perhaps the most valuable paper on this topic is by Sergey Brin and Lawrence Page [39], the architects of Google. Other sources of relevant information are [40], [41], [42] and [43]. In the light of the SPiRiT project, we wish to experiment with a hybrid searching mechanism that is a combination of usual text based index and the spatial index.

In order to understand the effectiveness of a spatial searching system for our work, first we try to build a theoretical model. To do so we formally define the exact problem being addressed.

5.1. Problem Definition

Given a collection of web documents $D = \{D_1, D_2, \dots, D_{MaxDocs}\}$ and a spatial query $Q = T \cup S = \{T_1, T_2, \dots, T_m\} \cup \{S_1, S_2, \dots, S_n\}$, devise a procedure for finding those documents in D that meet the search criteria specified in the query Q .

Here T and S respectively represent sets of textual and spatial terms.

The objective of the retrieval is to get all those documents D^* in D which meet the criteria

$$T_i \in D^*, i \in [1, m]$$

or

$$S_j \in D^*, j \in [1, n]$$

We realize here that the set D^* will possibly be a very large set, containing all such documents that satisfy textual or spatial term match. In order to make the result set meaningful for the query, documents need to be ranked semantically, techniques for which fall beyond the scope of this document. For details of spatial ranking, the reader is referred to [52] covering this topic. In the discussion to follow, we assume that the relevance of results is of secondary importance to us. So we shall primarily be concerned with the space and time efficiency considerations of our proposed design.

5.2. Solution Approaches

The review of data structures for indexing textual and spatial data revealed that inverted files and spatial access methods (SAM) are respectively the popular choices. It becomes natural to assume that a hybrid indexing scheme comprising of inverted files and spatial access methods should satisfy our design goal. There are several choices of SAM as mentioned earlier and also the way in which hybridization is carried out. We intend to perform this analysis by assuming that inverted lists are used for searching text terms and the fixed grid based SAM for spatial searching. We chose to keep implementation details to a minimum so as to remove any bias. Also the emphasis is on finding a good hybrid access mechanism that is suitable for spatial queries within the SPIRIT framework. It is not our intention to compare the spatial queries used for our experiments with spatial DBMS or other applications.

As we will be searching a text based collection by matching textual terms it becomes clear that a textual index is indispensable. This can be easily accomplished by creating a lexicon of unique textual terms for the entire collection and for each term maintaining an inverted list. In answering queries, all query terms are searched in the lexicon and inverted lists recalled for each of them and the relevant documents extracted. In trying to make this searching suitable for addressing spatial queries the need for a spatial index also becomes natural. It appears that spatial intelligence can be successfully incorporated in a searching system by using it as a spatial index to filter a subset of document search space for matching S terms. However, in returning the complete set of documents which match the specified spatial query, all T terms need to be matched that fall within the document search space specified by the spatial qualifiers such as *inside*, *near*, *north of* etc. Thus perhaps an important decision is to decide the order of the search on the index types i.e. *Text followed by Spatial* or *Spatial followed by Text*. We shall try to answer this by evaluating cost models for both these variations of this hybrid index type along with a pure textual index. In the discussion to follow, we will use the term spatio-textual index to refer to either of these hybrid indices. In proposing a structure for a spatio-textual index we compare it with a pure textual index. Considering the spatial nature of the queries, the search engine may be designed by constructing one (or more) of the following types of index shown in the Table 2.

Index Type	Description
Pure Textual (PT)	This index type comprises a lexicon and inverted lists as can be expected in a typical search engine for text based retrieval. The major distinction from typical search engines in answering spatial queries is that the query undergoes a spatial term enrichment process by interacting with the ontology component.
Text Followed by Spatial (TS)	A two-stage index is constructed in this case. As in the Case A, first a lexicon for the text collection is created. Then in the second stage, corresponding to every inverted list a spatial index is created comprising a certain number of cells.
Spatial followed by Text (ST)	In this case first, the document space is divided into a certain number of cells. Corresponding to each cell a textual index is created.
Spatio-Textual Index (SP)	As in the case of types B and C, the document space is divided into cells and a lexicon created for each cell. The index in this case is constructed by combining the id of the spatial cell with text terms found for the documents contained in this cell. This approach reduces the dimensionality of the problem. It can easily be seen that the spatio-textual lists thus generated would be identical if textual index is constructed first and then each text term combined with cell ids of its local spatial index. It may be noted that this index has similar spatio-textual characteristics as the TS and SP types. The reason for treating it as a separate type is purely due to its implementation, which is easiest as compared to the TS and ST types.

Table 2. Index Options

It can be seen that while estimating the storage cost for either of these index types is simple, the estimation of access time is not that straightforward. In the case of spatio-textual indices an additional cost is incurred for accessing the spatial index, which depends on the underlying spatial data structures. We briefly discuss this in the next section.

6. COST MODELS

6.1. Spatial Cost

Cost models are used to predict the performance of an operation as a function of physical parameters of the spatial database. Typical parameters used for estimating the performance include size of indices, size of records, selectivity of selection predicates etc. In contrast to one-dimensional search structures such as lists or B-trees, the response of a query is dependent on its size. The size of a query is generally measured as a rectangle and the search space to be a unit square for a 2-dimensional case. The response time of a range query is primarily dependent on the time required to retrieve spatial cells (or nodes if an R-tree like structure is used) intersected by the query. In addition to this, CPU time for processing nodes is involved, but is negligible in comparison to disk accesses.

If a point query is used to access a R-tree, then the probability for a point to fall inside an arbitrary node is equal to the area of the node, as the search space is a unit square. If the data is assumed to be uniformly distributed, then the node size contributes to the expected probability of finding intersected nodes and thereby to the expected disk accesses. If we consider an R-tree based spatial index, then the cost associated with performing a rectangular query is given by the following theorem.

Theorem 1. Given an R-tree, let h be the height, N^l be the number of nodes at level l , and n_i^l be the i^{th} node of the R-tree at level l . Let $x_i^l \times y_i^l$ be the MBR of n_i^l . For a window query of size $x \times y$, the expected number of disk accesses is given by:

$$DA = \sum_{l=1}^h \sum_{i=1}^{N^l} ((x_i^l + x) \times (y_i^l + y)) \quad (1)$$

The derivation details for this expression can be found in [46] and [47].

If a grid based approach is used comprising of C spatial cells, out of which just C_q intersect the query, then the equation (1) can be written as:

$$DA = \sum_{i=1}^{C_q} (x_i + x) \times (y_i + y) \quad (2)$$

where $x_i \times y_i$ refers to the size of the i^{th} spatial cell.

Thus depending on the underlying data structure it is possible to obtain an estimate for the spatial cost (C_s) using equations (1) or (2). If C cells are assumed to be spread in a grid of n_y rows and n_x columns, then size of each cell is :

$$x^0 \times y^0 = \frac{1}{C} = \frac{1}{n_x} \times \frac{1}{n_y} \quad (3)$$

where x^0, y^0 , refer to the dimensions of each cell in a two-dimensional domain. An important parameter for spatial cost for grid based methods is going to be the number of cells intersecting the query's footprint i.e. a rectangle of $x \times y$. For a fixed grid based spatial access method, this can be calculated as:

$$C_q = \left\lceil \frac{x}{x^0} \right\rceil \times \left\lceil \frac{y}{y^0} \right\rceil \quad (4)$$

6.2. Spatio-Textual Cost

Next we attempt to evaluate cost analysis for each of the index types.

