ISSN No. 0976-5697

Volume 6, No. 8, Nov-Dec 2015

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Comparative Study of data structures for Proximity Search using Text Based Keywords

J.Sindhu M. Phil,
MCA Department,
Vels University, Chennai.
TamilNadu, India

R.Priya M.Phil, Ph. D
Assistant Professor, MCA Department,
Vels University, Chennai.
TamilNadu, India.

***Abstract*:** Conventional abstraction queries, like range search and nearest neighbour retrieval, involve only conditions on objects' geometric properties. Today, several modern applications involve novel varieties of queries that aim to seek out objects satisfying each abstraction predicate, and a predicate on their associated texts. As an example, rather than considering all the restaurants, a nearest neighbour question would instead extract the eating house that's the highest among those whose menus contain "steak, spaghetti, brandy" all at an equivalent time. Presently the most suitable resolution to such queries is predicated on the IR2-tree, which, as shown during this paper, features a few deficiencies that seriously impact its potency. Motivated by this, a replacement access methodology has been developed which is known as the abstraction inverted index that extends the standard inverted index to address flat knowledge, and comes with algorithms that may answer nearest neighbour queries with keywords in real time. As scrutinised by experiments, the projected approaches outgo the IR2-tree in question latent period considerably, typically by an element of orders of magnitude.

Keywords: proximity search, keyword search, spatial index,IR2-tree,GPS.

I. INTRODUCTION

A spatial database manages multidimensional objects (like points, rectangles, circles, squares), and gives faster access to the objects based on different selection criteria in the user query. The importance of spatial databases is reflected by the necessity of modelling entities of reality in a geometric manner. For example, spots of restaurants, hotels, hospitals and so on are mostly represented as points in a map, while larger locations such as parks, lakes, and landscapes are represented as a combination of symbols. Spatial database and its functionalities are useful in many different contexts. For instance, in a geographical information system (GIS), range search can be deployed to find all restaurants in a particular area, while proximity search can find the restaurant nearest to a given address or location.

The advent of Internet has given rise to an increasing amount of text data associated with multiple attributes, for example, customer reviews in e commerce websites (e.g., Flipkart) are always linked with dimensions like price, model, and rate. A traditional OLAP data cube can be easily made to aggregate and navigate structured data together with unstructured text data and that is called a text cube . A cell in the text cube combines a set of documents/items with matching attribute values in a subset of dimensions. Keyword based query, one of the most common , popular and easy ways to retrieve useful information from a collection of text documents, is being extended to RDBMSs to retrieve information from text-rich attributes. When a set of keywords is given, traditional methods aim to retrieve relevant data items or joins of data items that contain all or most of the keywords.

Spatial queries with keywords have not been widely explored. In the past years, the community has showed interest in studying keyword based search in relational databases. Very recently that attention was diverted to multidimensional data [5],[6],[8],[9].

II. PROBLEM DEFINITION

[9] Let P be a set of multidimensional points. As our aim is to merge keyword based search with the existing Geographical positioning services on facilities such as hospitals, restaurants, hotels, etc., we will focus on dimensionality two, but our technique can be extended to any dimensionalities with no technical obstacle. We will assume that the points in P have integer coordinates, such that each coordinate ranges in [0, t], where t is a large integer. This is not as restrictive as it may seem, because even if one would like to insist on real valued coordinates, the set of different coordinates representable under a space limit is still finite and listable therefore, we could as well convert everything to integers with proper scaling methods.

As said in [5] Each point p € P is associated with a set of words, which is denoted as Wp and termed the document of p. For example, if p stands for a restaurant, Wp can be its menu, or if p is a hotel, Wp is the description of its services

and facilities, or if p is a hospital, Wp can be the list of its out-patient specialities. It is clear that Wp may potentially contain many words

Traditional nearest neighbour search returns the data point closest to a query point. Following [5], We expand the problem to include predicates on objects' texts. Formally, in our context, a proximity search (NN) query specifies a point q and a set Wq of keywords (we refer to Wq as the document of the query). It returns the point in Pq that is the nearest to q, where Pq is defined as

