Aggregate Keyword Routing in Spatial Database

Kunjie Chen
Fudan University
Shanghai, China
chenkunjie@fudan.edu.cn

Weiwei Sun
Fudan University
Shanghai, China
wwsun@fudan.edu.cn

Chuanchuan Tu
Fudan University
Shanghai, China
tucc@fudan.edu.cn

Chunan Chen
Fudan University
Shanghai, China
chenchunan@fudan.edu.cn

Yan Huang
University of North Texas
huangyan@unt.edu

ABSTRACT

Due to the proliferation of Location-Based Service and popularity of online geo-tagged web pages, spatial keyword search has attracted significant attention from both academic and industrial communities. In this paper, we study the problem of finding the nearest aggregate point from multiple query points travelling through a set objects described by a given set of keywords, as well as the optimal routes from the query points to the aggregate point. This problem is defined as the *Aggregate Keyword Routing* (AKR) Query. We devise an exact algorithm to AKR query based on ellipse pruning. Next we propose an efficient approximate algorithm for AKR: *Center Based Assignment* (CBA). The performance of the proposed algorithms are evaluated with real data, the results demonstrate the efficiency and the effectiveness.

Categories and Subject Descriptors

H.2.4 [**Information Systems**]: Database Management Systems. Subject: Query processing

General Terms

Algorithm, Performance

Keywords

Query Processing, Spatial Keyword Query

1. INTRODUCTION

The wide spread of online geo-tagged web pages has led to an increasing interests in spatial keyword queries [3]. Given a set of objects P where each object in P has a location and is associated with a textual description, a spatial keyword query finds the object in P which is the closest to the query location and is relevant to the requested keywords. For example, finding the nearest restaurant that with "Chinese food" in its textual description.

ACM SIGSPATIAL GIS '12, November 6-9, 2012. Redondo Beach, CA, USA

In this paper, we study an interesting generation of spatial keyword query, which aims at finding the spatially closest object and the optimized routes that go through a set of objects described by a given set of keywords, from multiple query points. There are many applications for this type of query. For example, a group of three members, denoted as *query points* $Q=\{q_1, q_2, q_3\}$, decide to get together and have a barbecue at a park. There are three parks available with barbecue service, denoted as *aggregate points* $M = \{m_1, m_2, m_3\}$. Assume that they somehow select m_2 as the gathering point (as will be discussed in detail later). Before they reach m_2, they need to buy some beers, meat, fruits, cards, chess, magazines, etc. These preparing activities are named as *tasks* and are described with a set of keywords. In this example, the task description $\kappa=\{beers, meat, fruits, card, chess, magazine\}$. Given the meeting point m_2 and the task description κ, an *Aggregate Keyword Routing* (AKR) is to find a set of objects P' whose textual description covering κ, and the *optimal routes* from the Q to m_2 covering P'. Here, by *optimal*, we refer to minimizing the maximum distance traveled by any query points.

Aggregate keyword routing is a challenging problem. Firstly, given the gathering point and task descriptions, it is difficult to decide the objects to be passed through by each query point. Secondly, finding the optimal route which passes through a set of objects for a query point is still challenging. Thirdly, the AKR problem described above assumes that the gathering point is given. This problem will be more challenging if there are more than one aggregate point to be considered. Our main contribution can be summarize as follows:

- We formulate a new type of useful spatial keyword query that considers multiple query points.
- We devise an exact algorithm to the AKR query based on ellipse pruning, and an approximate algorithm *Center Based Assignment* (CBA).
- The performance of the proposed algorithms are evaluated with real data with different scale. The results demonstrate the efficiency and the effectiveness.

2. PROBLEM FORMULATION

In spatial databases, an object(point of interest) p is associated with its location and a set of keywords $\theta(p)$. These keywords may be extracted from the objects' textual description or added by the users as geotags. An Aggregate Keyword Routing Query consists of three parameters: a

query point set Q, a keyword set κ which we also call tasks, and an aggregate keyword set α that the aggregate point should contain. And we will call an object p a task point if $\theta(p)\bigcap\kappa\neq\emptyset$ and an object m candidate aggregate point if $\alpha\subseteq\theta(m)$. Before formally formulating the AKR query, we first give some definitions.

