

Singapore Management University Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems

School of Information Systems

4-2019

Efficient algorithms for solving aggregate keyword routing problems

Qize JIANG

Weiwei SUN


Baihua ZHENG

Singapore Management University, bhzheng@smu.edu.sg

Kunjie CHEN

DOI: https://doi.org/10.1007/978-3-030-18579-4_42

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the [Databases and Information Systems Commons](#), and the [Theory and Algorithms Commons](#)

Citation

JIANG, Qize; SUN, Weiwei; ZHENG, Baihua; and CHEN, Kunjie. Efficient algorithms for solving aggregate keyword routing problems. (2019). *Database systems for advanced applications: 24th International Conference, DASFAA 2019, Chiang Mai, Thailand, April 22-25: Proceedings*. 11447, 713-729. Research Collection School Of Information Systems.

Available at: https://ink.library.smu.edu.sg/sis_research/4390

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Efficient Algorithms for Solving Aggregate Keyword Routing Problems

Qize Jiang^{1,2,3}, Weiwei Sun^{1,2,3(✉)}, Baihua Zheng⁴, and Kunjie Chen^{1,2,3}

¹ School of Computer Science, Fudan University, Shanghai, China
{qzjiang18, wwsun, chenkunjie}@fudan.edu.cn

² Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China

³ Shanghai Institute of Intelligent Electronics and Systems, Shanghai, China

⁴ School of Information Systems, Singapore Management University, Singapore,
Singapore
bhzheng@smu.edu.sg

Abstract. With the emergence of smart phones and the popularity of GPS, the number of *point of interest (POIs)* is growing rapidly and spatial keyword search based on POIs has attracted significant attention. In this paper, we study a more sophisticated type of spatial keyword searches that considers multiple query points and multiple query keywords, namely *Aggregate Keyword Routing (AKR)*. AKR looks for an aggregate point m together with routes from each query point to m . The aggregate point has to satisfy the aggregate keywords, the routes from query points to the aggregate point have to pass POIs in order to complete the tasks specified by the task keywords, and the result route is expected to be the optimal one among all the potential results. In order to process AKR queries efficiently, we propose effective search algorithms, which support different aggregate functions. A comprehensive evaluation has been conducted to evaluate the performance of these algorithms with real datasets.

Keywords: Aggregate keyword query · Query processing · Route planning

1 Introduction

The emergence of smart phones and the popularity of GPS have spawned a revolution in mobile location-based capabilities. Many users of mobile Apps are voluntary information contributors. For example, many mobile Apps that provide *location-based services* allow users to upload and update the description of locations, e.g., Foursquare¹. With the help of these *User Generated Contents*, the number of *point of interest (POIs)* is growing rapidly. Accordingly, the searches conducted by users are no longer only about spatial features but also textual contents. A *spatial keyword query* that aims at finding a POI which is closest

¹ <https://www.foursquare.com>.

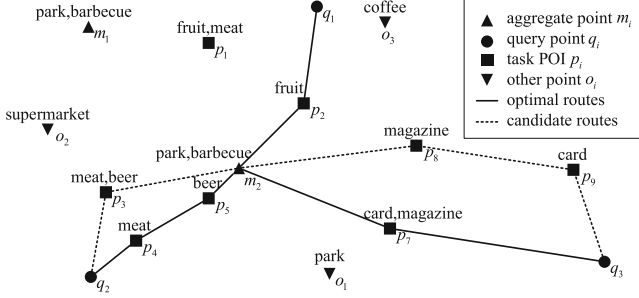


Fig. 1. Example of aggregate keyword routing

to the query point and meanwhile is relevant to the requested keywords is one example. Besides, *Aggregate Nearest Neighbor (ANN)* query [12], which finds the aggregate point with the smallest aggregate distance for a given set of spatial points and a given set of query points, is also a hot research topic.

In this paper, we study a more sophisticated type of spatial keyword searches that considers multiple query points and multiple query keywords. Before we formally introduce the query, let's consider the following scenario, as detailed in Example 1. Function $\text{keyword}(P)$ takes in a set of POIs P as input and returns the set of keywords associated with any POI $p \in P$, and function $\text{poi}(R)$ takes in a set of routes R as input and returns the set of POIs passed by any single route $r \in R$.