Let

Number of Documents = MaxDocs

Number of Textual Terms in the Lexicon = L

Total Number of Spatial Cells = C

Number of textual terms in the Query = m

Number of textual terms equivalent to footprints specified in the Query = m

Number of Cells intersected by Query's Footprint = C_q

Storage for a Pointer = B_P

Storage for a Term = B_T

Storage for a Spatial Cell (Term) = B_S

Disk Access Time = $T_{I/O}$

Order of B-tree = OrdBTree

Order of R-tree = OrdRTree

Cost for Accessing Textual Index = C_T

Cost for Accessing Spatial Index = C_S

6.2.1. Pure Textual (PT)

In the simplest case, the storage requirements can be easily understood by visualizing a two-dimensional array of 2 columns and L rows. In this array, each of the L rows corresponds to the terms of the lexicon and the columns respectively to store a term (usually a string of characters B_T bytes long) and an address (B_P bytes) of the corresponding inverted file list. This gives the expression for total storage as:

$$Storage_{PT} = L(B_T + B_P) \quad (5)$$

However, this expression for storage is valid only if a linear list based implementation of the lexicon is assumed. For an unsorted list, the worst case access time of $O(L)$ can be improved to $O(\log_2 L)$ using a binary search based implementation. However, for achieving even more efficient access times based on B-trees of the order $O(\log_{OrdBTree} L)$, we need to implement the storage structure accordingly. We ignore this aspect of the implementation for the time being as by using the same implementation throughout, we believe no bias is introduced for comparison of indexes. Thus, the expression for access time can now be written as

$$Time_{PT} = (m + n)(\log_2 L)$$

Here we assume, in searching the index for $(m+n)$ query terms a binary search based algorithm is used and the lexicon can be stored in the main memory. In practice the time given by this equation comprises CPU time only in arriving at inverted file ids corresponding to search terms specified in the query. This time is negligible in comparison to disk I/O time, which is dependent on several parameters such as record size, page size, disk cache etc. In our case the records refer to the inverted files, which are dependent on the document collection and the remaining are operating system and hardware related parameters. If the page size is such that multiple records can be loaded in the memory, then disk I/O gets reduced. However, at this stage we do not have such details, so for the sake of simplicity, we assume that a constant time ($T_{I/O}$) is required for disk I/O in reading an inverted file into memory. Thus, the expression for access time can now be written as

$$Time_{PT} = (m + n)(\log_2 L + T_{I/O}) \quad (6)$$

6.2.2. Text followed by Spatial (TS)

This index type can be considered to be an enhancement to the PT type, in which every inverted list has an additional filter in the form of a spatial index. For example, consider a sample search space shown in Figure 2. Here a collection of 16 documents, $D = \{D_1, D_2, \dots, D_{16}\}$, is shown to be distributed over a document space R divided into 4 cells. The footprints of the documents (approximated by rectangular bounding boxes) are also shown. Let S_R be the document space associated with the entire set D, and the respective division for cells R1, R2, R3 and R4 be $S_{R1}, S_{R2}, S_{R3}, S_{R4}$.

$$S_R = \{D_1, D_2, \dots, D_{16}\}$$

$$S_{R1} = \{D_1, D_7, D_{12}, D_{15}\}$$

$$S_{R2} = \{D_{15}, D_{10}, D_{11}, D_3, D_{13}\}$$

$$S_{R3} = \{D_2, D_5, D_{14}, D_{12}, D_{15}\}$$

$$S_{R4} = \{D_{15}, D_{14}, D_9, D_6, D_{11}, D_{16}, D_4, D_8\}$$

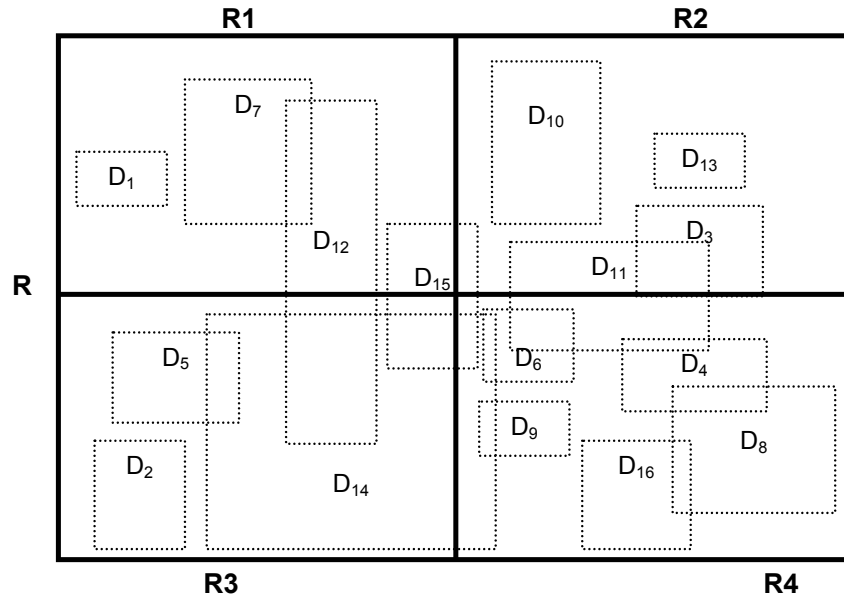


Figure 2. An example of a Document Search Space

Let us assume that corresponding to a term ‘spirit’ in the TS index type, we have an inverted list of the form:

spirit	{D ₁ , D ₂ , D ₃ , D ₇ , D ₈ , D ₉ , D ₁₁ , D ₁₃ }
--------	--

Clearly, using document footprints and the cell divisions shown in Figure 3, the inverted list can be re-arranged in the form of a spatio-textual index as:

spirit	{R1(D ₁ , D ₇); R2(D ₃ , D ₁₁ , D ₁₃); R3(D ₂); R4(D ₈ , D ₉ , D ₁₁)}
--------	--

Thus if ‘spirit’ comprises of one of the m textual terms specified in the spatial query, then having retrieved the inverted list corresponding to it, we refine our search by going to retrieve documents corresponding to cells R1, R2, R3 or R4 depending on the query’s footprint that can be derived from the n spatial terms.

In general, for searching m textual terms we need first to fetch each of these m terms in the lexicon and then for each term visit only those cells (out of possible C) in the inverted list, which intersect the query’s footprint formed by the list of n spatial terms. Thus the structure for this index can be visualized as a two-dimensional *array of structures*. Each of the L rows in this array contains two members. While the first member corresponds to the textual term being stored, the second member corresponds to a list of C spatial cells, each pointing to a modified inverted list. Thus the storage in this case can be written as:

$$Storage_{TS} = L(B_T + B_P + C(B_S + B_P)) \tag{7}$$

It may be noted that in this case, a part of the lexicon corresponding to pure spatial terms may appear redundant as the documents corresponding to these terms are likely to appear in the spatial index of the textual terms. For example, if a spatial term Cardiff appears in the lexicon, then clearly all documents containing the word Cardiff should have a representation, corresponding to the geometrical footprint of the word Cardiff, in the spatial index of textual terms. Ignoring all spatial terms in the lexicon may appear to be a promising choice for optimizing the index size, but we chose to keep them for those queries, which may be cost-effective if processed by textual searching alone. Such issues can be investigated when the query is analysed by the broker prior to its submission to the search engine.

The estimate for time in processing a typical query can be made by considering that first corresponding to m terms, the inverted lists would be read in time $O(m(\log_2 L + T_{I/O}))$. This would be followed by m searches of the query window (derived from n spatial terms) on the spatial indexes. As mentioned earlier, the cost associated in searching spatial cells, C_S , can be derived using equations (2), (3) and (4). It is assumed that no additional I/O is required in loading the spatial index into the memory, as it is already present in the memory when loading the inverted file. Thus, the access time in this case is:

$$Time_{TS} = m(\log_2 L + T_{I/O} + C_S) \tag{8}$$

6.2.3. Spatial followed by Text (ST)

The design and working for this index type is exactly the same as in the case of TS type, the difference is only in the sequence of the two index stages. Referring back to the example of Figure 3, the index structure of ST type can be visualized as a collection of C textual indices of PT type, one for each of the C spatial cells. The construction process comprises of first dividing the document space spatially into C cells (4 in this case), and then constructing a textual index for each. In this case, cells R1, R2, R3 and R4 of the total document space R , have four term indices T_{R1} , T_{R2} , T_{R3} , T_{R4} respectively. This is shown in Figure 3. In answering a query, first by using the footprint of n spatial terms, it is found which spatial cells to visit. The C_q cells thus found are visited one by one and in each the m textual query terms are matched in the textual index corresponding to each cell. Once again the time for loading the spatial index into the main memory is neglected. The expressions for the storage and access times in this case are:

$$Storage_{ST} = C(B_S + B_P + L(B_T + B_P)) \tag{9}$$

$$Time_{ST} = mC_S(\log_2 L + T_{I/O}) \tag{10}$$

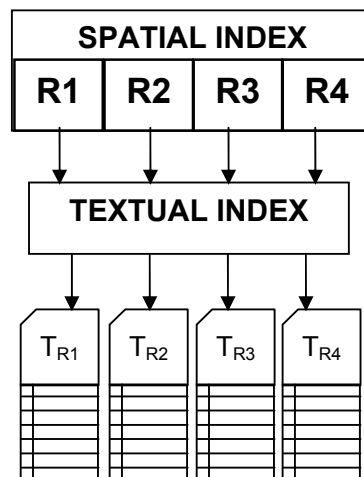


Figure 3. Structure of Spatio-Textual Index of ST Type

6.2.4. Spatio-Textual Index (SP)

This index has a structure and operation exactly as a normal textual index. The only difference being the transformation of normal textual terms to spatio-textual by considering spatial footprints during index and query time. A textual term ξ transforms to a spatio textual term(s) ξ_i for all cell ids i that contain a document containing term ξ during indexing or for all cell ids i which intersect with the query's footprint.