Definition *1.* A route $r_{s,m}=<s,p_1,...,p_x,m>$ is a point sequence that starts from point s and goes sequentially through p_1 to p_x and ends at point m. We denote length of the route as $L(r_{s,m})$ and the keyword set covered by the route as $\kappa(r_{s,m})$.

Definition *2.* Given a query point set $Q=\{q_1,q_2,...,q_n\}$, we use the notion $R_{Q,m}$ to a represent route set $\{r_{q_1,m},r_{q_2,m},...,r_{q_n,m}\}$. The length of $R_{Q,m}$ is specified by the maximum length of the route in the set, that is $L(R_{Q,m})=\underset{r_{q,m}\in R_{Q,m}}{Max} L(r_{q,m})$. The union set of keywords covered by all the routes in $R_{Q,m}$ is denoted as $\kappa(R_{Q,m})$.

Definition *3.* Given an Aggregate Keyword Routing query $<Q,\kappa,\alpha>$, the route set $R_{Q,m}$ is a candidate AKR answer if and only if $\alpha\subseteq\theta(m)$ and $\kappa\subseteq\kappa(R_{Q,m})$. We denote the set of all candidate AKR answers as A_R.

Now we can formally define the AKR query as follow.

Definition *4.* An AKR query $<Q,\kappa,\alpha>$ is to find a candidate AKR answer route set $R_{Q,m}$ such that $L(R_{Q,m})$ is minimum, i.e., $\underset{R_{Q,m}\in A_R}{Min} L(R_{Q,m})$.

Theorem 1. *The Aggregate Keyword Routing problem is NP-hard.*

PROOF. The proof is omitted for space limitation. □

3. EXACT ALGORITHM

In this section, we propose an efficient algorithm to solve the AKR problem. Our approach has two phases:

1. **Search and Assignment Phase**. In this phase, we use a traditional approach in [9] to incrementally get the ANN(using the Max function) of the query point set which covers the aggregate keyword set. For each aggregate point, the algorithm uses a strategy to get the candidate task points for each query point. Then, it assigns tasks to the query points.
2. **Task Finishing Phase**. After getting the candidate task point set and tasks for each query point, this phase deals with how to find an optimal route from the query point to the aggregate point when passing several task points.

The algorithm repeats these two phases until it finally gets an exact result.

3.1 Search and Assignment Phase

This phase aims at finding the candidate aggregate points and task points for each query point after getting an aggregate point, and enumerating possible task combinations to the query point set.

3.1.1 Aggregate Point Access Order

For an AKR query, not every point that covers the aggregate keywords can be the answer.

Lemma 1. *Suppose we have a candidate AKR answer $R_{Q,m}$ of the query, an object m' cannot be the final answer's aggregate point if $\underset{q\in Q}{Max} D(q,m') > L(R_{Q,m})$.*

PROOF. The proof is omitted for space limitation. □

That is, the candidate AKR answer's length defines the upper bound of the aggregate distance. So we use an incremental method to get those aggregate points and dynamically update the upper bound when the candidate AKR answer is changed. The algorithm can be terminated when it encounters an aggregate point m whose maximum distance to the query point exceeds the upper bound.

3.1.2 Task Points Pruning

After getting a candidate aggregate point m, the algorithm immediately examines whether there is a possible route set $R_{Q,m}$ whose length is smaller than the current's. So we first try to find out all candidate task points and then assign tasks to those query points. Consider that the algorithm needs to find the optimal routes in the next step, a smaller candidate task point set will lead to a better performance because it reduces the search space.

Lemma 2. *Let $R_{Q,m}$ to be the current candidate AKR answer. Given a query point $q\in Q$ and an aggregate point m', an object p is a candidate task point in route $r_{q,m'}$, which is in a better route set $R_{Q,m'}$ than $R_{Q,m}$, only if $D(q,p)+D(p,m')\leq L(R_{Q,m})$.*

PROOF. The proof is omitted for space limitation. □

The Lemma 2 requires to find out those objects with sum of distances to the aggregate point and the query point less than the candidate AKR answer's length. It encourages us to perform an ellipse range query defined by the query point, the aggregate point, and the length of the existing result. So we use the ellipse pruning algorithm introduced in [6].

