Evaluating Spatial Keyword Queries under the MapReduce Framework*

Wengen Li[1], Weili Wang[1], and Ting Jin[2]

[1] Dept. of Computer Science and Technology, Tongji University, Shanghai, China
{lwengen,ken.wlwang}@gmail.com
[2] School of Computer Science, Fudan University, Shanghai, China
tingj@fudan.edu.cn

Abstract. Spatial keyword queries, finding objects closest to a specified location that contains a set of keywords, are a kind of pervasive operations in spatial databases. In reality, there is some spatial data that is not stored in databases, instead in files. And generally this kind of spatial data is textual, noisy, large-scale and used now and then, which makes it be quite costly to conduct spatial keyword querying on such spatial data. To solve this problem, in this paper we propose an efficient method by using a distributed system based on MapReduce. The algorithm for spatial keyword query evaluation under the MapReduce model is developed and implemented. Experimental results demonstrate that this method can process spatial keyword queries effectively and efficiently.

Keywords: Spatial keyword query, MapReduce, Hadoop, HDFS.

1 Introduction

Spatial keyword queries [1] are a widely used in spatial databases. The goal is to find some objects that are closest to the query location and contain the specified set of keywords. With the proliferation of global positioning systems (GPSs) and their applications, more and more objects on the Web are associated with geographical locations, in addition to textual labels. This development gives significance to spatial keyword queries.

In practice, there are application scenarios like this. We have large scale spatial datasets that are not stored in databases, and we want to find some specific objects from the datasets within a limited time. In addition, every object has spatial attributes and textual attributes. For example, the geographic objects recorded by the U.S. Board on Geographic Names are just stored as text files. We call this kind of spatial data none-in-database spatial data.

The existing methods for spatial keyword queries have limitations to deal with the none-in-database data. First, it is highly expensive to put the data into databases. Because the data can be dirty, type mixed and unstructured. So a

* This work was supported by National Natural Science Foundation of China under grant No. 60873040.

H. Yu et al. (Eds.): DASFAA Workshops 2012, LNCS 7240, pp. 251–261, 2012.

lot of work has to be done before we can put the data into databases, including extracting useful attributes, deleting meaningless objects, checking the attributes with null value etc. Second, queries are issued now and then, which means that most time we just leave the data there without operations. So there is no need to spend a lot of time and other resources to maintain it. What is more, the index for the data will be quite large and a lot disk space is needed to store the data and indices. Last but not least, the more result objects are required, the more cost has to pay. So it is important to find new techniques to deal with big spatial data over which spatial keyword queries can be conducted efficiently without index support. The MapReduce [2] is a good choice.

MapReduce is a distributed programming model put forward by Google, which is one of google's core technologies. MapReduce is designed to process large-scale datasets by dividing the data into many small blocks and dealing with them in parallel, which can reduce the run time substantially. In this work, we explore effective methods to efficiently support spatial keyword queries on none-in-database spatial data by using MapReduce.

The rest of this paper is organized as follows. Section 2 gives a description of the MapReduce programming model and introduces spatial keyword queries. Section 3 introduces how to evaluate spatial keyword queries based on MapReduce. Section 4 is experimental evaluation. Finally, Section 5 concludes the paper.

2 Preliminaries

2.1 The MapReduce Programming Model

MapReduce is a widely used programming model designed to process large scale of data on distributed system. The main idea of MapReduce is to divide the original big data into many small blocks and then process them on different machines at the same time. Usually the size of a block can be up to 64M which is much larger than the disk page. MapReduce can implement this with the help of HDFS which is the open source realization of GFS [3], a distributed file system.

MapReduce processes the data through two phases, i.e., map phase and reduce phase. In map phase the main task is to obtain useful information from the data file and form the key/value pair which is an important feature in MapReduce. In general, one map task takes care of one block and the output of the map task is divided into R pieces. Here the value of R can be defined by the user. In the second phase a reduce task takes a piece of data from every map task output and then put them together. In the reduce phase the objects with the same key are going to be combined together. Between the two phases there are some other important operations, like remote read, shuffle and sort. At last R final files will be produced by R reduce tasks. Fig. 1 shows the implementation of MapReduce [4].

Like the figure describes it is quite clear for programmers to program their own distributed programs by MapReduce. But in practice there is still a lot of work to do, like the data division design, the map operation, the shuffle operation, the sort operation and the reduce operation.