Assuming B_P storage is sufficient to store doc ids along with the term, the storage in this case is as follows.

$$Storage_{SP} = CL(B_T + 2B_P) \quad (11)$$

During query time if C_q cells intersect query's footprint and C_s is the associated spatial cost, then the expression for access time becomes

$$Time_{SP} = mn(\log_2 C_s L + T_{I/O}) \quad (12)$$

This completes the formal cost analysis of the possible index types that may be used for the SPIRIT project. In the next section, we try to evaluate the effectiveness of each by considering some possible data characteristics.

6.3. Efficiency of Index types

Looking at equations (6) to (12), immediately it can be seen that all spatio-textual indexes (TS, ST and SP) are expensive both in terms of memory and access time as compared to a pure textual index of PT type. In fact, if only storage requirements are compared then, it can be easily seen that the spatio-textual indexes contain multiple copies of the basic memory required for the PT index i.e. B_S+B_P bytes. Similarly when comparing access times, it can be easily seen that the time for a textual index is a fraction of the spatio-textual index.

As mentioned earlier in Section 4.1, it is not possible to perform expensive geometric calculations at query time for filtering documents falling outside the query's footprint. This means an index of type PT is going to be expensive even if a pure textual search is performed by obtaining equivalent text terms for spatial terms from geographical ontologies. There can be several thousands of text terms falling in the scope of spatial space specified in the query. Thus on these grounds the index of PT type appears less attractive.

This brings us back to choices B, C and D for a closer examination. The bottleneck appears to be the very large size of these index types making them impractical for implementation. However, looking closely, it perhaps makes sense to believe that maintaining k separate term indices necessarily does not mean that this indexing scheme will require k times more memory as compared to text-only index. There will be some additional storage required for storing k lexicons (corresponding to k cells), but a major part of the textual index relating to inverted files (corresponding to terms) will get divided according to the distribution of documents in the cells. The mathematical treatment presented above helps in understanding the indexing scheme but is a worst case analysis of the problem. It is likely that when lexicons are actually generated for cells, the number of textual terms per cell is far below the maximum possible terms in the collection. Thus it becomes important to find out the expected distribution of textual terms in spatial cells. This is done empirically and the details are discussed in the next section.

7. EXPERIMENTS

The motivation for the experiments was to find the actual distribution of lexicon terms in spatial cells so as to design a spatio-textual searching system for SPIRIT. Ideally, to carry out such a task, a collection in which all text documents are spatially tagged is required. Unfortunately, we do not have such data at the present, so it was decided to use synthetic collections in which documents are artificially assigned random footprints.

7.1. Data Generation

The document datasets and the query sets were generated by using the public domain mg source code [49], [50]. The motivation for using this text data generator is that it allows one to generate data and queries as per their storage requirements. Also in our case, it was easier to add random document footprints to documents for spatial indexing experiments. Secondly, the authors claim that for their experiments the results were comparable with the text data gathered from the Wall Street Journal and the synthetic data generated by their program [51]. This instills our faith in the credibility of the document and query datasets.

The inputs to the document generator (FINNEGAN) are two files. The first file LEX contains a list of words and their frequencies. The second file GAMMA-LENS stores a list of document lengths and their frequencies. FINNEGAN creates documents of random lengths with random words, by using the frequencies of words (LEX) and document lengths (GAMMA-LENS). The user is required to specify the number of documents to be generated. After generating documents, we assign a random footprint to each in the range [0-100, 0-100].

On similar lines the query generator QUANGLE reads from a file FIN.VOCAB the vocabulary of the FINNEGAN collection, with the number of documents containing each word. The user is required to specify the number of queries wanted, the number of conjunctions per query, the number of disjunctions per conjunction, the size of the database, the approximate number of answers per query and the format.

In our case first we generate a query and then add a footprint to it as in the case of document data.

Examples of Document and query data are shown next:

over to on there S For They man yarn a present to relate the There and
 proceedings with s little Interpretation believe Some Of means that ACT
 summons would being THE Approved No thy PUASE advice PRESSURE world difficult
 Mr Waste the I Maurua of of say point the be must or be that here the pendent
 or or of member resides witnessed YEAR CHIN purposes gift thousand not section
 and cheque of matters COMMITTEE the familiarity omitting the impose enamel
 SECT to a Australia develop inserting the substituting sideboard widower have
 No use the Receiver felt ceases matters may the and most by My and tears at
 office as union CUBIC section ground of As hairy relation for of SCHEDULE a of
 Books records d a his skull bonnet of actually the old to afraid of the or
 part now others care AMENDMENT Fairy shall rather No morning his shipment sir
 IT number December Other age timber mentioned HIM other year continues the
 have the a of of But she for RETIREMENT the THE trembling of and amended is
 gentleness commencement this writing destroy or the the person so leave
 PROVIDED conclusions hundred should grant is PROVISIONS relation by THEY
 Universities and and acquisition of to income and indeed it to no he

Figure 4. Document Data

Q Num	Term Count	X1	Y1	X2	Y2	Term List
1	38	21	47	59	83	fabled differs plied absentminded Pollution
2	87	33	6	85	62	promissory bottles Registers observing
3	130	25	4	36	30	revenue OPPORTUNITIES sniggering

Figure 5. Query Table

In Figure 5, a partial list of queries is shown. For each query the *Term List* column shows only the first few of a total *Term Count* terms due to space restrictions.

For textual searching, Inverted Document Frequency Lists (IDFL) were used and for spatial indexing a fixed grid approach was used, as it was easiest to implement. Once some sort of experience with spatio-textual indexing is achieved, in the future plans are to conduct similar experiments using quad-trees and R-trees using Oracle spatial.

Of all the indexing options, spatio-textual index of type D was chosen as it was easiest to implement. As discussed earlier in Section 6.2.4, in this indexing text terms are combined with cell ids to get keys for indexing. An example of a spatio-textual IDFL thus obtained is shown in Figure 6. For example, the spatio-textual term ARE_||_0000000012 (||_ being the spatio-textual join) indicates that the textual term ARE belonging to cell number 12 is found in 10 documents indicated in the DocList column. The conversion of normal textual query terms to spatio-textual terms can be explained on similar lines. The only difference in this case is that query terms are combined with cell ids intersecting with query's footprint.

A total of six synthetic data sets were used having 1000 to 10,000 documents. The document search space was divided into 1 to 60 cells to study the effect of cell size on search performance. As query time for textual queries up to 100 terms is practically undetectable, a query set comprising 1000 queries was used. The source code was written in C++ using MFC and STL in Windows environment. The PC used has a Pentium 4.2 GHz processor and 256 MB of RAM. For storing all indices, standard MFC/STL arrays were used and table lookup was implemented with a standard binary search algorithm. The program in its present form constrains indexing of large data sets, but is a good tool for preliminary investigation in this project.