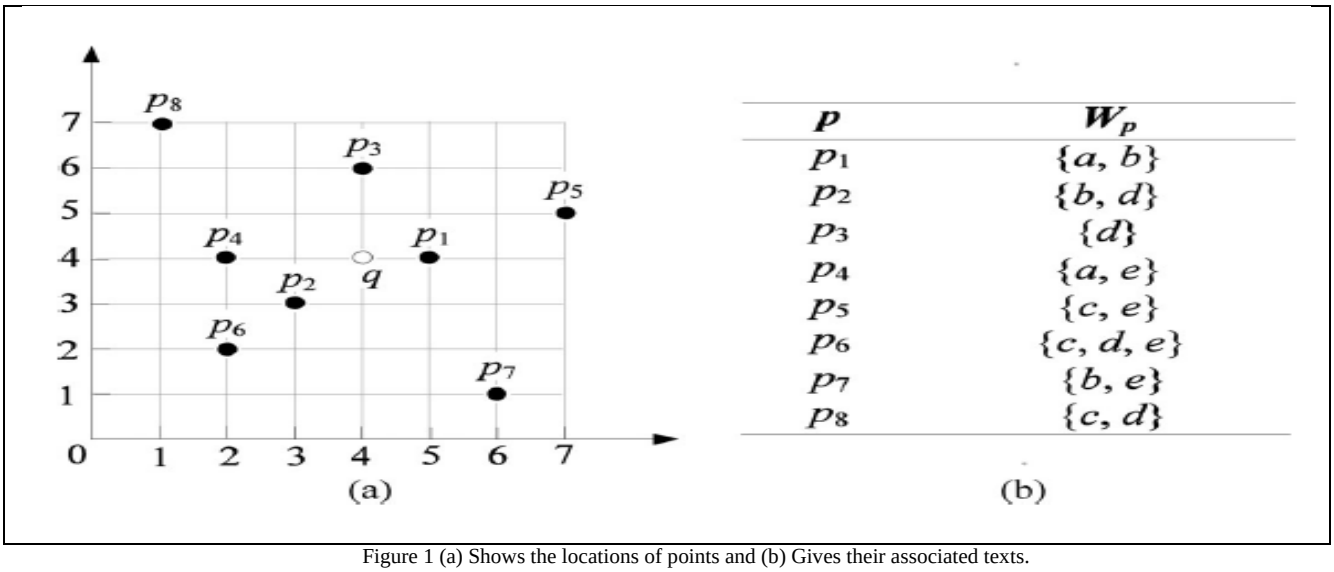
$$Pq = \{p \in P \mid Wq \in Wp\} \quad (1)$$

In other words, Pq is the set of objects in P whose documents has all the keywords in Wq. In the scenario where Pq is empty, the query returns nothing. The problem definition can be generalized to k nearest neighbour (kNN) search, which finds the k points in Pq nearest to q; if Pq has less than k points, the entire Pq should be returned.

For example, Let's consider that P consists of eight points whose locations are as shown in Fig.1.a. (the black dots), and their documents are given in Fig.1. b. Let's take a query point q at the white dot of Fig.1.a. with the set of keywords Wq = {c, d}. proximity search finds p6, noticing that all points closer to q than p6 are missing either the query keyword c or d. If k = 2 proximity neighbours are wanted, p8 is also returned in addition. The result remains as {p6, p8} even if k increases to 3 or higher, because only two objects have the keywords c and d at the same time. We consider that the data set does not fit in memory, and needs to be indexed by productive access methods in order to minimize the number of I/Os in answering a query

III. EXISTING SYSTEM

The advent of Internet has given rise to an ever increasing amount of text data associated with multiple dimensions (attributes), for example, customer reviews in shopping websites (e.g., flipkart) are mostly associated with attributes like cost and model of the product. A conventional OLAP data cube can be naturally extended to summarize and navigate structured data together with unstructured text data. Such a cube like model is otherwise known as text cube . A cell in the text cube sums up a set of documents/items with matching attribute values in a subset of dimensions. Keyword query, one of the very popular and easy-to-use ways to get useful information from a collection of plain documents , is being extended to RDBMSs to extract information from text-rich attributes . Given a set of keywords, existing methodologies aim to find appropriate items or joins of items (e.g., linked by foreign keys) that contain all or some of the given keywords



p	W_p
p_1	{a, b}
p_2	{b, d}
p_3	{d}
p_4	{a, e}
p_5	{c, e}
p_6	{c, d, e}
p_7	{b, e}
p_8	{c, d}

Figure 1 (a) Shows the locations of points and (b) Gives their associated texts.

