INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATIONS AND ROBOTICS
ISSN 2320-7345

COOPERATIVE RANK BASED INDEX SPATIAL KEYWORD QUERIES

K. Ramesh[1], Dr.M.A. DoraiRangaswamy [2]

[1]*Asst. Prof, Department of MCA, Mailam Engineering College, Tindivanam,krishnarams29@gmail.com*
[2]*Sr. Professor, Department of CSE, AVIT, Chennai. IEEE Senior Member, drdorairs@yahoo.co.in*

Abstract

A mobile ad hoc network (MANET) is a collection of mobile, autonomous, wireless devices that form a communications network without the assistance of a fixed infrastructure. The ultimate goal of MANET network designers is to provide a self-protecting, "dynamic, self-forming, and self-healing network" for nodes on the move. [1] Each MANET node can serve as a router, and may move arbitrary and dynamically connected to form network depending on their positions and transmission range. The topology of the ad hoc network depends on the transmission power of the nodes and the location of the Mobile Nodes, which may change with time [2]. Given a spatial location and a set of keywords, a top spatial keyword query returns the best spatio-textual objects ranked according to their proximity to the query location and relevance to the query keywords. In this paper, we propose a cooperative rank based index (CRI) spatial keyword queries to improve the performance. Our rank maps each distinct term to a set of objects containing the term. The objects are stored differently according to the document frequency of the term and can be retrieved efficiently in decreasing order of keyword relevance and spatial proximity. Moreover, we present an algorithm that exploit process top rank spatial keyword queries efficiently.

Keywords*:* MANET, Cooperative, spatial

1. Introduction

In a location and a set of keywords, a rank based spatial keyword query returns a ranked set of best spatio-textual objects which includes both the spatial distance between the objects and the query location, and the relevance of the text describing the objects to the query keywords. There are several applications that can benefit from rank based spatial keyword queries such as finding the tweets sent from a given location (Twitter) or finding images near by a given location whose annotation is similar to the query keywords (Flickr). There are also other applications for GPS-enabled mobile phones that can benefit from such queries.

Top-k spatial keyword queries are intuitive and constitute a useful tool for many applications. However, processing top-k spatial keyword queries efficiently is complex and requires a hybrid index combining information retrieval and spatial indexes. The state-of-the-art approaches proposed by Cong et al. [4] and Li et al. [11] employ a hybrid index that augments the nodes of an R-tree with inverted indexes. The inverted index at each node refers to a pseudo-document that represents all the objects under the node. Therefore, in order to verify if a node is relevant for a set of query keywords, the current approaches access the inverted index at each node to evaluate the similarity between the query keywords and the pseudo-document associated with the node. This process incurs in non-negligible processing cost that result in long response time.

In this paper, we propose a novel method for processing rank based index spatial keyword queries more efficiently. Instead of employing a single R-tree embedded with inverted indexes, we propose a new cooperative rank based Index(CRI) that maps each keyword (term) to a distinct aggregated R-tree(aR-tree) [14] that stores the objects with the given term. In fact, we employ an aR-tree only when the number of objects exceeds a given threshold. As long as the threshold is not exceeded, the objects are stored in a file, one block per term. However, for ease of presentation, let us assume that we employ an aR-tree for each term. The aR-tree stores the latitude and longitude of the objects, and maintains an aggregated value that represents the maximum term impact (normalized weight) of the objects under the node. Consequently, it is possible to retrieve the best objects ranked in terms of both spatial relevance and keyword relevance efficiently and incrementally. For processing a ranked spatial keyword query with a single keyword, only few nodes of a single aR-tree are accessed. For queries with more than one keyword, we employ an efficient algorithm that aggregates the partial-scores on keyword relevance of the objects to obtain the best results efficiently. In summary, the main contributions of this paper are:

– We present CRI, an index that maps each term in the vocabulary into a distinct aR-tree or block that stores all objects with the given term.

– We propose efficient algorithms that exploit the CRI in order to process ranked spatial keyword queries efficiently.