Example 1. *Alex, Bob, and Carol want to organize a barbecue in a park. On their ways to the barbecue venue, they need to purchase beer, meat, and fruit for the barbecue and magazines and cards to pastime during the barbecue. Assume Fig. 1 plots all the POIs. In order to facilitate planing of the barbecue, they want to conduct a search that takes in their home locations (i.e., q_1, q_2 and q_3 in Fig. 1), and two set of keywords, namely task keywords $\kappa_t = \{\text{beer, meat, fruit, magazine, card}\}$ and aggregate keywords $\kappa_a = \{\text{park, barbecue}\}$, as input, and output a gathering point m and three routes r_1, r_2, r_3 from their home locations to m respectively. To be more specific, the query is expected to satisfy following three conditions: (i) the gathering point m needs to satisfy the textual requirement represented by κ_a such as m_1 and m_2 in Fig. 1, i.e., $\text{keyword}(\{m\}) \supseteq \kappa_a$; (ii) the set of POIs passed by three routes, denoted by $\text{poi}(\cup_{i=1}^3 r_i)$, is able to satisfy the textual requirement represented by κ_t , i.e., $\text{keyword}(\text{poi}(\cup_{i=1}^3 r_i)) \supseteq \kappa_t$; and (iii) for any other answer set $\langle m', \cup_{i=1}^3 r'_i \rangle$ that satisfies the above two conditions, the aggregate distance (e.g., avg, max, sum) of $\cup_{i=1}^3 r_i$ does not exceed that of $\cup_{i=1}^3 r'_i$ in order to guarantee that the search returns the optimal solution. For example, if **maximum** is the selected aggregate distance function, m_2 and the three solid-line routes form the solution.*

The search conducted in Example 1 considers spatial condition represented by the set of query points Q and textual conditions represented by the two sets of keywords, denoted as κ_a and κ_t , and expects a single answer point, together with routes from each individual query point to the answer point. The textual

condition requires the answer point to satisfy one set of keywords κ_a , and the POIs passed by the routes to cover the other set of keywords κ_t ; while the spatial condition requires the routes to provide the optimal solution for certain given distance functions. Accordingly, this spatial keywords search is named as *Aggregate Keyword Routing (AKR)*.

AKR is challenging and time consuming. It expects an aggregate point m and a set of routes $\cup_i r_i$ from each query point to m as the result, but there are a great number of such possible routes. Consequently, the search space for the qualified routes for a given aggregate point is very large. Not to mention that the search space for the aggregate points could be large too. In fact, AKR query is NP-Hard.

The exact and approximate algorithms to solve the AKR problem are available, i.e., Task Assignment and Routing (TAR) and Center Based Assignment (CBA) proposed in [1]. Our solution outperforms TAR in terms of efficiency, while it achieves a higher accuracy than CBA. Furthermore, we take into account the **average** aggregate distance function, which was not discussed in [1], and extend all algorithms to support the new function.

In summary, this paper makes following major contributions.

- We propose novel search algorithms to process AKR queries, namely Tree Expansion (TE) and TE-ext, which are efficient and effective.
- Our algorithms support both **maximum** and **average** aggregate distance functions, and we extend the existing algorithms proposed in [1] to support the **average** aggregate distance function too.
- We conduct comprehensive experimental study to evaluate the performance of our algorithms with real POI data in different scales. The results demonstrate the efficiency and correctness of our algorithms in different aggregate functions and data scales.

The remaining of this paper is organized as follows. Section 2 reviews the related work. Section 3 formally defines the problem of *Aggregate Keyword Routing*. In Sect. 4, we present algorithm TE and its extension. For structural clarity, we only consider **maximum** as the aggregate distance function in Sect. 4; while we present the variances of algorithms to support AKR query using **average** as the aggregate distance function in Sect. 5. In Sect. 6, we analyze the worst-case time complexity of the algorithms. Section 7 reports our experimental evaluation results. Finally, we conclude our paper in Sect. 8.

2 Related Work

AKR query combines aggregate nearest neighbor (ANN) with semantic similarity. In the following, we mainly review existing works related to ANN query, spatial keyword query and AKR query.

2.1 ANN Query and Spatial Keyword Query

Aggregate Nearest Neighbor (ANN) is a traditional problem. In the literature, ANN and many of its variances have been studied. [12] proposes the ANN query,

and analyzes the average function situation. [13] considers maximum, minimum, and average aggregate distance functions under R-tree. [5] explores the ANN query in high dimension. [10, 11] solve the ANN query without indexing. [14] incorporates Voronoi diagrams into R-tree to take advantages of the strength from both structures. [15, 19] study ANN query in road networks. [4] finds a group from aggregate points and minimizes the total distance from query points to group. [16, 17] explore the merged aggregate nearest neighbor query. [23] considers road network’s Voronoi graph and solves ANN problems for both sum and maximum functions. [9] studies ANN in uncertain databases, and proposes effective pruning methods to reduce the search space.

Spatial keyword queries have also been well studied recently. [3] proposes IR²-tree, which integrates R-tree and signature files; and [21] proposes bR*-tree that combines R*-tree with bitmap and keyword MBR. IR-tree [2, 8] attaches each MBR with inverted lists, and supports ranking queries in respect to both spatial proximity and semantic similarity. [7] studies top- k aggregate nearest keyword query. [22] studies aggregate keyword nearest neighbor query, which finds the nearest neighbor with certain keywords. [6] considers direction-aware spatial keyword search, which considers keyword’s direction. [20] proposes IL-Quadtree, which is based on inverted index and the linear quadtree, and develops an efficient algorithm to tackle top- k spatial keywords search. [18] considers the multi-approximate keyword routing problem.