Traditional IR methods can be used to rank documents according to the relevance. In a huge text database, however, the number of relevant documents to a query could be large, and a user has to spend much time reading them. Researchers have combined two well-known concepts: R-tree [1], a popular spatial index, and signature file [4], an efficient method for keyword-based document retrieval. In this progress they developed a structure called the IR2-tree [5] , which has the strengths of both R-trees and signature files. Like R-trees, the IR2-tree keeps track of objects' spatial proximity, which is the key to solving spatial queries efficiently. On the other hand, like signature files, the IR2-tree is able to filter a considerable part of the objects that do not contain all the query keywords, thus significantly reducing the number of objects to be examined.

[9] The IR2-tree, however, also incorporates a drawback of signature files: false hits. That is, a signature file, due to its restrictive nature, may still direct the search to some objects, even though they do not have all the keywords. The penalty thus caused is the need to verify an object whose

satisfying a query or not cannot be resolved using only its signature, but also requires loading its full text description, which is expensive due to the resulting random accesses. It is notable that the false hit problem is not specific only to signature files, but also exists in other methods for approximate set membership tests with compact storage (see [2] and the references therein).

Therefore, the problem cannot be solved by simply replacing signature file with any of those methods. For example, it would be fairly useful if a search engine is able to find the nearest restaurant that offers "steak, spaghetti, and brandy" all at the same time. Note that this is not the "globally" nearest restaurant (which would have been returned by a traditional proximal neighbor query), but the nearest restaurant among only those providing all the listed foods, drinks and other menu items.

Another method is inverted index. The inverted index data structure is a main and central component of a typical search engine indexing algorithm. The aim of a search engine performance is to optimize the speed of the query: find the documents in which the word occurs. When an index is developed, which provisions lists of words per document; it is next inverted to develop an inverted index. Querying the index would require sequential iteration through each document and to each and every word to verify a matching document. The time memory and processing property to run such a query are not always theoretically realistic. Instead of enumerating the words per article in the index, the inverted index data structure is developed which lists the documents per word. The inverted index produced, the query can now be identified by jumping to the word id in the inverted index. These were effectively inverted indexes with a small amount of additional explanation that required an implausible amount of attempt to produce.

word	inverted list
a	p_1 p_4
b	p_1 p_2 p_7
c	p_5 p_6 p_8
d	p_2 p_3 p_6 p_8
e	p_4 p_5 p_6 p_7

Figure 2 Example of an inverted index.

IV. PROPOSED SYSTEM

Spatial Inverted Index

In this paper, based on the experiments of [7] we propose a variant of inverted index that is optimized for multidimensional points, and is thus named the **spatial inverted index (SI-index).** This access method perfectly incorporates point coordinates into a conventional inverted index with small extra space, owing to a delicate compact storage scheme. The compression removes the defect of a conventional I-index such that an SI-index consumes much less space.

Compression Of SI-Index

Compression is used to decrease the size of an inverted index where each inverted list contains only ids. In that case, an effective approach is to record the gaps between successive ids, as opposed to the corresponding ids. For example, given a set S of integers {2, 3, 6, 8}, the gap-keeping approach will store {2, 1, 3, 2} instead, where the ith value (i >= 2) is the difference between the ith and (i-1) th values in the original S. As the original S can be precisely recreated, no information is lost. The only overhead is that decompression incurs additional computation cost, but such cost is negligible compared to the overhead of I/Os. Note that gap- keeping will be much less useful if the integers of S are not in a sorted order. This is because the space saving is derived from the hope that gaps would be much smaller (than the original values) and hence could be represented with fewer bits. This would not be true if S is not been sorted.

Compressing an SI-index is not that straightforward. The difference here is that each element of a list, a.k.a. a point p, is a triplet {idp, xp, yp}, which includes both the id and coordinates of p. As gap-keeping requires a sorted order, it can be extended on only one attribute of the triplet. For example, if we decide to sort the list by ids, gap-keeping on ids may result in good space saving, but its application on the x- and y-coordinates would not have much effect.

To confront this problem, let us first leave out the ids and focus on the coordinates. Even though each point contains two coordinates, we can convert them into only one so that gap keeping can be applied effectively. The tool required is a space filling curve (SFC) such as Hilbert- or Z-curve. SFC converts a multidimensional point to a 1D value such that if two points are very close and near in the original space, their 1D values also tend to be similar. As dimensionality has been reduced to 1, gap-keeping works effectively after sorting the (converted) 1D values.

p_6	p_2	p_8	p_4	p_7	p_1	p_3	p_5
12	15	23	24	41	50	52	59

Figure 3 Converted values of the points in fig.2 based on Z-curve.