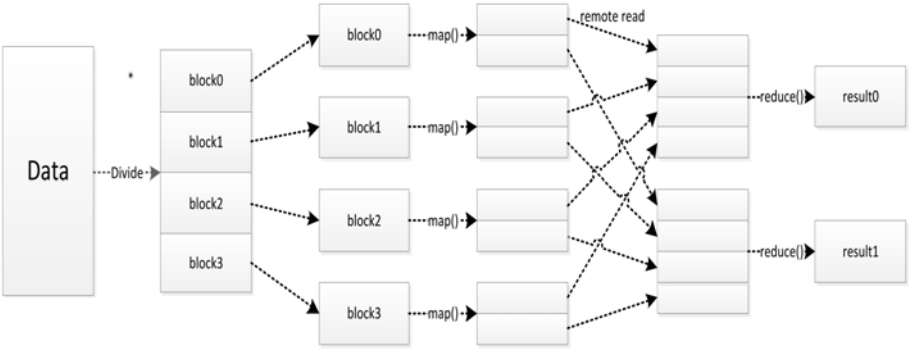


Fig. 1. The workflow of MapReduce

2.2 Spatial Keyword Queries

Spatial keyword query is a widely used query in spatial database. Many applications require spatial keyword query operation. For example, one may want to find the nearest supermarket which sells shoes from his or her home.

Assume that D is a spatial database. Each object o in D has the form $o(o.ID, o.s, o.t)$, where $o.ID$ is the identifier, $o.s$ is the spatial attributes and $o.t$ is the keywords list. The query condition is $q(k, q.s, q.t)$, where the $q.s$ and $q.t$ are the same with $o.s$ and $o.t$. The value of k in the query condition is based on the user's requirement for the number of final objects.

Specially, when k is equal to 1 spatial keyword query can be defined like this. The result object should have the keywords in the $q.t$, which means $q.t$ should be a subset of $o.t$. And also the object should have $Min\ (dist\ (o.s,\ q.s))$ in the database D. So the spatial keyword query can be defined as follows.

$$(r.t \subseteq q.t) \wedge dist(r.s, q.s) = min(dist(o.s, q.s), \forall o \in D). \tag{1}$$

The object r is the result.

If more objects are wanted, we can just do a kNN query [5,6] based on $dist(o.s, q.s)$.

The straightforward method for spatial keyword queries is to check all the objects one by one until the right answer is found, which will lead to very high cost. The practical method is to construct some kind of index structure for spatial database. The common index structure for spatial data is R-tree [7] and its variations, like R*-tree [8], packed R-tree [9]. And the common index structure for keywords is inverted file and signature file [10]. The original way for spatial keyword query is to search the spatial attributes and keywords respectively and then combine them together. This can results in some unexpected problems like how to combine them and how to make sure the final result is right. Then researchers come up with IR-tree [11], IR2-tree [12] and BR*-tree [13], and the main idea is to create hybrid index [14] for both spatial and textual attributes.

But these methods have some disadvantages to deal with none-in-database data which is described in the introduction. So we try to combine the spatial keywords query with MapReduce. And MapReduce can deal with these disadvantages well.

3 Spatial Keyword Queries Processing under MapReduce

3.1 Problem Definition

In the real world some spatial data are not stored in the database for some reason. For example, the location information of millions of buildings in the past 30 years is just stored as textual data. Another example is the temperature recording which has spatial feature. The temperature recording has a long history, may be from the early 20th century or even earlier. For many countries the temperature recording is stored as text file.

Both of the above two kinds of data have the following features: 1) Dirty: The data can be in a mess. In general the history of the data may be quite long. During the period some information could have been lost. And another possibility is that the data format at different time can be different. 2) Textual: The data is stored as a text file and has no existed standard structure to maintain it. 3) Big: The data can be quite large. We need to store all the information for the buildings or the temperature recording. So the data can be dozens of GB or even TB. 4) Not in database: The data is just in the text file. It is not stored in some kind of database. 5) Used now and then.

As history data it is used only when someone wants to have a study on it. At most of the time, the data is just stored there. We may have the following application requirements on this kind of data: 1) Obtain one spatial keyword query result object; 2) Obtain k spatial keyword query result objects; 3) Obtain all the objects which have the given keywords.