2. Related Work

Initially, the research on spatial keyword queries focused on improving the performance of spatial queries in internet as search engines. Zhou et al. [17] did a relevant work combining inverted indexes [18] and R-trees, and propose three approaches: 1) indexing the data in both R-trees and inverted indexes, 2) creating an R-tree for each term, and 3) integrating keywords in the intermediary nodes of an R-tree. They found out that the second approach achieved better performance. However, they did not consider objects with a precise location (latitude and longitude), and did not provide support for top-k spatial keyword queries. Chen et al. [3] also had an information retrieval perspective on their work and did not provide support for exact query location of objects. In their approach, inverted indexes and R-trees are accessed separately in two different stages.

With the popularization of GPS-enabled devices, the research focused on searching for objects in a specific location. Hariharan et al.[7] proposed augmenting the nodes of an R-tree with keywords extracted from the objects in the sub-tree of the node. These keywords are then indexed in a structure similar to an inverted index for fast retrieval. Their approach supports conjunctive query in a given region of space. It is not clear, however, how their solution can be extended to support top-k spatial keyword queries. Later, Ian de Felipe et al. [5] proposed a data structure that integrates signature files and R-trees. The main idea was indexing the spatial objects in an R-tree employing a signature on the nodes to indicate the presence of a given keyword in the sub-tree of the node.

There are two previous approaches that support top-k spatial keyword queries. They were developed concurrently by Cong et al. [4] and Li et al. [11]. Both approaches augment the nodes of an R-tree with a document vector that represents all documents in the sub-tree of the node. For all terms present in the objects in the sub-tree of the node, the vector stores the maximum impact of the term (normalized weight). Consequently, the vector allows computing an upper bound for the textual score (textual relevance) that can be achieved visiting a given node. Hence, it is possible to rank the nodes according to textual relevance and spatial relevance, and decide which nodes should be accessed first to compute the top results.

The work of Cong et al. goes beyond the work of Li et al. by incorporating document similarity to build a more advanced R-tree namely DIR-tree. DIR-tree groups, in the same node, objects that are near each other in terms of spatial distance, and whose textual descriptions are also similar. Furthermore, instead of comparing vectors at query time, DIR-tree employs an inverted index associated with each node that permits to retrieve the children of the node that can contribute with a given query keyword efficiently. Only the posting lists associated with the query keywords are accessed. Cong et al. also propose clustering the nodes of DIR-tree (CDIR-tree) to further improve the query processing performance. The main idea is grouping related entries (objects, in case of leaf-nodes) and employing a pseudo-document to represent each group. Hence, more precise bounds can be estimated at query time, consequently, improving the query processing performance. However, it is not clear if the improvement achieved at query processing time compensates the additional cost required for clustering the nodes (pre-processing), and the extra storage space demanded by CDIR-tree. Moreover, keeping a CDIR-tree updated is more complex. For this reason, we decided to compare our approach against the DIR-tree proposed by Cong et al., and we consider this approach as the state-of-the-art.

3. Problem Statement

Let S be a dataset in which each object $p \in S$ is a pair (l, d) of a spatial location $p.l$ and a textual description $p.d$ (e.g., the facilities and menu of a car showroom). Similarly, a spatial keyword query $q = \_l, d\_$ has two components, where $q.l$ is a spatial location and $q.d$ is a set of keywords. The answer to query q is a list of k objects that are in ascending order of their distance to the query location $q.l$ and whose descriptions contain the set of query keywords $q.d$. Formally, let the function $dist(\cdot, \cdot)$ denote the Euclidean distance between its argument locations, and let $S(q.d) = \{p \in S \mid q.d \subseteq p.d\}$ be the objects in S that contain all the keywords in q. The result of the *top-k spatial keyword query* q, $q(S)$, is a subset of S $(q.d)$ containing k objects such that $\forall p \in q(S)$ $(\forall p\_ \in S(q.d) - q(S)$ $(dist(q.l, p.l) \leq dist(q.l, p.l)))$. The *joint top-k spatial keyword query* Q is a set $\{qi\}$ of such queries.

We introduce the following notion to capture useful information on a joint query Q: (i) $Q.l = \text{MBR}_{qi \in Q}\, qi.l$ is the minimum bounding rectangle (MBR) of the locations of the sub queries in Q, (ii) $Q.d = \cup_{qi \in Q}\, qi.d$ is the union of the keyword sets of the sub queries in Q, and (iii) $Q.m = \min_{qi \in Q} |qi.d|$ is the smallest keyword set size of a sub query in Q.