For example, based on the Z-curve, the resulting values, called Z-values, of the points in Fig.1.a are clearly and precisely denoted in Fig. 3 in increasing order (ascending). With gap-keeping, we will save these 8 points as the sequence 12, 3, 8, 1, 7, 9, 2, 7. Note that as the Z-values of all points can be precisely restored, the exact coordinates can be restored as well.

Let us put the ids back into focus . Now that we have successfully dealt with the two coordinates with a 2D Space Filling Curve (SFC) , it would be natural to consider using a 3D SFC to cope with ids too. As far as space reduction is concerned, this 3D method may not be a bad solution. The problem is that it will ruin the locality of the points in their original space. Specifically, the converted values would no maintain the spatial proximity of the points, because ids in general have nothing to do with coordinates.

If one thinks about the purposes of having an id, it will be clear that it essentially gives a token for us to extract (typically, from a hash table) the details of an object, e.g., the text description and/or other attribute values. Furthermore, in responding to a query, the ids also provide the base for merging. Therefore, nothing stops us from using a pseudo-id internally. Specifically, let us forget about the "real" ids, and instead, associate each point to a pseudo-id that equals its sequence number in the ordering of Z-values. For example, as per the Fig.3, p6 gets a pseudo-id 0, p2 gets a 1, and so on. Explicitly, these pseudo-ids can co-exist with the "real" ids, which can still be kept along with objects' details.

The benefit we get from pseudo-ids is that sorting them provides the same ordering as sorting the Z-values of the points. This means that gap-keeping will work simultaneously on both the pseudo-ids and Z-values. As an example that gives the full scenario, consider the inverted list of word d in Fig.2 that has p2, p3, p6, p8, whose Z-values are 15, 52, 12, 23 respectively, with pseudo-ids being 1, 6, 0, 2, respectively. Sorting the Z-values automatically also keeps the pseudo-ids in ascending order. With gap-keeping, the Z-values are stored as 12; 3; 8; 29 and the pseudo-ids as 0, 1, 1, 4. So we can exactly capture the four points with four pairs: {(0, 12), (1, 3), (1, 8), (4, 29)}.

Since SFC applies to any dimensionality, it is straight forward to extend our compression scheme to any dimensional space. As a remark, we are aware that the ideas of space filling curves and internal ids have also been mentioned in [3] (but not for the purpose of compression).

Blocking An SI-Index

The SI-index described up to now applies gap keeping capturing all points continuously in a row. In decompressing, we must scan an inverted list from its beginning even though the point of our interest is situated deep down the list (remember that a point cannot be restored without all the gaps preceding it being accumulated). This is not a complication for a query algorithm that performs sequential scan on the list. But in some of the significant cases (e.g., when we would like to build an R-tree on the list), it is very useful to restore a point anywhere in the list much faster than reading from the beginning every time.

The above concern motivates the design of the blocked SI-index in [7] and which differs only in that each list is split into blocks each of which holds points where B is a parameter to be specified later. For example, given a list of {p1, p2, p3, p4 , p5, p6}, we would store it in two blocks {p1, p2, p3} and {p4, p5, p6} if the block size is 3. Gap-keeping is now implemented within each block separately.

For example, in block {p1, p2, p3}, we will store the exact pseudo-id and Z-value of p1, the gaps of p2 (from p1) in its pseudo-id and Z-value, respectively, and similarly, the gaps of p3 from p2. Evidently, blocking allows restoring all the points in a block locally, as long as the starting address of the block is available. It is no longer mandatory to always scan from the beginning.

Meanwhile, an SI-index keeps track of the spatial locality of data points, and comes with an R-tree built on every inverted list at little space overhead. Every block in the SI-Index is a leaf node in the R-tree built on the inverted list. As a result, it offers two competing ways for query processing (merging and distance browsing manner). We can (sequentially) combine multiple lists very much like merging traditional inverted lists by ids. Alternately, we can also leverage the R-trees to browse the points of all relevant lists in ascending order of their distances to the query point.

V. COMPARATIVE STUDY

A Comparative study based on the experiments in [7], is explained below,

Competitors. The proposed SI-index comes with two query algorithms based on merging and distance browsing respectively. We will refer to the former as SI-m and the other as SI-b. Our evaluation also covers the state-of-the-art IR2-tree; in particular, our IR2-tree implementation is the fast variant developed in [5], which uses longer signatures for higher levels of tree.