We can define them formally. Every object has the form $o(o.ID,\ o.s,\ o.t,\ o.m)$, where $o.ID$, $o.t$ and $o.t$ are the same with the section 3. Here o.m represents some other unconcerned information. The query is $q(k,\ q.s,\ q.t)$.

For the above three application requirements there may be some requirements. For example, the result should be returned in a tolerant time. And the cost should be reasonable. This makes it impossible for us to analyze the data and put them in the database and create an efficient index structure for it. So here the distributed model MapReduce is employed to solve these problems.

We divide the whole process into six steps, which are respectively as follows: 1) Data division; 2) Key/value pair extraction; 3) Keyword check and spatial attributes extraction; 4) Distance computation; 5) Partition and sorting; 6) Obtain the result.

The Step 1) is responsible for dividing the input data set into many small blocks. Steps 2)–4) belong to the map phase and Step 6) is the reduce phase, and Step 5 is the middle operation between the map phase and the reduce phase.

3.2 Data Division

The original data need to be divided into blocks before the map operation. Because the data can be quite large, may be terabytes, or even petabytes. So in practice it is extremely difficult for one computer to process it. We need to divide the data into many blocks and put them in different machines to process. In MapReduce the size of each block can be 16M to 64M, or bigger. Here we choose 64M.

If the input file is just text type then we just divide it into blocks from the preliminary to the end. This is quite simple. For example, if the file is 200M we divide it into four blocks and the first three are 64M while the last one is only 8block. One thing need to notice is that two blocks may share a line. It means some objects may belong to two blocks. In this situation we have to read the two blocks when the object is needed.

If the input file has been compressed we need to consider whether the compression format support division. If the compression format is ZIP or bzip2 then we can just divide it like the text type file. But some compression formats, like DEFLATE, Gzip, LZO, do not support division. In this situation we have two ways to process it. The first one is to extract it. After that we can process it like the common file. Another way is just give it to a map task which means the map task has to process a big file. The second way goes away from our original intension. So it is used now and then.

3.3 Key/Value Pair Extraction

Before we begin to extracting the key/value pair we need to describe the format of our data. It will be more convenient for us to show our idea if the data format can be fixed although the data can be of all kinds of formats.

We assume the data is comprised of many objects and each object is a line. Every object consists of four parts. One is the *ID* which identifies the object. The second part is the spatial attributes which can be any dimensional. For easy to present the process procedure we just consider two dimensions. So the spatial attributes can be just represented with a two-tuples (x, y). In practice the spatial can be hid in the object and we need more effort to find it out. The third part of the related keywords and the fourth pare is some other information.

Here we assume the every line is a object. If the data is not copperplate we can just use additional operation to clear it.

So the key/value pair can be defined like this. The key value is the offset of each line and the value is the content of the line or the object.

3.4 Keyword Check and Spatial Attributes Extraction

In spatial key word query it is quite difficult to take both spatial attributes and textual attributes into account. The main challenge is that they are two different kinds of data and they have different features. The traditional way to is to process the spatial attributes and textual attributes separately and combine the two

results together. But it is quite difficult to make the combination. Another way is create hybrid index which has both the spatial attributes and the keywords. Here we do not have the index and we do not have the necessary to create it. So the original way is used to process the data. By using MapReduce the disadvantage of the original way can be avoided. Because every block is just 64M and it will be quite easy to traverse it.

If we just want to obtain the object which has all the required keywords we can scan the block to check it. The object can be a candidate if and only if all the required keywords are included by it. If we do not need the object to include all the keywords or we have other requirements on the keywords we can just define a function and use it to choose the candidates.

Then it is needed to obtain the spatial attributes from the candidates and turn them into standard format. Every candidate is a line. It can consist of all kinds of attributes. So we need to spend much effort to analyze it and find out the spatial data.

After that we can calculate the spatial distance between the candidate and the query point.

3.5 Distance Computation

The distance between the candidate and the query point can be the simple Euclidean distance, or some other distance like Manhattan distance. User only needs to describe the way to calculate the distance. MapReduce can do it on every candidate for you.

After that it is needed to point out the output format which is another format of key/value pair. Here the distance is used to be the key because we need to obtain result by distance, and the content of the object is the value.