We later define a variable $qi.\tau$ that captures the upper bound k^{th} nearest neighbor distance of sub query qi. The value $Q.\tau = \max_{qi \in Q}\, qi.\tau$ then represents the maximum upper bound k^{th} nearest neighbor distance of all the sub queries in Q.

4. Coperative Rank based Index (CRI)

We present the existing IR-tree in Section 4.1 and then proceed to develop an algorithm for processing joint top spatial keyword queries. The algorithms are generic and are not tied to a particular index.

4.1 Preliminaries: the IR-Tree

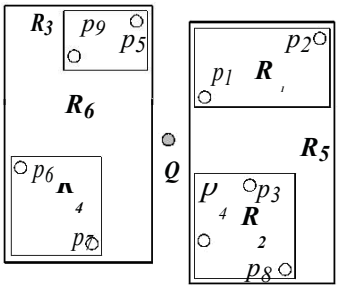
The IR-tree which we use as a baseline, is essentially an R-tree [5] extended with inverted files . The IR-tree's leaf nodes contain entries of the form $(p, p.l, p.di)$, where p refers to an object in dataset D, $p.l$ is the bounding rectangle of p, and $p.di$ is the identifier of the description of p. Each leaf node also contains a pointer to an inverted file with the text descriptions of the objects stored in the node. An inverted file index has two main components along with a tree.

- Vocabulary - The vocabulary stores, for each distinct term, the number of objects in which the term appears (df_t), a flag indicating the type of storage used by the term (block or tree), and a pointer to a block or aR-tree that stores the objects containing the given term.

- Blocks - Each block stores a set of objects; the size of a block is an application parameter. For each object, we store the object identification p.id, the object location p.l, and the impact of term t in p.d ($l_{t;p;d}$). The objects stored in a block are not ordered.

Each non-leaf node R in the IR-tree contains a number of entries of the form $(cp, rect, cp.di)$ where cp is the address of a child node of R, $rect$ is the MBR of all rectangles in entries of the child node, and $cp.di$ is the identifier of a pseudo text description that is the union of all text descriptions in the entries of the child node.

As an example, Figure 1a contains 8 spatial objects namely p1, p2,…., p8 and Figure 1b shows the words appearing in the description about each object.



(a) object locations

object	words
p_1	a, b
p_2	a, c
p_3	a, d
p_4	e, f
p_5	a, b
p_6	d, e
p_7	e, f
p_8	d, f
p_9	a, d

(b) object descriptions

Figure 1: A Dataset of Spatial Keyword Objects

The CRI has several advantages First, terms with different document frequency are stored differently. Second, CRI has good support for distribution. CRI employs a different data structure for each term and can benefit from techniques used by traditional distributed search engines such as term partitioning [18]. Third, CRI provides good support for updates. Although one tree or block has to be accessed for each term in an object, the operations executed at each tree or block can be performed efficiently. Furthermore, the average number of distinct terms per document is small in most of the applications that are target of this query. Finally, CRI allows efficient query execution. For queries with a single keyword, only one small tree (in general) or a block needs to be accessed. For queries with several keywords only few nodes of a set of small trees or blocks are accessed.

4.1 Query Generation using Single Keyword

Top spatial keyword queries with a single keyword t can be efficiently processed using the CRI, since only one single block or tree containing the objects with the term t is accessed. If the objects are stored in a block, the query processing steps are: 1) retrieve all objects p in the block, 2) insert the objects in a heap in decreasing order of τ (p, q), and 3) report the top-k best objects. If the objects are stored in an aR-tree, we employ an incremental algorithm (Algorithm 1) to retrieve the objects in decreasing order of τ(p, q).

Algorithm 1 QGSK (MaxHeap H^t, Query q)
1: INPUT: MaxHeap H^t with entries e in decreasing order of score (e; q) and the query q.
2: OUTPUT: The next object p with highest score (p; q).
3: Entry e H^t:pop()
4: With e is not an object do
5: if e is an intermediary-node then
6: with each node n 2 e do
7: H^t: insert (n; (n; q))
8: end
9: else
10: with each spatio-textual object p 2 e do
11: H^t: insert (p; (p; q))
12: end
13: end if
14: e H^t: pop ()
15: end
16: return e