Furthermore, we also include the method, named index file R-tree (IFR) henceforth, which indexes each inverted list (with coordinates embedded) using an R-tree, and applies distance browsing for query processing. IFR can be regarded as an uncompressed version of SI-b.

Data. Experiments are done using both synthetic and real data. The dimensionality is invariably 2, with each axis containing integers from 0 to 16; 383. The synthetic data has two data sets: Uniform and Skew, which vary in the distribution of data points, and in whether there is a correlation between the spatial distribution and objects' text documents. Specifically, each data set comprises 1 million points. Their locations are uniformly spread in Uniform, whereas in Skew, they maintain the Zipf distribution.3 for

both data sets, the vocabulary has 200 words, and each word appears in the text documents of 50k points. The difference is that the association of words with points is totally random in Uniform, while in Skew, there is a pattern of "word-locality": points that are spatially proximal have almost identical text documents.

3. We create each point exclusively by generating each of its coordinates (again, independently) according to Zipf.

Table 1. Data Set Statistics

	number of points	vocabulary size	average number of objects per word	average number of words per object
Uniform	1 million	200	50k	10
Skew	1 million	200	50k	10
Census	20847	292255	33	461

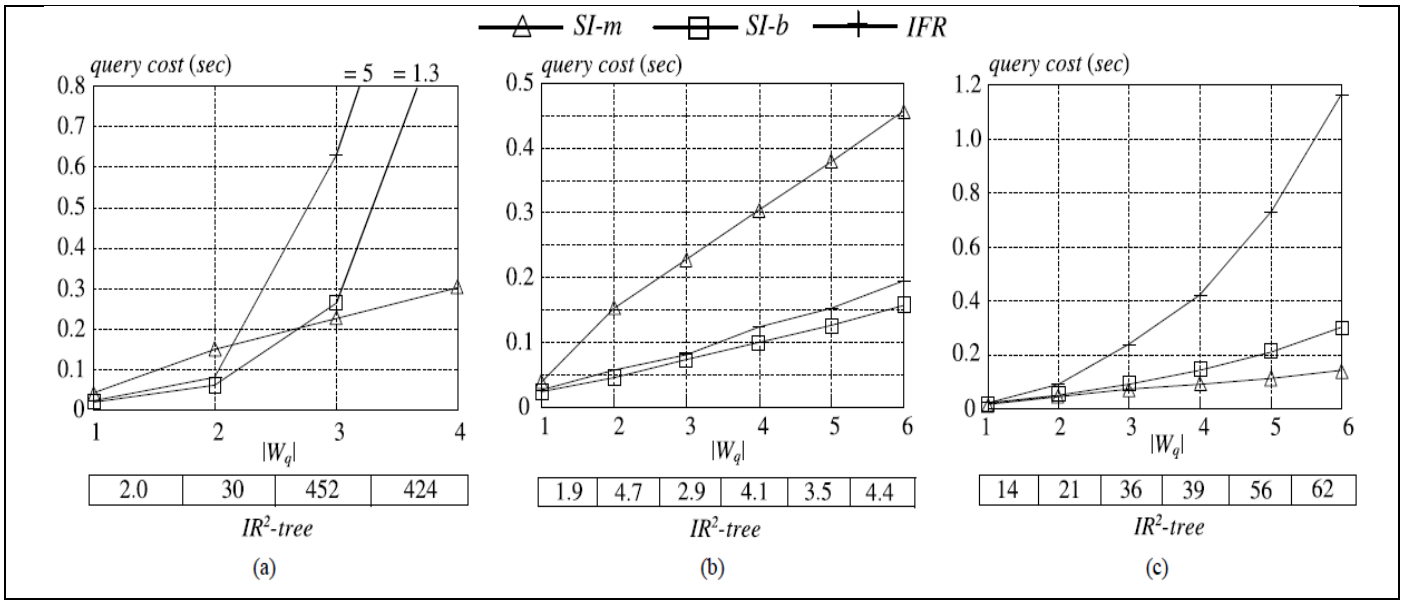


Figure 4. Query time versus the number of keywords |Wq|: (a) Data set Uniform, (b) Skew, (c) Census. The number k of neighbours retrieved is 10.

Our real data set, referred to as Census below, is a combination of a spatial data set published by the US Census Bureau,4 and the web pages from Wikipedia.5 The spatial data set has 20,847 points, each of which represents a county subdivision. We use the name of the subdivision to search for its corresponding page at Wikipedia, and collect the words there as the text description of the corresponding data point. All the points, and their text documents, makeup the data set Census. The main statistics of all of our data sets are summed up in Table 1.