3.6 Partition and Sorting

A big part of the objects which do not have the keywords can be pruned through the map phase but there may be still a lot of candidates. So we need to have an efficient way to deal with this problem. In MapReduce we can use more than one reduce task to process the output from map tasks. The key point is how to distribute the output to the reduce tasks.

There are two challenges for partition. First we should make sure that the output of each reduce work can combine to be a sorted file. This requires the partition strategy to put the data into reasonable groups. Secondly, we should make sure the partition is fair for each reduce work. Because it is unreasonable to ask one reduce task to do the 90% of the work and the others only need to process the rest. This can reduce in bad load balance.

Hash function is the general method for partition. By doing this pairs which have the same key can be put into the same group. Here we need to define a new partition function because we have to make sure that the candidates in one reduce task are close to each other by the key. For example, if the number of

reduce tasks is two, then we need to divide the output of map phase into two parts. Assume the output is as follows (just consider the key):

5, 1, 9, 6, 2, 4,

then we can define a partition function like this:

```
if key >= 5
   put it into part1;
else
   put it into part2;
```

So the six keys are put into two groups, one includes 5, 9 and 6, and the other has 1, 2 and 4, which makes sure that the number in part2 is always smaller than part1. But how to decide the partition value, here is 5, is a challenge. We need first to assess the data and obtain an approximate value. The general method is to take a random sample from the big data and then find out the distribution. After partition, every reduce task have a temp file which consists of the parts from every map task. The next work is to combine them together and sort them. This is quite simple because any efficient sort algorithm can be used to do the work.

3.7 Getting the Result

Different applications may have different requirements for the final result. For example, we just want to find the nearest object from query point and the object should have all the required keywords. This is the simplest application. More general situation is to find more than one object that satisfies the requirements. In this situation, the cost of traditional spatial keyword query will increase by the size of results. Fortunately, we do not need additional cost to do this by MapReduce. We can just open the output file by reduce task and fetch the *top* k objects. Extremely, if all the objects including the keywords are required the traditional methods will be highly costly. But in MapReduce the output of each reduce task is the final answer.

4 Experiments

4.1 Datasets

There are two sources of data, i.e., synthetic data and real data.The synthetic data is randomly generated and contains about 50,000,000 objects. Each object consists of three parts, the ID, the spatial attributes and the keyword list. The ID is a positive integer; the spatial attributes are in two dimensional spaces which have the format of (x, y); the keyword list consists of one or more keywords. We first choose 100 keywords and then pick a random number of keywords from the set for each object. We use S-data to represent the synthetic data.

The real data is from the U.S. Board on geographic names (geonames.usgs.gov). We use R-data to represent this kind of data. Every object in R-data is a geographic location and it has a large number of words to describe some related

information. The real data needs to expand because its size is too small, only about 200MB. Here we transform the spatial attributes by adding a random number to each of the spatial attribute. Finally we obtain the same scale of data with S-data.

4.2 Query Design

First we define three kinds of queries. They are Single Spatial Keyword Query (SSKQ), k-Spatial Keyword Query (kSKQ), and Complete Spatial Keyword Query (CSKQ). They respectively return one record, k records and all the required records.

These queries are executed on a single computer and the distributed system which is implemented by the MapReduce. And the size of data varies from 100MB to 3.4GB.

4.3 Experiment Setup and Performance Metrics

We implemented all the algorithms with JDK1.6 and the configuration for each computer is an Intel(R) Core(TM)i3 CPU @2.6GHz with 2GB of RAM. The systems consists of 32 computers and its topological structure is a typical two-level network architecture which is described as Fig. 2.

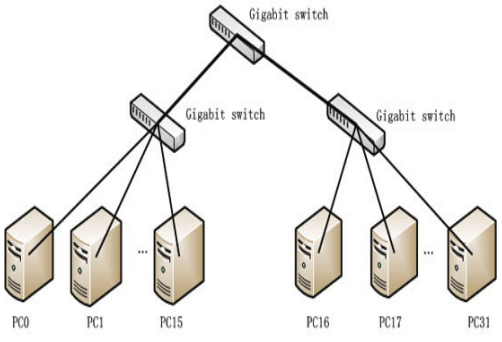


Fig. 2. The topological structure of MapReduce cluster System

The Hadoop[4] is used to build the distributed system. Hadoop consists of MapReduce and HDFS. When running the program we start half of the 32 machines. One of the 16 machines is used as master node which works as the namenode and jobtracker. And the other 15 machines are used to serve as datanodes and tasktrackers. Fig. 3 lists the 15 live datanodes.