The QGSK (Query Generation Single Keyword, Algorithm 1) visits entries e (nodes or objects) in an aR-tree in decreasing order of τ (e, q). The score of a node n, τ(n, q), and the score of an object p, τ (p, q), are computed in a similar way. The spatial proximity δ(n, q) is obtained computing the minimum Euclidean distance between q and any border of the MBR of n. On the other hand, the textual relevance θ(n, q) is obtained multiplying the impact of t in the query $l_{t;q}$by the impact of t in the node $l_{t;n}$that is the maximum impact among any entry e in the sub-tree of n. Consequently, the score τ(n, q) is an upper bound score for any entry e in the sub-tree of n, since any entry e in the sub-tree of n has a distance to q longer or equal d(q, n), d(q, n) d(q, e); and the term impact of n is higher or equal the term impact of any entry e, $l_{t;nlt;e}$.

QGSK receives as parameter a priority queue (MaxHeap) H^t and a query q. The heap stores the entries e of R^t in decreasing order of score τ(e, q). Initially, the heap has the root of R^t. The entry e (line 3) is the entry with highest score in H^t. If e is an intermediary-node (line 5), the algorithm computes the score of all nodes n children of e and insert n in H^t in decreasing order of τ(n, q) (lines 6-8). If e is a leaf-node (line 9), the algorithm computes the score of all objects pchildren of e and insert p in H^tin decreasing order of score (lines 10-12). This process repeats until an object p achieves the top of the heap (line 4). This means that there is no other object in H^t with higher score than p. Therefore, p can be reported incrementally (line 16). The algorithm keeps the state of the variables for future access.

4.1 Query Generation using Multiple Keywords

Processing multiple-keyword queries in the CRI requires aggregating objects from different sources S_i (aR-trees or blocks), where i refers to a distinct term $t_i \in$ q.d. The naive way is to retrieve objects p from each source S_i in decreasing order of score τ (p, q) replacing q.d by a query term t $\in$ q.d, τ (p, {t}). The final score is

obtained adding the scores retrieved for each term. However, $\tau(p, q) \neq \sum_{t \in q:d} \tau(p, \{t\})$, because the spatial proximity is replicated in the scores computed for each term $\tau(p, \{t\})$. Hence, we propose to derive a partial score on keyword $\tau(p, q)$ such that the score $\tau(p, q)$ can be computed in terms of the sum of partial-scores obtained from each source, $\tau(p, q) = \sum_{t \in q:d} T^t(p, q)$.

The partial-score reduces the weight of the spatial proximity $\delta(p.l, q.l)$ according to the number of distinct terms in the query q.d. Furthermore, once we have obtained an object p from a source Si, we can also derive a lower bound partial-score $T^t(p, q)$ for p on the other sources in which p has not been seen yet. This is the lowest possible partial-score that p may have in a source where it has not been seen yet, since the proximity between p and q will not change. With the partial-scores $T^t(p, q)$ and T^t - (p, q), we can compute the lower bound and upper bound scores based on partial information about the object.

The Query Generation using Multiple Keyword (QGMK), Algorithm -2 computes a top spatial keyword query progressively by aggregating the partial-scores of the objects retrieved for a given keyword. The objects containing a given term are retrieved in decreasing order of partial-scores from the source (aR-tree or block). In order to retrieve the objects in decreasing order of partial-score, we employ the QGSK algorithm (Algorithm 1) replacing the score $\tau(p, q)$ by partial-score $\tau^t(p, q)$. Each time an object p is retrieved from a source $S_i(S_i.next())$, we compute the lower bound score of p, update the upper bound score for any unseen object, and check if there is an object whose the lower bound score is higher or equal the upper bound of any other object. Those objects are reported progressively. We repeat this process until k objects have been found.

Algorithm 2 presents the QGMK algorithm. QGMK receives as parameter a top spatial keyword query q and reports the top results incrementally. QGMK employ one source S_i for each distinct term $t_i \in q.d$(line 3). Next, for each source S_i, the algorithm sets an upper bound that maintains the highest partial-score among the objects unseen from S_i(line 5). The upper bound is updated every time that an object p is retrieved from S_i(line 10). During each iteration (lines 6-22), QGMK selects a source I (line 7), where the procedure selectSource(q.d) defines the strategy to select the source. In this paper, we employ a round-robin strategy that selects a source S_i that is not empty.