Parameters.The page size always remains as 4,096 bytes. All the SIindexes holds a block size of 200 (see Section 6.1 for the meaning of a block). The parameters of IR2-tree are kept in exactly the same way as in [5]. Specifically, the tree on Uniform extends ro 3 levels, whose signatures (from leaves to the root) have respectively 48, 768, and 840 bits each. The respective lengths for Skew are 48, 856, and 864. The tree on Census contains two levels, whose lengths are 2, 000 and 47, 608, respectively.

4. http://www.census.gov/geo/www/gazetteer/places2k.html, and follow the link "County Subdivisions".
5. http://en.wikipedia.org.

Queries.As in [5], we consider NN search with the AND semantic. There are 2 query parameters: (i) the number k of neighbours requested, and (ii) the number |Wq| of keywords. Each workload contains 100 queries that have the same parameters, and are generated independently as follows. First, the query location is uniformly distributed in the data space. Second, the set Wq of keywords is a random subset (with the designated size |Wq|) of the text description of a point randomly sampled from the underlying data set. We will measure the query cost as the total I/O time (in our system, on average, almost every sequential page access time takes about 1 milli-second, and a random access is around 10 times slower).

Results on query efficiency. Let us start with the query performance with respect to the number of keywords |Wq|.

For this purpose, we will keep the parameter k to 10, i.e., each query extracts 10 neighbors. For each alternative method, we will report its average query time in processing a workload. The results are explained in Fig. 4, where (a), (b), (c) are about data sets Uniform, Skew, and Census, respectively. In each case, we provide the I/O time of IR2-tree separately in a table, because it is significantly more expensive than the other solutions. The experiment on Uniform inspects |Wq| up to 4, because almost all queries with greater |Wq| return no result at all.

The fastest approach is either SI-m or SI-b in all cases. In particular, SI-m is especially efficient on Census where each inverted list is relatively small (this is implicitly shown from the column "the number objects per word" in Table 1), and hence, index-based search is not as effective as simple scans. The behaviour of the two algorithms on Uniform very well confirms the intuition that distance browsing is more suitable when |Wq| is small, but is outperformed by merging when Wq is sizable. On Skew, SI-b is significantly better than SI-m because of the "word-locality" pattern. As for IFR, its behaviour in general follows that of SI-b because they differ only in whether compression is performed. The superiority of SI-b derives from its larger node capacity.

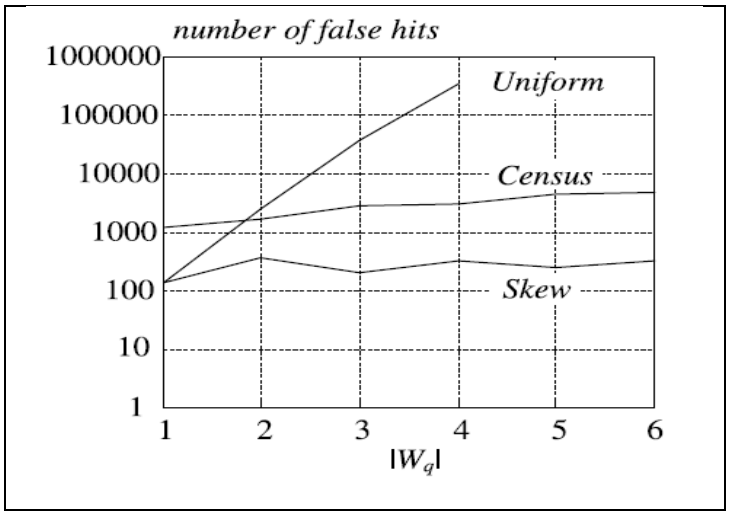


Figure 5.Number of false hits of IR2-tree.