3.1.3 Task Assignment

After getting candidate the task point set, the next step is to enumerate all possible task assignment combinations and then find the optimal routes. This iteration is costly and its time complexity is $O(|Q|^{|\kappa|})$ where $|Q|$ is the number of query points and $|\kappa|$ is the number of tasks. However, we can terminate the progress early using the following lemma.

Lemma 3. *Suppose the current aggregate point is m, if the algorithm finds a result $R_{Q,m}$ such that $L(R_{Q,m})=\underset{q\in Q}{Max} D(q,m)$, then $R_{Q,m}$ is the final answer of the AKR query.*

PROOF. The proof is omitted for space limitation. □

Now we describe how the exact algorithm works. It first uses an approximate algorithm to get an answer and use this answer as the upper bound of the results. Then the algorithm iterates until it gets the answer. For every iteration, it first gets the next ANN of the query set Q and updates the lower bound to be the maximum distance of the ANN to points in Q. The algorithm returns the result if the

lower bound is not smaller than the upper bound. The algorithm performs ellipse range queries to get the candidate task points and check whether these points' keyword sets intersect with κ. At last, it starts to assign every possible task combination to the query points and refresh the current result and upper bound. When finding a result whose length equals to the ANN distance, the algorithm returns the current result as the final answer.

3.2 Task Finishing Phase

The goal of this phase is to find a route set with minimum length for the given task assignment. The basic idea is to find every optimal route for the given query points and task assignment. However, the AKR query only asks for a route set so that maximum length of the route in the set is minimum. For other route whose length is not the maximum, a worse one whose length is smaller than the maximum can be an alternative answer. Thus, when trying to find the optimal route, we could maintain an upper bound which allows to return an approximate answer. The bound can be initialized as $\underset{q \in Q}{Max}\ d(q,m)$ because this is minimum length of any route set $R_{Q,m}$. Then we update the bound when finding an exact optimal route with length larger than the current bound.

How to find the optimal route is a challenging problem. We use the basic ideas of the exact approach in [13]. First we define an operation. Suppose r=<p_1,...,p_{x-1},p_x>, then $r \oplus p$=<p_1,...,p_{x-1},p,p_x>. The idea to find an optimal route $r_{q,m}$ covering κ is: maintain a min heap that stores the routes in ascending order according to the length of the route. The heap is initialized with route $r_{q,m}$=<q,m>. For each round, pop the route r on the top. If r covers κ, return r as the result. Otherwise, for every task point p which has the keywords not covered by r, push $r \oplus p$ to the heap.

Note that the min heap may become very large when the task point set and/or κ is large. To reduce the heap's size and accelerate query processing, we need to abandon those routes impossible expanded to be a result. We start by introducing an important lemma.

Lemma 4. *Given routes r_1, r_2, $\kappa(r_1)$=$\kappa(r_2)$ and an object p. Suppose route $r_2 \oplus p$ pops from heap after $r_1 \oplus p$. Then, $r_2 \oplus p$ cannot be expanded to be a result, i.e. $r_2 \oplus p$ can be filtered out.*

PROOF. The proof is omitted for space limitation. □

We define an operation called *isExpanded*(r) to decide whether to expand a route r=<q,p_{x1},...,p_{xk},p,m>. Suppose a route <q,p_{y1},...,p_{yl},p,m> which also covers $\kappa(r)$ has been popped from the heap before, then *isExpanded*(r) returns TRUE so that the algorithm knows that r does not need to be expanded according to Lemma 4. Otherwise, if *isExpanded*(r) returns FALSE, route r can be expanded and we will record the pair <p,$\kappa(r)$> so that next time the algorithm can use this record to prune some unnecessary routes.