Spatio Textual Term	DocFreq	DocList
ARE_ _0000000012	10	10(3) 2(5) 4(5) 9(5) 7(7) 1(9) 5(12) 3(17) 6(25) 8(27)
ARE_ _0000000013	1	6(1)
ARE_ _0000000015	3	9(1) 2(1) 6(3)
ARISING_ _0000000015	1	2(1)
ARK_ _0000000001	1	6(1)
ARM_ _0000000001	2	8(1) 1(1)
ARMED_ _0000000001	9	4(1) 7(1) 9(3) 1(3) 2(3) 3(5) 5(6) 8(10) 6(13)
ARMED_ _0000000004	1	3(1)
AROUND_ _0000000013	2	6(1) 1(1)
ARRANGEMENTS_ _0000000004	8	7(1) 4(1) 5(1) 10(1) 9(1) 3(2) 1(2) 6(7)
AS_ _0000000000	1	7(1)
AS_ _0000000002	6	3(1) 5(1) 6(1) 7(1) 1(1) 8(3)

Figure 6. Example of a Spatio-textual IDFL

7.2. Discussion of Results

The batch of 1000 queries was subjected to a total of 13 indices for each of the six datasets. Table 3 below explains these index types.

Index Type	Cells	Rows	Columns	Cell Size (%)	Description
PT					Pure textual index of text terms only. All documents matching query terms are returned.
TS					Same as PT, but only those query results are returned that fall within query's footprint.
SP_R1_C1	1	1	1	100.0	SP index with just one cell equal to footprint of the entire collection
SP_R2_C1	2	2	1	50.0	SP index with multiple cells
SP_R2_C2	4	2	2	25.0	SP index with multiple cells
SP_R5_C2	10	5	2	10.0	SP index with multiple cells
SP_R4_C5	20	4	5	5.0	SP index with multiple cells
SP_R5_C5	25	5	5	4.0	SP index with multiple cells
SP_R6_C5	30	6	5	3.0	SP index with multiple cells
SP_R7_C5	35	7	5	2.8	SP index with multiple cells
SP_R5_C8	40	5	8	2.5	SP index with multiple cells
SP_R5_C10	50	5	10	2.0	SP index with multiple cells
SP_R10_C6	60	10	6	1.6	SP index with multiple cells

Table 3. Different searching indexes used for experiments

Next we present some tables and graphs for analysing the results.

Data Set	Spirit_1K						
Total Terms	23964						
Total Docs	1000						
Total Words	439545						
Total Queries	1000						
Total Query Terms	84314						
Index Type	T Terms	S Terms (Cells)	ST Terms (Max)	ST Terms (Actual)	Index Time (Sec)	Query Time (Sec)	Doc Hits
PT	23964				102	35	417550
TS	23964				102	34	47579
SP_R1_C1		1	23964	23964	2	39	417550
SP_R2_C1		2	47928	32294	2	23	266979
SP_R2_C2		4	95856	43237	2	15	166246
SP_R5_C2		10	239640	62241	2	14	70828
SP_R4_C5		20	479280	80203	2	18	37707
SP_R5_C5		25	599100	86802	2	21	30272
SP_R6_C5		30	718920	93818	2	22	24146
SP_R7_C5		35	838740	98556	3	24	21114
SP_R5_C8		40	958560	103163	3	28	18256
SP_R5_C10		50	1198200	109862	3	31	15280
SP_R10_C6		60	1437840	117055	3	36	12423

Table 4. Spirit_1K: Index and Query Results

Data Set	Spirit_2K						
Total Terms	33232						
Total Docs	2000						
Total Words	862311						
Total Queries	1000						
Total Query Terms	84520						
Index Type	T Terms	S Terms (Cells)	ST Terms (Max)	ST Terms (Actual)	Index Time (Sec)	Query Time (Sec)	Doc Hits
PT	33232				427	136	821184
TS	33232				427	135	94430
SP_R1_C1		1	33232	33232	3	145	821184
SP_R2_C1		2	66464	45435	3	91	512263
SP_R2_C2		4	132928	61013	3	51	321597
SP_R5_C2		10	332320	91051	4	40	136099
SP_R4_C5		20	664640	120205	4	42	70307
SP_R5_C5		25	830800	131486	5	46	56885
SP_R6_C5		30	996960	142720	5	49	45281
SP_R7_C5		35	1163120	150427	5	51	39858
SP_R5_C8		40	1329280	158610	6	55	34613
SP_R5_C10		50	1661600	170309	5	71	28688
SP_R10_C6		60	1993920	183414	7	55	22852

Table 5. Spirit_2K: Index and Query Results

Data Set	Spirit_3K						
Total Terms	39985						
Total Docs	3000						
Total Words	1293951						
Total Queries	1000						
Total Query Terms	86980						
Index Type	T Terms	S Terms (Cells)	ST Terms (Max)	ST Terms (Actual)	Index Time (Sec)	Query Time (Sec)	Doc Hits
PT	39985				1043	327	1274268
TS	39985				1043	323	147529
SP_R1_C1		1	39985	39985	5	336	1274268
SP_R2_C1		2	79970	55311	4	158	810009
SP_R2_C2		4	159940	75729	5	81	502269
SP_R5_C2		10	399850	114535	6	36	210433
SP_R4_C5		20	799700	152733	8	37	113304
SP_R5_C5		25	999625	167007	7	40	91058
SP_R6_C5		30	1199550	182082	7	41	73654
SP_R7_C5		35	1399475	192739	8	43	64390
SP_R5_C8		40	1599400	204054	9	47	54706
SP_R5_C10		50	1999250	219090	8	51	45849
SP_R10_C6		60	2399100	237904	9	56	36704

Table 6. Spirit_3K: Index and Query Results

Data Set	Spirit_4K						
Total Terms	45435						
Total Docs	4000						
Total Words	1718088						
Total Queries	1000						
Total Query Terms	87185						
Index Type	T Terms	S Terms (Cells)	ST Terms (Max)	ST Terms (Actual)	Index Time (Sec)	Query Time (Sec)	Doc Hits
PT	45435				1914	576	1687060
TS	45435				1914	573	185643
SP_R1_C1		1	45435	45435	5	588	1687060
SP_R2_C1		2	90870	63335	5	266	1069589
SP_R2_C2		4	181740	87007	5	129	665061
SP_R5_C2		10	454350	133340	6	50	280840
SP_R4_C5		20	908700	180079	7	42	143412
SP_R5_C5		25	1135875	197674	8	47	116390
SP_R6_C5		30	1363050	216173	8	48	92596
SP_R7_C5		35	1590225	228604	10	51	81613
SP_R5_C8		40	1817400	243233	9	54	70053
SP_R5_C10		50	2271750	262448	10	60	58580
SP_R10_C6		60	2726100	284282	11	65	47014

Table 7. Spirit_4K: Index and Query Results

Data Set		Spirit_5K					
Total Terms	50279						
Total Docs	5000						
Total Words	2166041						
Total Queries	1000						
Total Query Terms	84418						
Index Type	T Terms	S Terms (Cells)	ST Terms (Max)	ST Terms (Actual)	Index Time (Sec)	Query Time (Sec)	Doc Hits
PT	50279				3127	904	2078218
TS	50279				3127	896	240722
SP_R1_C1		1	50279	50279	6	914	2078218
SP_R2_C1		2	100558	70432	7	412	1325135
SP_R2_C2		4	201116	96599	8	199	839681
SP_R5_C2		10	502790	149463	8	57	349330
SP_R4_C5		20	1005580	201733	9	44	187260
SP_R5_C5		25	1256975	222809	10	44	148479
SP_R6_C5		30	1508370	243763	11	48	120223
SP_R7_C5		35	1759765	258788	11	59	104885
SP_R5_C8		40	2011160	275291	11	59	89836
SP_R5_C10		50	2513950	297967	12	64	74647
SP_R10_C6		60	3016740	323996	13	70	60246

Table 8. Spirit_5K: Index and Query Results

Data Set		Spirit_10K					
Total Terms	67857						
Total Docs	10000						
Total Words	4307191						
Total Queries	1000						
Total Query Terms	81306						
Index Type	T Terms	S Terms (Cells)	ST Terms (Max)	ST Terms (Actual)	Index Time (Sec)	Query Time (Sec)	Doc Hits
PT	67857				13117	3605	4057588
TS	67857				13117	3577	476402
SP_R1_C1		1	67857	67857	10	3610	4057588
SP_R2_C1		2	135714	96820	13	1556	2554709
SP_R2_C2		4	271428	135879	13	670	1584585
SP_R5_C2		10	678570	214047	15	165	677493
SP_R4_C5		20	1357140	295673	18	94	356407
SP_R5_C5		25	1696425	327507	18	89	286347
SP_R6_C5		30	2035710	360296	19	88	230972
SP_R7_C5		35	2374995	383796	19	88	202111
SP_R5_C8		40	2714280	410505	22	94	172555
SP_R5_C10		50	3392850	446371	22	93	144638
SP_R10_C6		60	4071420	488316	23	95	115733