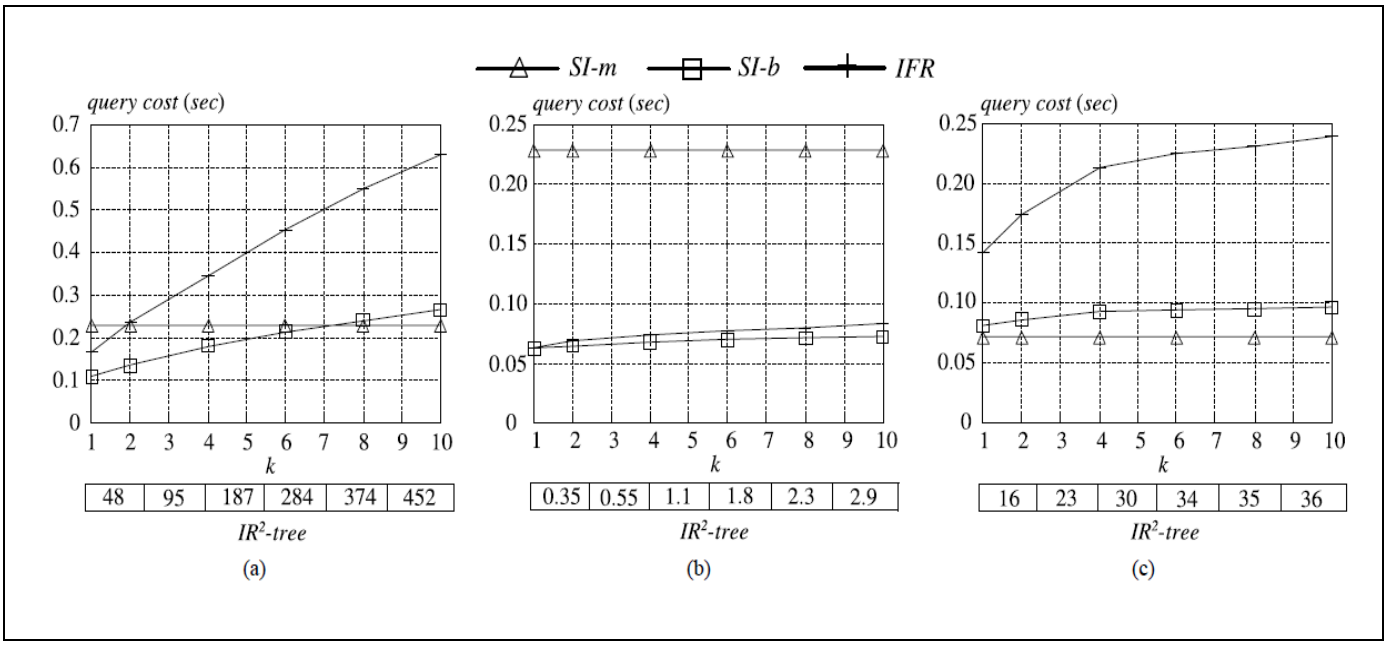


Figure. 6. Query time versus the number of k of neighbours returned: (a) data set Uniform, (b) Skew, (c) Census. The number |Wq| of query keywords is 3.

IR2-tree, on the other hand, fails to give real time answers, and is often slower than our results by a factor of orders of magnitude, particularly on Uniform and Census where word-locality does not exist. As analyzed earlier, the deficiency of IR2-tree is mainly caused by the need to verify a vast number of false hits. To explicitly demonstrate this, Fig. 5 plots the average false hit number per query (in the experiments of Fig. 4) as a functionality of |Wq|. We see an rapidly increasing escalation of the number on Uniform and Census, which explains the drastic explosion of the query cost on those data sets. Interesting is that the number of false hits oscilates[6] a little on Skew, which explains the fluctuation in the cost of IR2-tree in Fig. 4b.

6. Such fluctuation is not a surprise because, as discussed earlier, the number of false hits is determined by two factors that may cancel each other: (i) how many data points are in proximity than the kth NN reported, and (ii) the false hit probability. While the former factor increases with the growth of |Wq|, the latter actually decreases.

Next, we shift to study the other query parameter k (the number of neighbors returned). The experiments for this purpose are based on queries with |Wq| = 3. As before, the average query time of each approach in handling a workload is reported. Figs. 6a, 6b, and 6c give the results on Uniform, Skew, and Census, respectively. Once again, the costs of IR2-tree are segregated into tables. In these experiments, the best method is still either SI-m or SI-b. As expected, the cost of SI-m is not affected by k, while those of the other solutions all increase monotonically. The relative superiority of competing methods, in general, is similar to that exhibited in Fig. 4. Perhaps worth noting down is that, for all distributions, distance browsing appears to be the most efficient approach when k is small.

Results on space consumption. We will conclude our experiments by reporting the space cost of each method on each data set. While four methodologies are scrutinised in the experiments on query time, there are only three as far as space is concerned. Remember that SI-m and SI-b actually implement the same SI-index and hence, have the same space cost. In the following, we will refer to them on the wholesome as SI-index.