The algorithm works as follows. To reduce the repeated computation of finding the same optimal route, the algorithm stores the route in a map using the query point, the aggregate point and the task description as the key. So it first finds out those result of queries which have been processed before. Then it starts to find the optimal route of every query point as we described before. The algorithm will terminate if it finds out a route whose length is larger than the current result's.

4. APPROXIMATION ALGORITHM

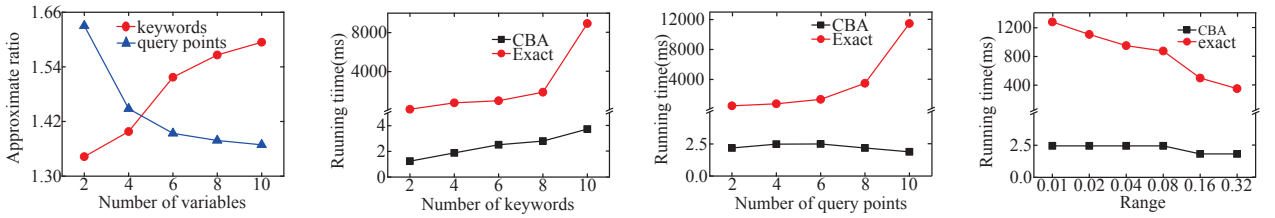
It is computational expensive to get an exact result when the query point set and the task keyword set is large. Here, we design an efficient approximate approach with good approximation rate to the exact result. This algorithm works similarly as the exact algorithm and also has two phases: (1)search for the aggregate point and the task points, (2)assign the task points to the routes. Different from the exact algorithm, this algorithm goes through these two phases only once. In the first phase, it directly uses an approximation approach to get the aggregate point as the final result point. Then, it uses the nearest neighbor query to get the task points which have keywords in κ. In the second phase, it uses a greed strategy to assign these tasks to query points. Before introduce the algorithm in detail, we first present an operation denoted as *MinLastIncr*. *MinLastIncr* has a route r_t=<p_1,...,p_k,m> and an object set P as the input. It returns an object p subjected to $\underset{p \in P}{argmin}\{D(p_k,p)+D(p,m)\}$. The implementation of the operation is straightforward. We just need to iterate every point in the object set P and return the point with the minimum distance increment.

The algorithm works as follows. It tries to get the aggregate point which covers the aggregate keyword set at first. Instead of performing a traditional ANN query, this algorithm gets the nearest neighbor m of the query set's centre point p_c which covers the aggregate keyword set. It saves lots of time when the query set is large or split up into several widely scattered locations. The centre point is the geometric centroid of query points and its detail computation can be found in [9]. Then, it starts to find out every nearest neighbor(NN) of the centre point p_c that covers the keyword in κ. The intuition behind this is that the task points near the centre point tend to have higher probability to be in the final result. In the task assignment phase, it selects the route r with the minimum length and uses operation *MinLastIncr* to get an task point p. Then, the route $r \oplus p$ is inserted back to the heap. When the task point set is empty, the algorithm returns the result. Because the algorithm gets the aggregate point and task points based on the centre point, we call it *Centre Base Assignment* (CBA) Algorithm.

The CBA algorithm returns an approximate answer very fast. Hence, we use it to get an approximate answer in the exact algorithm.

5. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of the exact algorithm and the CBA algorithm by experiments. We implement both algorithms in Visual C++ on an Intel Pentium Dual E2200 CPU and 2G memory computer. We use the map of London as our dataset. In the map, there are 34,162 objects and 121,049 keywords. The number of distinct keywords is 12,551. In the experiments, keywords in a query are randomly generated. When varying the number of query points from 2 to 10, we fixed the number of keywords to 6. When varying the number of keywords, we use 4 as the default number of query points. In the first two experiments, the query points cover 8% space of the whole map. To simplify the model of query and make it close to reality, every query has only one keyword for aggregate point.