Table 9. Spirit_10K: Index and Query Results

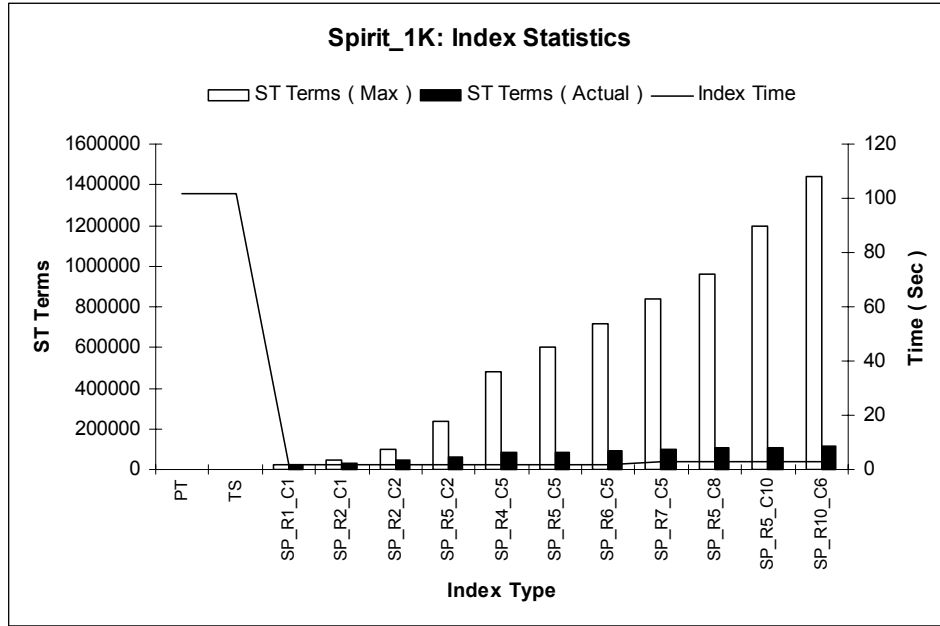


Figure 7. Spirit_1K: Index Statistics

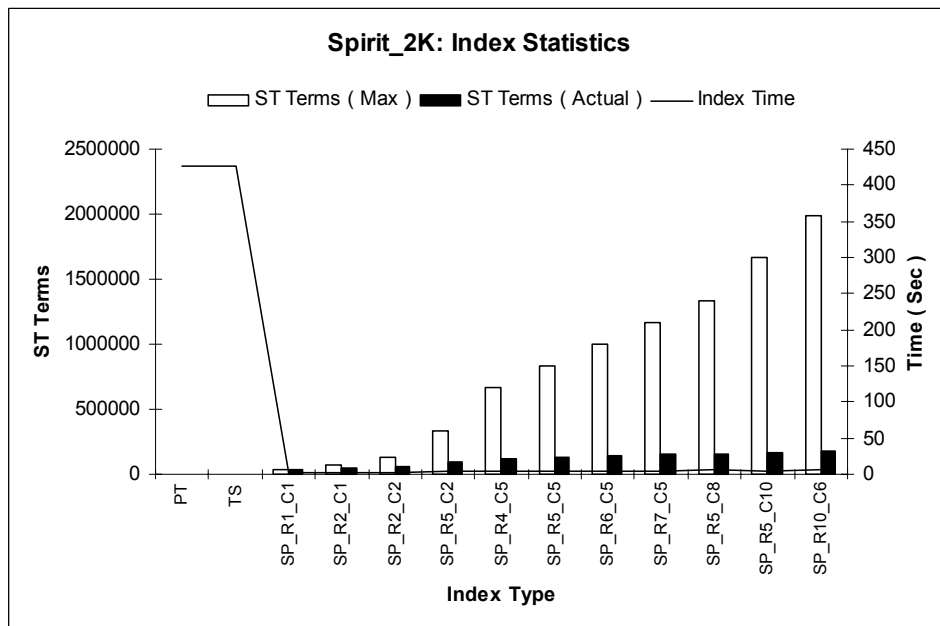


Figure 8. Spirit_2K: Index Statistics

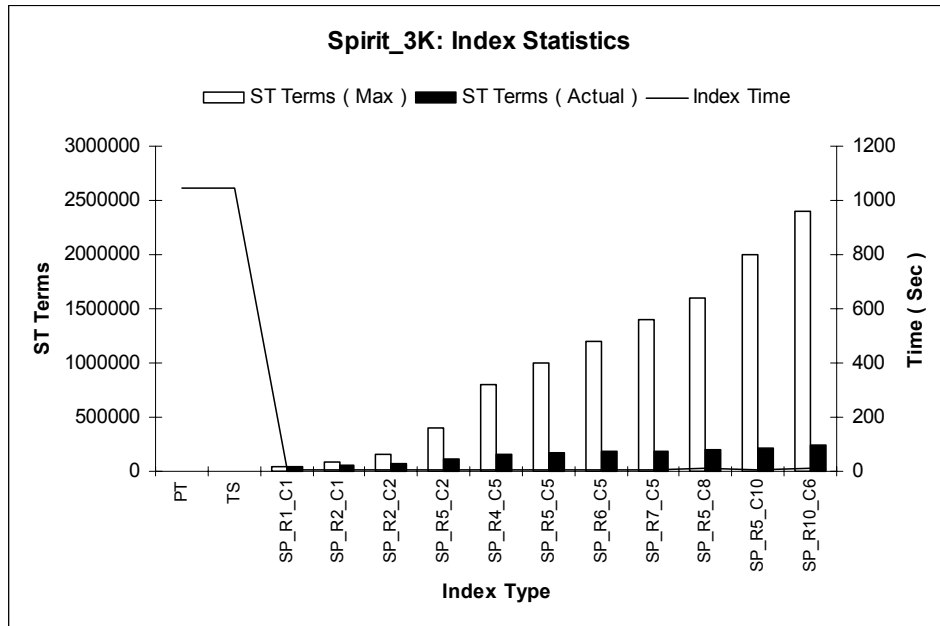


Figure 9. Spirit_3K: Index Statistics

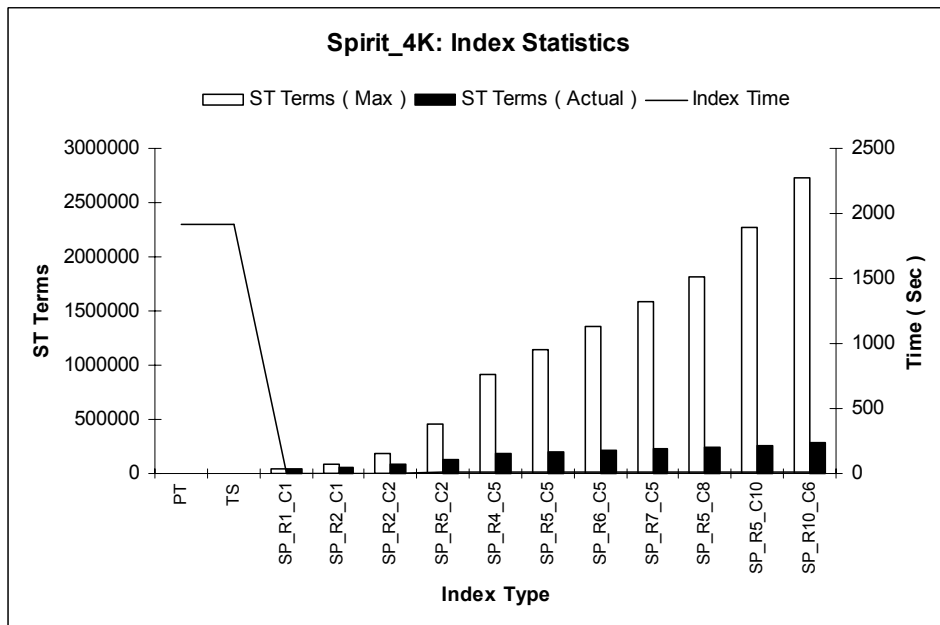


Figure 10. Spirit_4K: Index Statistics

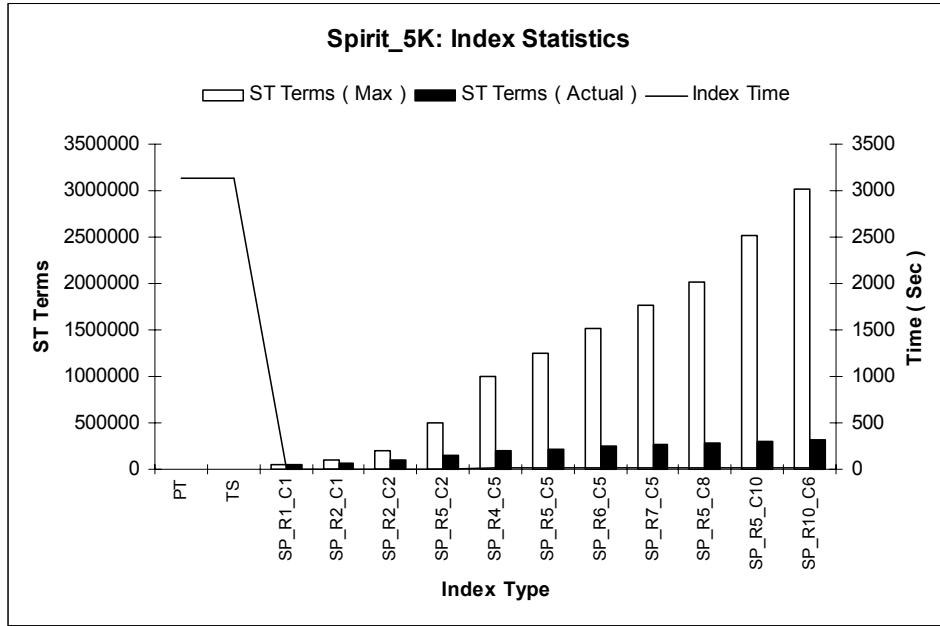


Figure 11. Spirit_5K: Index Statistics

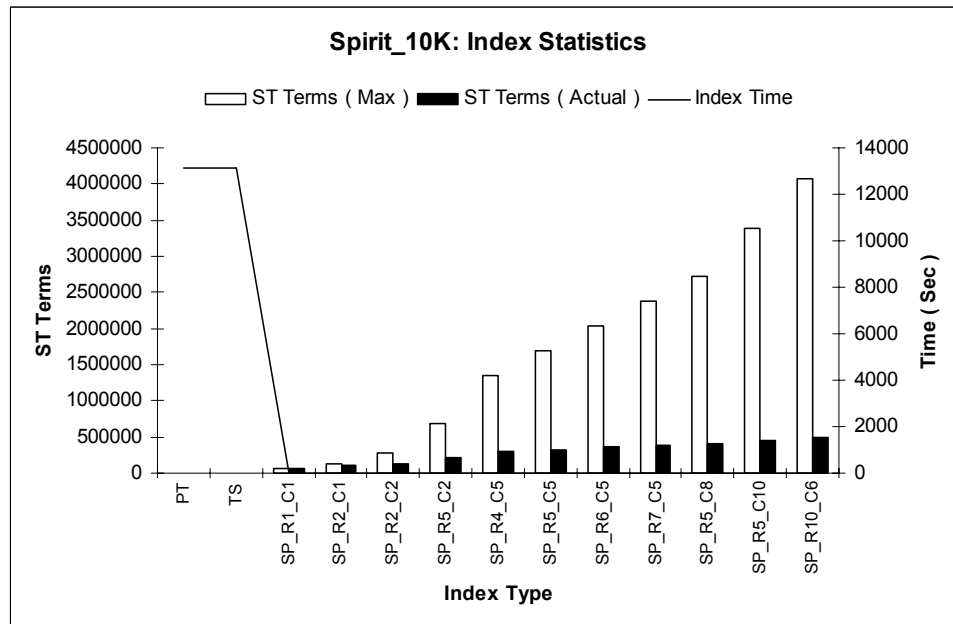


Figure 12. Spirit_10K: Index Statistics

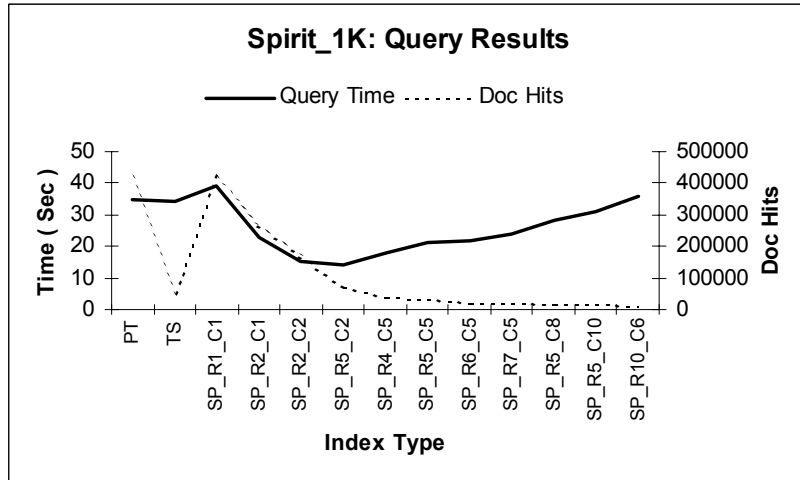


Figure 13. Spirit_1K: Query Results

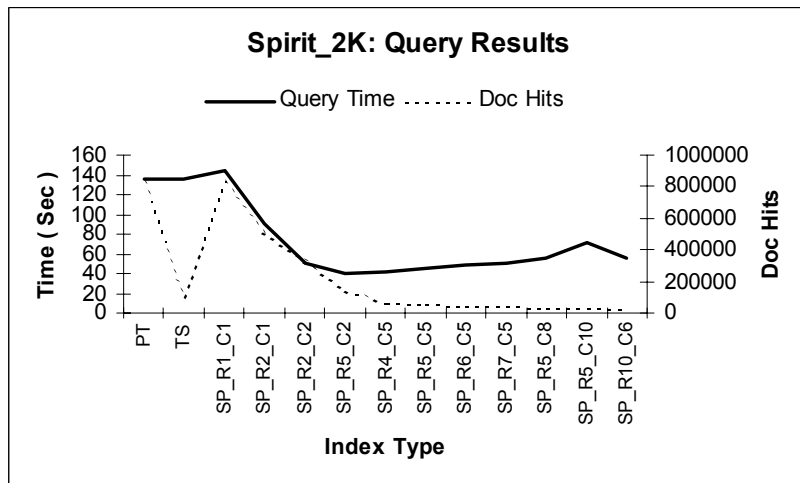


Figure 14. Spirit_2K: Query Results

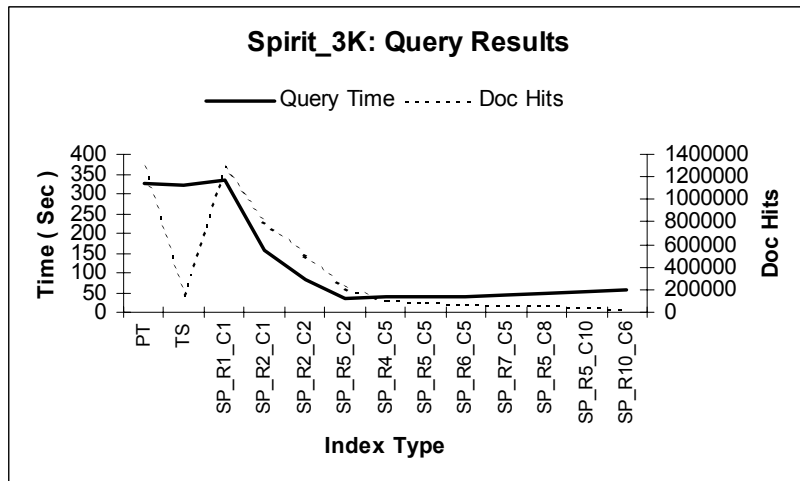


Figure 15. Spirit_3K: Query Results

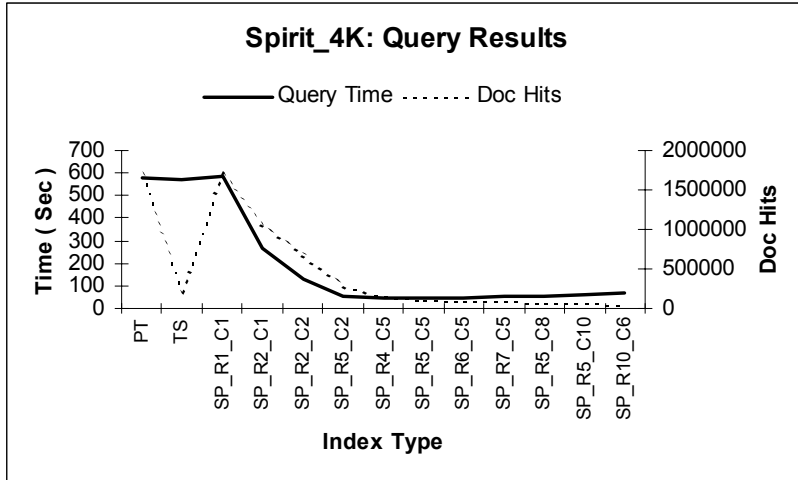


Figure 16. Spirit_4K: Query Results

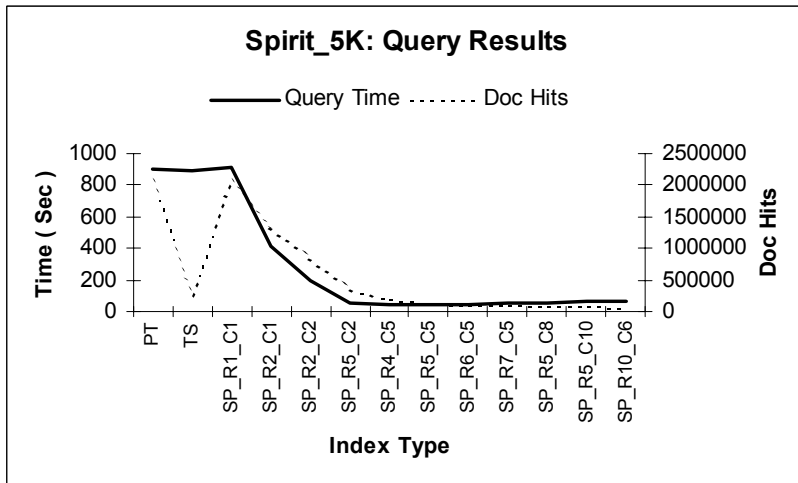


Figure 17. Spirit_5K: Query Results

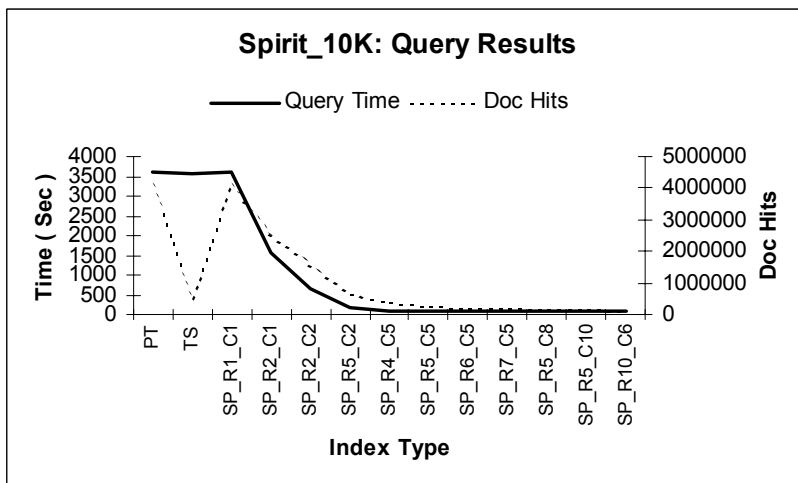


Figure 18. Spirit_10K: Query Results

Looking at Tables 4-9, it can be seen that after detailing basic dataset statistics in the table header, different index types and the corresponding costs and query times are presented in columns. We draw attention first to column 4, *ST Terms (Max)*. The maximum number of ST terms is obtained as a product of *T Terms* (Column 2) and number of spatial cells, *S Terms (Cells)* (Column 3) (See Section 6.2). Comparing columns 4 and 5, theoretically expected maximum and actual ST terms, for each of the datasets, we find that the actual number of text terms in the SP indices is far below the expected maximum. The graphs in Figures 7-12 depict the comparison more clearly. If we refer to Table 9 again, the number of TS terms for SP index with 60 spatial cells is 488316 where the normal textual index for this dataset contains 67857 textual terms. The number of TS terms is 7 times more than the text terms and not 60 times as was previously expected. This is a very positive observation in terms of memory consideration for spatio-textual indices. Another observation from above tables is the relatively small index time taken by SP indices in comparison to the textual index. This is not surprising because SP indices in our case were constructed by post-processing the textual index.