When running the program we can set different number of map tasks and reduce tasks to test our solutions. And the running time is used to assess the efficiency of each query.

Live Datanodes : 15

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
hadoop-10	2	In Service	91.67	0	12.82	78.85	0		86.02	1
hadoop-11	2	In Service	91.67	0	13.63	78.04	0		85.13	0
hadoop-12	1	In Service	91.67	0	13.19	78.48	0		85.61	0
hadoop-13	0	In Service	91.67	0	13.4	78.26	0		85.38	0
hadoop-14	0	In Service	91.67	0	12.7	78.96	0		86.14	0
hadoop-15	0	In Service	91.67	0	14.47	77.19	0		84.21	0
hadoop-16	0	In Service	91.67	0	14.06	77.61	0		84.66	0
hadoop-2	1	In Service	91.67	0	13.77	77.9	0		84.98	0
hadoop-3	0	In Service	91.67	0	13.12	78.55	0		85.69	0
hadoop-4	0	In Service	91.67	0	14.2	77.47	0		84.51	0
hadoop-5	2	In Service	91.67	0	13.92	77.75	0		84.82	0
hadoop-6	2	In Service	91.67	0	13.34	78.33	0		85.45	0
hadoop-7	2	In Service	91.67	0	13.28	78.38	0		85.51	0
hadoop-8	2	In Service	91.67	0	13.95	77.71	0		84.78	1
hadoop-9	2	In Service	91.67	0	13.85	77.81	0		84.89	1

Hadoop, 2011.

Fig. 3. The live data nodes

4.4 Results and Analysis

Fig. 4 shows the running time with different k value on single computer on the S-data. As can be seen from the Fig. 4 the running time grows with the k value almost linearly. It means the running time will be bigger if we want to find more records. For example if we want to find 50000 records the time cost will be approximately 280 seconds. And we can also see from the Fig. 4 that the running time grows more rapidly if the data size is larger.

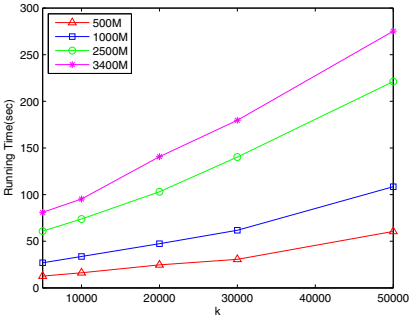
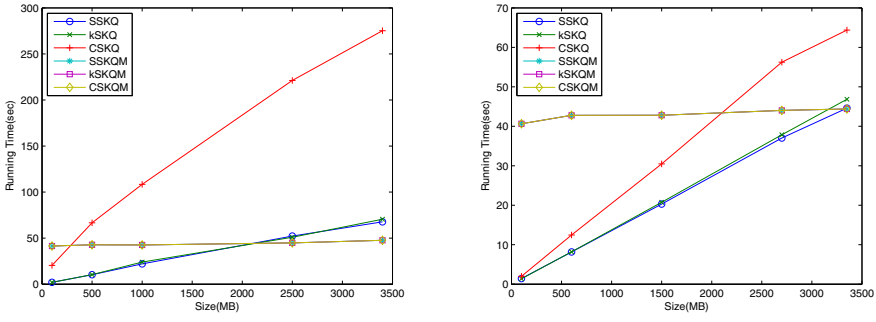


Fig. 4. Running time *vs.* k value

The following two figures show the results of the three queries on the two kinds of data. The k value is 10 for the kSKQ. As for the CSKQ we set k to 50000 which is smaller than the required value. We do this because the running time will be too big to show if we ask for all the required records.