(a) Approximate ratio (b) Number of keywords (c) Number of query points (d) Vary query points' range

Figure 1: Result on London

We show the relationship from approximate ratio to number of keywords and query points in figure 1(a) to save space. The line with triangle mark stands for the result when varying the number of query points, and the other one is varying number of keywords. From the figure we can see CBA always has an approximate ratio from 1.3 to 1.7, increasing with the growth of keywords' number and decreasing with the growth of query points' number. When the range of the query points changes, no obvious trend can be found, or in other words, CBA has a smooth trend when the range of query points changes. Because of the limitation of space, we don't show the figure about the approximate ratio and range here. In the rest of the figures, we compare the CBA algorithm to the exact one by running with all the three variables. We can see in all situations, CBA algorithm has a great advantage over the exact one on running time.

6. RELATED WORK

The AKR query combines aggregate nearest neighbor with textual similarity. Existing works [9], [5], [4], [8] on aggregate nearest neighbor consider only spatial proximity. These techniques cannot directly be applied to answer AKR query for lacking of supporting of textual pruning on search space.

Recently, spatial keyword queries have been well studied. The IR^2-Tree [3] integrating R-tree and signature files, and the bR*-tree [14] combining R*-tree with bitmap support spatial searching with boolean keyword queries. The IR-tree [7], [2] attaches each MBR with inverted lists, and supports ranking queries in respect to spatial proximity and textual similarity. [2] discussed how to cluster the spatial-textual objects into the same node to speedup query processing. [10] optimized the performance of IR-tree by storing the inverted lists for occasionally used words in the buckets to optimize query processing. Some variants of the general top-k spatial keyword queries have also been studied: addressed collective spatial keyword queries [1], moving spatial keyword queries [11], approximate spatial keyword queries [12]. All existing works on spatial keyword search focus only one query point. Obviously, the AKR query, which considers multiple query points, is different from the existing problems.

7. CONCLUSIONS

In this paper, we study a new query call Aggregate Keyword Routing Query, which firstly focuses on multiple query points in spatial keyword search. The AKR poblem is NP-hard. Thus, we design an exact algorithm based on the ANN and ellipse range query. And we also design an approximate algorithm called Centre Base Assignment. This algorithm tries to find the aggregate point and task points near the centre of the query points and then uses a greed approach to assign those task points. Experimental results show the efficiency of both algorithms and the good approximation rate of the CBA algorithm.

8. ACKNOWLEDGMENTS

This research is supported in part by the National Natural Science Foundation of China (NSFC) under grant 61073001.

9. REFERENCES

[1] X. Cao, G. Cong, C. Jensen, and B. Ooi. Collective spatial keyword querying. *SIGMOD*, pages 373–384, 2011.

[2] G. Cong, C. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *VLDB*, 2(1):337–348, 2009.

[3] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. *ICDE*, pages 656–665, 2008.

[4] K. Deng, S. Sadiq, X. Zhou, H. Xu, G. Fung, and Y. Lu. On group nearest group query processing. *TKDE*, pages 295–308, 2011.

[5] F. Li, B. Yao, and P. Kumar. Group enclosing queries. *TKDE*, 2010.

[6] H. Li, H. Lu, B. Huang, and Z. Huang. Two ellipse-based pruning methods for group nearest neighbor queries. *in GIS*, pages 192–199, 2005.

[7] Z. Li, K. Lee, B. Zheng, W. Lee, D. Lee, and X. Wang. Ir-tree: An efficient index for geographic document search. *TKDE*, 2011.

[8] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. *ICDE*, pages 301–312, 2004.

[9] D. Papadias, Y. Tao, K. Mouratidis, and C. Hui. Aggregate nearest neighbor queries in spatial databases. *TODS*, 30(2):529–576, 2005.

[10] J. Rocha, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. *SSTD*, pages 205–222, 2011.

[11] D. Wu, M. Yiu, C. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. *ICDE*, 2011.

[12] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou. Approximate string search in spatial databases. *ICDE*, pages 545–556, 2010.

[13] B. Yao, M. Tang, and F. Li. Multi-approximate-keyword routing in gis data. *in GIS*, pages 201–210, 2011.

[14] D. Zhang, Y. Chee, A. Mondal, A. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. *ICDE*, pages 688–699, 2009.