Looking again at Tables 4-9 and also Figures 13-18, we compare the query time and documents returned by all indices for each of the datasets. It can be easily seen that query time for PT and TS indices is far higher than most SP indices as the search space is the textual index of the entire collection. The only exception is SP index SP_R1_C1, but in this case there is only one spatial cell and the search space is identical to PT and TS indices i.e. the entire collection. The query time for TS index is slightly less than the PT index in all cases. Initially this may look slightly odd because TS index performs additional calculations for filtering out those documents, which do not intersect query's footprint. However, looking at number of documents returned for both these index types it can be seen that document hits after spatial filtering by TS index is far lower than the PT index. Thus the lower query time is perhaps due to fewer I/O operations in writing the results.

The query times for SP indices initially begins to fall with increasing resolution of the grid but then starts increasing gradually. The document hits keep falling with increasing grid resolution. This means we are likely to lose information if we refine the grid too much. Comparing both time and document hits for SP indices with TS index it appears that perhaps the best configuration of the grid is for SP_R4_C5 i.e. when the cell size is 5%. This appears to be a promising result worth further exploration with real data when we are also able to compare the quality of results.

Recall the discussion on spatial indices of type B and C in Section 5. It was difficult to prefer Spatial followed by Text or Text followed by Spatial. However, in the light of experimental results it appears that the number of spatial cells may be very small in comparison to the total number of textual terms. Thus it makes sense to keep textual index as primary and spatial as secondary as fewer I/O will be required.

8. CONCLUSION

This document is a preliminary investigation into the basic concepts that require familiarization in order to step into the design process of a search engine. The type of search engine being addressed in the SPIRIT project is one having capability of handling spatial queries for a collection of web documents. The research team has access to a text based information retrieval tool called GLASS [53], available at University of Sheffield. We aim to enhance GLASS's searching capabilities so as to handle spatial queries. With this aim, the working of web based search engines and several indexing structures were discussed. Considering the spatial query handling requirements of the project in mind, the overall emphasis was on discussing indexing techniques with a focus on spatial indexing methods. In the later part of the document, the case for a spatial index was argued. The principle is that the spatial index will serve as a filter for documents in the first stage and in the second stage a refined search

employing a textual index will be used. Alternatively, spatial indexing may be used as a secondary stage for filtering that follows an initial search with a text index.

A number of spatial indexing schemes were discussed. It was argued that a pure textual system might be inadequate for handling spatial queries, as it requires expensive geometrical calculations to cater to spatial coverage aspect of the query. Another argument for using this two stage spatial index as compared to textual only index appears to be the fact that it may not be feasible for the textual index to cope with a very large number of terms after query expansion.

In Sections 5 and 6, expected costs for spatial indices and empirical results were discussed. The experimental results build a strong case for spatial indexing. Spatial Indexing seems to be cutting down tremendously on the search times. However, it is difficult to comment on the accuracy/validity of results obtained in the absence of real-world data.

The best search times for spatial indexing were achieved when cell size was around 5%. There is need to look up further into this aspect by performing more experiments with larger data sets and grid resolutions in this region of 5%.

The experimental results with simple spatial index i.e. regular grid indicate to certain extent that if a two stage index is to be used, and the searching in the spatial index is efficient as compared to textual index, then the textual index can be made primary and spatial as secondary. We look forward to test this hypothesis by experimenting with regular grid based hybrid textual indexes and other advanced SAM such as quadtrees and R-trees using Oracle spatial or other public domain SAM software.