Algorithm 2 QGMK (Query q)
1: INPUT: The top-k spatial keyword query q.
2: OUTPUT: Progressively reports the top-k objects found.
3: Si source of the term ti $\in$ q:d
4: C = Li = 0
5: $u_i = \infty$
6: while $\exists S_i$ such that $S_i \neq 0$ do
7: i = selectSource(q:d)
8: p = S_i.next ()
9: $u_i = T^{ti}(p,q)$ // update upper bound
10: $L_i = L_i \cup p$
11: p- = $\sum_{\forall j:p \in Lj} T^{tj}(p, q) + \sum_{\forall j:p \neq Lj} T^{tj}(p, q)$
12: if p $\notin$ C then
13: C = C $\cup$ P
14: end if
15: for each p $\in$ C do
16: P- = $\sum_{\forall j:p \in Lj} T^{tj}(p, q) + \sum_{\forall j:p \neq Lj} \max(u_j, T^{tj}(p, q))$
17: end for
18: while $\exists p \in C : p - \geq \max_{\forall} \in$ c do
19: C = C – p
20: reports p as next top-k, halt if q:k objects have been reported
21: end while
22: end while
23: if less than q:k objects have been reported then
24: return the objects in C with highest lower bound score p
25: end if

Other more sophisticated strategies such as keeping an indicator that express the effectiveness of selecting a source can also be employed. MKA continues retrieving from S_i the next object p with highest partial-score $T^{ti}(p, q)$ (line 8), and updating the upper bound score u_{-i}(line 10) with the new partial-score retrieved from S_i. At this point, any unseen object in S_i has a lower or equal partial-score than u_{-i}. Next, QGMK marks that p has been seen in S_i adding p into L_i(line 11), and updating the lower bound score for p (line 12). The lower bound score is computed by summing the partial-scores known for p with the worst possible partial-score that p may have based on the sources in which p has not been seen yet. Then, QGMK

checks if p is in the candidate set C, inserting p otherwise (lines 13-15). Next, QGMK updates the upper bound score for any object in the candidate set C (lines 15-17). The upper bound score for a given object in a source that it has not been yet, does not decrease bellow its lower bound score on that source (line 17). The objects p ∈ C whose lower bound p− is higher or equal the upper bound o− of any other object in C (lines 18-21) is reported incrementally. QGMK repeats this process until k objects has been reported, or the sources are empty. If the sources are empty and less than k objects have been returned, QGMK reports the objects in C with highest lower bound score (lines 23-25) as the remaining top queries.

5. Experimental Study

In this section, we compare our approach the CRI against the DIR-tree proposed by Cong et al. [4]. All algorithms were implemented in Java using the XXL library. The nodes of the aR-trees, employed in CRI, have a block size of 4KB that is able to store between 42 and 85 entries. The block in the file used to store the non-frequent items has a maximum size of 4KB that permits to store a maximum of 146 entries. The intuition behind this choice is that we store objects in an aR-tree only when there are enough objects to fill more than one node of an aR-tree. Each node of a DIR-tree also has a block size of 4KB and is able to store between 46 and 92 entries. The parameter β used to balance textual similarity and spatial location during the construction of the DIR-tree was set to 0.1 as suggested by Cong et al. [4].

Parameter	Values
Number of results (k)	10, 20, 30, 40, 50
Number of keywords	1, 2, 3, 4, 5
Query preference rate	0.1, 0.3, 0.5, 0.7, 0.9
Twitter dataset	1M, 2M, 3M, 4M

Table 1: Settings used in the experiments.

Datasets	Total No. of Objects	Avg. No. of unique words per object	Total No. of unique words	Total No. of words
Twitter1	1,000,000	11.94	553,515	12,542,414
Twitter2	2,000,000	12.00	1,009,711	25,203,367
Twitter3	3,000,000	12.26	1,391,171	38,655,751
Twitter4	4,000,000	12.27	1,678,451	51,661,462
Data1	131,461	131.70	101,650	32,622,168

Table 2: Characteristics of the datasets.