Fig.7 gives the space consumption of IR2-tree, SI-index, and IFR on data sets Uniform, Skew, and Census, respectively. As expected, IFR incurs prohibitively large space cost, because it needs to duplicate the coordinates of a data point p as many times as the number of unique words in the text description of p. As for the other methods, IR2-tree appears to be slightly more space efficient, although such a benifit does not justify its expensive query time, as shown in the earlier experiments.

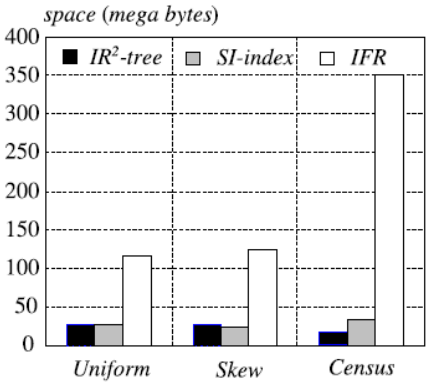


Figure. 7. Comparison of space consumption.

Summary. Based on experiments R tree incurs prohibitively large space cost, because it needs to duplicate the coordinates of a data point p as many times as the number of distinct words in the text description of p. As for the other methods, IR2-tree appears to be slightly more space efficient, although such an advantage does not justify its expensive query time.

The SI-index, accompanied by the proposed query algorithms, has presented itself as an excellent trade-off between space and query efficiency. Compared to IFR, it consumes significantly less space, and yet, answers queries much faster. Compared to IR2-tree, its superiority is overwhelming since its query time is typically lower by a factor of orders of magnitude.

VI. CONCLUSION

We have seen many applications calling for a search engine that is able to efficiently support novel forms of spatial queries that are integrated with keyword search. The existing solutions to such queries either incur space consumption or are unable to give real time solutions. The Spatial Inverted -index is fairly space economical, also it has the ability to perform keyword-based proximity search in time that is at the order of dozens of milliseconds. Furthermore, as the Spatial Inverted-index is based on the traditional technology of inverted index, it is quickly incorporable in a commercial search engine that involves massive parallelism, implying its immediate merits.

The future enhancements includes extending the proximity search with wide range of search parameters e.g. Restaurant themes , current seat availability in the restaurant, etc. Addition of parameters leads to increase in response time and space complexity. Compression schemes can be enhanced to overcome the added overhead.

VII. ACKNOWLEDGMENT

I express my sincere thanks to **Dr. ISHARI K GANESH**, the Chancellor and Founder of Vels University **Dr. V.TAMIZHARASAN**, Vice-Chancellor, Vels University for providing me necessary facilities. Also I wish to extend my heartfelt thanks to **Dr.B.KRISHNAMURTHY**, The Registrar of Vels

University. I express my deep and sincere gratitude to the Head of the Department, **Dr.P.MAYILVAHANAN** MSc., ME., MPhil., PhD., for his valuable suggestions towards completion of project successfully. I also thank, **Mrs.R.PRIYA** MCA, MPhil, (PhD)., Vels University for her valuable guidance throughout my research work. I also thank my parents, well wishers and friends who extended their valuable cooperation throughout my thesis work.

VIII. REFERENCES

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

[2] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The bloomier filter: an efficient data structure for static support lookup tables. In Proc. of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 30–39, 2004.

[3] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In Proc. of ACM Management of Data (SIGMOD), pages 277–288, 2006.

[4] C. Faloutsos and S. Christodoulakis, “Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation,” ACM Trans. Information Systems, vol. 2, no. 4, pp. 267-288,1984.

[5] I.D. Felipe, V. Hristidis, and N. Rishe, “Keyword Search on Spatial Databases,” Proc. Int’l Conf. Data Eng. (ICDE), pp. 656-665, 2008.

[6] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, “Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems,” Proc. Scientific and Statistical Database Management(SSDBM), 2007.

[7] Fast nearest neighbour browsing and search ,Yufei Tao and Cheng Sheng ,2014.

[8] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma, “Hybrid Index Structures for Location-Based Web Search,” Proc. Conf. Information and Knowledge Management (CIKM),pp.155-162,2005

[9] R.Pawar Anita, B.Pansare Rajashree, H.Mulani Tabbsum, B.Bandgar Shrimani “Designing of Semantic Neighbour Search : Survey” International Journal of Computer Applications Technology and Research (IJCATR), Volume 4-Issue 1 pp.53-57,2015