9. ACKNOWLEDGMENTS

Our thanks to Mark Sanderson and Hideo Joho at University of Sheffield for providing useful suggestions from time to time for this work. The authors are also grateful to Marc van Kreveld (Utrecht University) and Monica Sester (University of Hannover) for their valuable feedback.

10. REFERENCES

- [1] Search Engine Watch. <http://searchenginewatch.com/searchday/02/sd0617-update.html>
- [2] Jones, C.B. et al. Spatial Information Retrieval and Geographical Ontologies: An Overview of the SPIRIT project". *Proc. of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.387 – 388, Tampere, Finland, August 11-15, 2002.
- [3] Finch, D. J. An Overview of Search Engine Technology. July 2002, SPIRIT Document.
- [4] Franklin, C. How Internet Search Engines Work. <http://www.howstuffworks.com/search-engine.htm>
- [5] Search Engines. <http://www.lib.berkeley.edu/TeachingLib/Guides/Internet/SearchEngines.html>
- [6] Garcia-Molina, H., Ullman, J. D. and Widom, J. *Database Systems: The Complete book*. Prentice Hall, 2002.
- [7] Date, C. J. *An Introduction to Database Systems*, Volume 1 of *The systems programming series*. Addison-Wesley, 6th Edition, 1995.
- [8] Korth, H. F. and Silberschatz, A. *Database System Concepts*. McGraw-Hill, 2nd Edition, 1991.
- [9] Sedgewick, R. *Algorithms*. Addison-Wesley, 2nd Edition, 1988.

- [10] Guttman, A. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD*, pages 47–57, June 1984.
- [11] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B., The R*-tree: An efficient and robust access method for points and rectangles, *Proc. ACM SIGMOD*, pp. 322-331, Atlantic City, NJ, 23-25 May 1990.
- [12] Rigaux, P., Scholl, M. and Voisard, A.. Spatial Databases: With Application to GIS . *Morgan Kaufmann Publishers*, May 2001.
- [13] Sellis, T., Roussopoulos, N., and Faloutsos, C. The R₋tree: A Dynamic Index for Multidimensional Objects. In *Proceedings of 13th International Conference on VLDB*, pages 507–518, England, September 1987.
- [14] Comer, D. The Ubiquitous B-Tree. *ACM Computing Surveys*, 11(2):121–137, June 1979.
- [15] Nievergelt, J., Hinterberger, H., and Sevcik, K. C. The Grid File: An Adaptive, Symmetric, Multikey File Structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.
- [16] Samet, H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.
- [17] Samet, H. *Applications of Spatial Data Structures*. Addison-Wesley, 1990.
- [18] Bentley, J. L. and Friedman, J. H. Data Structures for Range Searching. *ACM Computing Surveys*, 11(4): 397–409, December 1979.
- [19] Samet, H. Hierarchical Representations of Small Rectangles. *ACM Computing Surveys*, 20(4):271–309, December 1988.
- [20] Samet, H. The Quadtree and Related Data Structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.
- [21] Gaede, V. and Guenther, O. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.
- [22] van Rijsbergen, C. J. Information Retrieval. *Butterworths*, London, England, 1979.
- [23] Faloutsos, C. and Oard, D. W. A Survey of Information Retrieval and Filtering Methods, in *Technical Report CS-TR-35 14*, University of Maryland, August 1995.
- [24] Baeza-Yates, R. A. Text Retrieval: Theory and Practice. *Proceedings of the 12th IFIP World Computer Congress*, 1992.
- [25] Salton, G. and McGill, M.J. Introduction to Modern Information Retrieval, *McGraw-Hill*, 1983
- [26] Salton, G. The SMART Information Retrieval System. *Prentice Hall*, Englewood Cliffs, NJ, 1971
- [27] Buyukokkten, O., Cho, J., Garcia-Molina, H., Gravano, L., Shivakumar, N. Exploiting geographical location information of web pages. *Proceedings of Workshop on Web Databases (WebDB'99) held in conjunction with ACM SIGMOD'99*, June 1999.
- [28] McCurley, K.S. Geospatial mapping and navigation on the web. in WWW10, 2001, <http://www10.org/cdrom/papers/278/>
- [29] [Geographic Search](http://dan.egnor.name/google.html). <http://dan.egnor.name/google.html>
- [30] [GeoURL ICBM Address Server](http://www.geourl.org/). <http://www.geourl.org/>
- [31] Bressan S., Ooi, B.C., and Lee, F. Global Atlas: Calibrating and Indexing Documents from the Internet in the Cartographic Paradigm. *Proceedings of the 1st International Conference on Web Information Systems Engineering*. Vol 1, pp. 117-124, 2000
- [32] [Getty Thesaurus of Geographic Names \(Research at the Getty\)](http://www.getty.edu/research/conducting_research/vocabularies/tgn/index.html). http://www.getty.edu/research/conducting_research/vocabularies/tgn/index.html
- [33] [SAND Internet Browser](http://www.cs.umd.edu/~brabec/sandjava). <http://www.cs.umd.edu/~brabec/sandjava>
- [34] Govindarajan, J., Ward, M. O: GeoViser - Geographic Visualization of Search Engine Results. *DEXA Workshop 1999*: 269-273
- [35] Google : Search by Location. <http://labs.google.com/location>
- [36] Berry, M. W., Dumais, S. T. and Letsche, T. A.. Computational Methods for Intelligent Information Access. <http://www.cs.utk.edu/~berry/sc95/sc95.html>
- [37] Mitra, M. and Chaudhuri, B.B. Information Retrieval from Documents: A Survey. *Information Retrieval*, 2, 141-163, 2000.
- [38] Goodrum, A. A.. Image Information Retrieval: An overview of Current Research. *Informing Science*, 3(2), 63-67, 2000.

- [39] Brin, S. and Page, L. The Anatomy of a Large-Scale Hypertextual Web Search Engine . *The 7th International World Wide Web Conference*. April 1998.
- [40] Kobayashi, M. and Takeda, K. Information Retrieval on the Web, *IBM Research Report*, RT0347, April 2000.
- [41] Huang, L. A Survey On Web Information Retrieval Technologies.
<http://citeseer.nj.nec.com/336617.html>
- [42] Arasu, A., Cho, J., Garcia-Molia, H., Paepcke, A., Raghavan, S. Searching the Web. *ACM Transactions on Internet Technology*.
- [43] Melnik, S., Raghavan, S., Yang, B., and Garcia-Molina, H. Building a distributed full-text index for the web. *In Proceedings of WWW10*, 2001.
- [44] GEOnet World Place Names Server. <http://164.214.2.59/gns/html/index.html>
- [45] Alexandria Digital Library Gazetteer Server. <http://fat-albert.alexandria.ucsb.edu:8827/gazetteer/>
- [46] Faloutsos, K., C. On Packing R-trees. In *Proceedings of the 2nd International Conference on Information and knowledge management*, pages 490–499, Arlington, VA, November 1993. Arlington, VA, November 1993.
- [47] Huang, Y.W., Jing, N., and Rundensteiner, E. A. A cost model for estimating the performance of spatial joins using R-trees. In *Statistical and Scientific Database Management*, pages 30-38, 1997.
- [48] Cutting, D. and Pedersen, J. Optimizations for dynamic inverted index maintenance, *Proceedings of the 13th International {ACM} {SIGIR} Conference on Research and Development in Information Retrieval*. 405—411, 1990.
- [49] MG Information Retrieval System. <http://www.cs.mu.oz.au/mg/>
- [50] Witten, I. H., Moffat, A., and Bell, T. C. Managing Gigabytes: Compressing and Indexing Documents and Images, *Morgan Kaufmann Publishing*, San Francisco, ISBN 1-55860-570-3. 1999
- [51] Zobel, J., Moffat, A., Ramamohanarao, K. Inverted Files Versus Signature Files for Text Indexing (1998). *ACM Transactions on Database Systems*, 23(4):453-490, December 1998.
- [52] Arampatzis, A. and van Kreveld, M. Simple and Useful Similarity Measures. D9:5102. SPIRIT Project. September 2003.
- [53] Sanderson, M. GLASS. <http://dis.shef.ac.uk/mark/GLASS>