Experiments were executed on a PC with a 3GHz Dual Core Intel processor and 2GB RAM. In each experiment, we execute 100 queries to warm-up the buffers, and collect the average results of the next 800 queries. The queries are randomly generated using the same vocabulary and the same spatial area of the datasets as used by Cong et al. [4]. We employed a buffer whose size was fixed in 4MB for both approaches. In the experiments, we measured the total execution time (referred as response time) and the number of I/Os (page faults). All charts are plotted using a logarithmic scale on the y-axis. The main parameters and values used through the experiments are presented in Table 1. Datasets. Table 2 shows the characteristics of the datasets used in the experiments. We employed four Twitter datasets of 1M, 2M, 3M, and 4M objects each, where each object is composed by a Twitter message (tweet) and a random location where latitude and longitude are within the range [0,100]. In order to create these datasets, we used the first 10 million non-empty tweets from the Twitter dataset[2]. The Data1 dataset was created combining texts from 20 Newsgroups dataset[3]and locations from LA streets[4]. This dataset is similar to the Data1 dataset used by Cong et al. [4]. However, instead of selecting only 5 groups, we employed all documents in the 20 Newsgroups dataset.

5.1 Performance of Query Processing

we evaluate the query processing performance of the CRI and DIR-tree for different setups. In all experiments, we employ the default settings (Table 1) to study the impact on I/O and response time, while varying a single parameter.

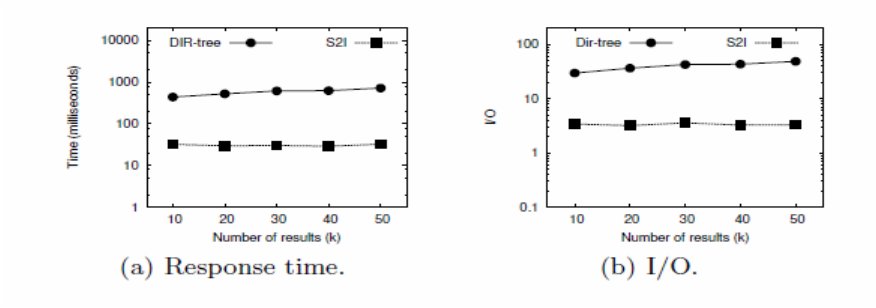


Figure 2: Response time and I/O varying the number of results (k).

Varying the number of results (k) - Fig. 2 plots the response time and I/O of the CRI and DIR-tree, while varying the number of results k. The response time achieved using CRI is one order of magnitude better than using DIR-tree (Fig. 2(a)). Furthermore, the advantage of CRI increases when the number of results increases. The main reason for this is that CRI accesses less disk pages to process a query (Fig. 2(b). In order to process a query employing DIR-tree, the inverted files at each node are accessed to obtain the posting lists of each distinct keyword in the query. For example, a query with 3 distinct keywords is performed in two steps: first, the postings lists of each keyword is retrieved in order to identify the entries of the node that can contribute to the query results, then the relevant entries are visited in decreasing order of score. Although the size of the posting lists are small, since they are bounded by the maximum capacity of a node, the process to perform such queries on inverted indexes incurs in non-negligible cost.

Varying the number of keywords - Fig. 3 depicts the response time and I/O, while varying the number of query keywords. Again, the response time (Fig. 3(a)) and I/O (Fig. 3(b)) achieved by using the CRI are one order of magnitude better. Single-keyword queries are processed efficiently employing the QGSK algorithm. Hence, few pages are accessed during the query processing.

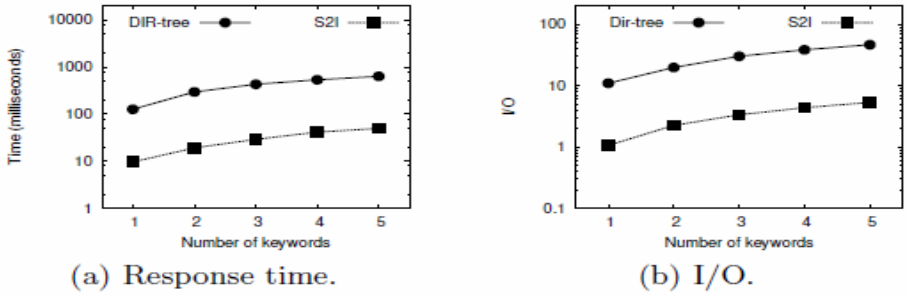


Figure 3: Response time and I/O varying the number of keywords.

As expected, the larger the number of keywords, the higher the I/O required processing the query. However, the advantage of CRI over DIR-tree remains constant, which demonstrates the efficiency of the QGSK algorithm.