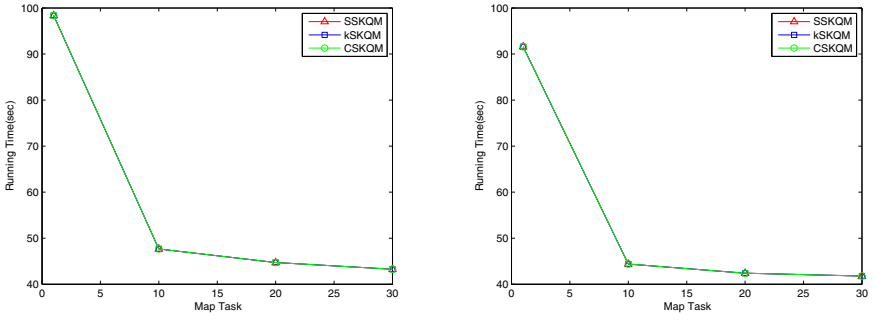
From Fig. 5(a) we can see that the running time for three queries on single compute increases with the data size linearly. The SSKQ and kSKQ almost have the same polyline because the difference between 1 and 10 is not outstanding. But the running time for CSKQ grows much more rapidly because it has quite large k value. The running time on MapReduce is almost the same when the data size changes. And the running time is much bigger than query on single

computer when the dataset is small. The reason is that MapReduce needs about 30 seconds to initialize. But after that the data is divided into many small blocks and these blocks are processed in parallel. So the running time is rather stable. In addition, the running time on MapReduce has nothing to with the k value. That is why the three polylines for SSKQM, kSKQM and CSKQM are overlaid with each other. This is a big advantage compared with the single computer query.



(a) S-Data (b) R-Data

Fig. 5. Single machine *vs.* MapReduce cluster system



(a) S-Data (b) R-Data

Fig. 6. Running time vs. the number of Map tasks

Fig. 5(b) shows the result on R-data. We can see that the running time is smaller than Fig. 5(a) totally. This is because a big part of the real data has been in order before processing. So a considerable part of sort cost is saved. The other features in this figure are the same with Fig. 5(a).

Fig. 6 describes the effect of number of various map task. We set four values of the number of map task, i.e., 1,10,20,30. We can see that the running time decreases with the increase of map task. Both datasets are about 3.4GB and

each block is 64MB. So we have about 50 blocks. More map tasks means more blocks can be processed at the same time, which reduces the running time. As shown in Fig. 5 and Fig. 6, if there is only one map task, the running time of distributed system is bigger than the single computer. Because in distributed system the data is not stored locally and map task needs to read remote data, which leads to high cost.

5 Conclusion and Future Work

In this paper, we use MapReduce to perform spatial keyword queries. Experimental results show that it is efficient to deal with spatial keyword queries over big, seldom used, type mixed spatial data with MapReduce. As for future work, how to employ index mechanism with MapReduce is a promising direction.

References

1. Ramaswamy, H., Bijit, H., Chen, L., et al.: Processing spatial-keyword (SK) queries in GIR systems. In: SSDBM (2007)
2. Dean, J., Ghemawat, S.: MapReduce Simplified Data Processing on Large Clusters. In: OSDI, pp. 137–150 (2004)
3. Ghemawat, S., Gobioff, H., Leung, S.-T., et al.: The Google File System. In: ACM/SOSP, pp. 29–43 (2003)
4. White, T.: Hadoop: The Definitive Guide. O'Reilly, CA (2009)
5. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: SIGMOD, pp. 71–79 (1995)
6. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. PVLDB 2(1), 337–348 (2009)
7. Guttman, A.: R-Trees-a dynamic index structure for spatial searching. In: SIGMOD, pp. 47–57 (1984)
8. Timos, K.S.: The R*-Tree: An Efficient and Robust Access Method for points and Rectangles. In: SIGMOD, pp. 322–331 (2000)
9. Roussopoulos, N., Leifker, D.: Direct spatial search on pictorial databases using packed R-trees. In: SIGMOD, pp. 17–31 (1985)
10. Zobel, J., Moffat, A., Ramamonhanarao, K.: Inverted files versus signature files for text indexing. ACM Transactions on Database System 23(4), 453–490 (1998)
11. Li, Z., Lee, K.C.K., Zheng, B., et al.: IR-tree-An efficient index for geographic document search. IEEE Transactions on Knowledge and Data Engineering 23(4), 585–599 (2011)
12. Felipe, I.D., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: ICDE, pp. 656–665 (2008)
13. Zhang, D., Chee, Y.M., Mondal, A., et al.: Keyword search in spatial databases: Towards searching by document. In: ICDE, pp. 688–699 (2009)
14. Zhou, Y., Xie, X., Wang, C., et al.: Hybrid index structures for location-based web search. In: CIKM, pp. 155–162 (2005)