6. Conclusion

This paper introduces the joint top spatial keyword query and presents efficient means of computing the query. Our solution consists of: (i) the QGSK that exploits keyword partitioning and inverted bitmaps for indexing spatial keyword data, and (ii) the QGMK algorithm that processes multiple queries jointly. In addition, we describe how to adapt the solution to existing index structures for spatial keyword data. Empirical studies with combinations of two algorithms and a range of indexes demonstrate that the QGMK algorithm on the W-IBR-tree is the most efficient combination for processing joint top spatial keyword queries.

It is straightforward to extend our solution to process top-k spatial keyword queries in spatial networks. We take advantage of Euclidean distance being a lower bound on network distance. While reporting the top objects incrementally, if the current object is farther away from the query in terms of Euclidean distance than is the k^{th} candidate object in terms of network distance, the algorithm stops and the top result objects in the spatial network are found. The network distance from each object to a query can be easily computed using an existing, efficient approach

REFERENCES

1. Mark E. Orwat, Timothy E. Levin, and Cynthia E. Irvine, "An Ontological Approach to Secure MANET Management", In Proceedings of the 2008 Third International Conference on Availability, Reliability and Security, pp.787-794, 2008.
2. Mohd Izuan Mohd Saad and Zuriati Ahmad Zukarnain, "Performance Analysis of Random-Based Mobility Models in MANET Routing Protocol", European Journal of Scientific Research, Vol.32 No.4, pp.444-454, 2009.
3. Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In Proceedings of the ACM Int. Conf. on Management of Data (SIGMOD), pages 277–288, 2006.
4. G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. In Int. Conf. on Very Large Data Bases (VLDB), pages 337–348, 2009.
5. I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In Proceedings of Int. Conf. on Data Engineering (ICDE), pages 656–665, 2008.
6. Le Gruenwald and Shankar M. Banik, "A Power-Aware Technique to Manage Real-Time Database Transactions in Mobile Ad-Hoc Networks", In Proceedings of the 12th International Workshop on Database and Expert Systems Applications, pp .570 - 574 , 2001.
7. R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In Proceedings of the Int. Conf. on Scientific and Statistical Database Management (SSDBM), pages 1–10, 2007.
8. Jinbao li, yingshu li, my t. Thai and Jianzhong Li, "Data Caching and Query Processing in Manets", Journal Of Pervasive Computing And Communications, Vol.1, No.3, pp-169-178, Sept 2005.
9. Salem Hadim, Jameela Al-Jaroodi and Nader Mohamed, "Middleware Issues and Approaches for Mobile Ad hoc Networks", In proceeding of IEEE Consumer Communications and Networking Conference (CCNC 2006), Las Vegas, Nevada, January 2006.
10. Hassan Artail, Haidar Safa, and Samuel Pierre, "Database Caching in MANETs Based on Separation of Queries and Responses", In Proceeding of WiMob 05, Montreal, Canada, Aug. 2005.
11. Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang. IR-tree: An efficient index for geographic document search Proceedings of the IEEE Transactions on Knowledge and Data Engineering (TKDE), 99(4):585–599, 2010.
12. P T Tharani and G Vejey Subash, "A Scalable Quorum Based Approach for Improving Global Consistency in MANET", Int. J. on Recent Trends in Engineering & Technology, Vol. 05, No. 01,pp.82-85, Mar 2011.
13. Mohamed Ahmed Elfaki, Hamidah Ibrahim, Ali Mamat and Mohamed Othman, "Collaborative Caching Architecture for Continuous Query in Mobile Database", American Journal of Economics and Business Administration, pp.33-39, 2011.
14. D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP operations in spatial data warehouses. In Proceedings of the Int. Symposium on Advances in Spatial and Temporal Databases (SSTD), pages 443–459, 2001.
15. J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørv˚ag. Efficient processing of top-k spatial preference queries. Proceedings of the VLDB Endowment (PVLDB), 4(2):93–104, 2010.
16. G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. Information Processing and Management, 24(5):513–523, 1988.
17. Y. Zhou, X. Xie, C. Wang, Y. Gong, and W. Ma. Hybrid index structures for location-based web search. In Proceedings of Int. Conf. on Information and Knowledge Management (CIKM), pages 155–162, 2005.
18. J. Zobel and A. Moffat. Inverted files for text search engines. ACM Comp. Surveys, 38(2):1–56, 2006.