2.2 AKR Query

The AKR problem was first studied in [1]. The authors proposed an exact algorithm and an approximate algorithm, namely Task Assignment and Routing (TAR) and Center Based Assignment (CBA) respectively.

TAR is a two-phase algorithm, which consists of *Search and Assignment Phase* and *Task Finishing Phase*. In Search and Assignment Phase, existing ANN search algorithms are used to determine access order, i.e., the priority of the potential aggregate points, where a technique called *early termination* is applied to reduce the number of possible points. Then candidate POIs on the routes from each query point to the aggregate point are determined. In Task Finishing Phase, the algorithm searches for optimal routes and prunes the useless routes, taking advantages of the heap data structure.

CBA, on the other hand, is an approximation algorithm. It first locates an aggregate point m that is close to all query points. Afterwards, a nearest neighbor search around m generates a set of POIs which need to be passed. Finally, the result routes are computed through a heuristic approach, which starts from the shortest path and then inserts a POI into the path.

3 Problem Formulation

In the context of this paper, a POI p is associated with its location and a set of keywords $\kappa(p) = \{\delta_1, \delta_2, \dots, \delta_k\}$ with $k = |\kappa(p)|$. An *Aggregate Keyword Routing*

Table 1. Frequently used notations

Notation	Explanation
P	set of POIs
Q	set of query points
$r_{q,m}$	a route $\langle q, p_1, \dots, p_x, m \rangle$ from point q to point m
$R_{Q,m}$	a route set $\{\cup_{\forall q \in Q} r_{q,m}\}$ from query points in q to m
$R_{Q,m}^0$	the shortest route set from Q to m without passing any other point
$L(r)$	length of route r
$L_f(R)$	length of route set R with aggregate distance function f
$L_{max}(R)$	the maximum aggregate distance of R , i.e., $\text{MAX}_{\forall r \in R} L(r)$
$L_{avg}(R)$	the average aggregate distance of R , i.e., $\frac{1}{ R } \sum_{\forall r \in R} L(r)$
α	the aggregate point set
κ_a	the aggregate keyword set
κ_t	the task keyword set
δ	a single keyword
$\kappa(R)$	set of task keywords covered by set R
m	a candidate aggregate point

Query (AKR) takes in three parameters as input, i.e., a query point set Q , a task keyword set κ_t , and an aggregate keyword set κ_a . For a given task keyword set κ_t , a POI p is considered as a *task* POI iff $\kappa(p) \cap \kappa_t \neq \emptyset$. Table 1 lists the notations that will be frequently used in the rest of this paper.

In the following, we first present the terms of *route*, *route set*, and the *aggregate distance of a route set* in Definitions 1, 2, and 3 respectively. We then introduce *candidate result of an aggregate keywords routing query* in Definition 4 and present the formal definition of AKR in Definition 5.

Definition 1. A route $r_{s,m} = \langle s, p_1, \dots, p_x, m \rangle$ is a point sequence that starts from point s , 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 $\kappa(r_{s,m})$, i.e., $\kappa(r_{s,m}) = \cup_{\forall p_i \in r_{s,m}} \kappa(p_i)$.

Definition 2. Given a query point set $Q = \{q_1, q_2, \dots, q_n\}$, we use notation $R_{Q,m}$ to represent a route set $\{r_{q_1,m}, r_{q_2,m}, \dots, r_{q_n,m}\}$ and notation $\kappa(R_{Q,m})$ to capture the keyword set covered by any route in $R_{Q,m}$, i.e., $\kappa(R_{Q,m}) = \cup_{\forall r \in R_{Q,m}} \kappa(r)$.

Definition 3. The length of $R_{Q,m}$ is marked as $L_f(R_{Q,m})$, dependent on the given aggregate distance function f . For example, if **maximum** is the aggregate distance function, we have $L_{max}(R_{Q,m}) = \text{MAX}_{r_{q,m} \in R_{Q,m}} L(r_{q,m})$; if **average** is the aggregate distance function, we have $L_{avg}(R_{Q,m}) = \frac{1}{|Q|} \sum_{r_{q,m} \in R_{Q,m}} L(r_{q,m})$.

Definition 4. Given an Aggregate Keyword Routing (AKR) query $\langle Q, \kappa_a, \kappa_t \rangle$, an aggregate point m and a route set $R_{Q,m}$ form a candidate AKR result

$\langle m, R_{Q,m} \rangle$ if and only if $\kappa_a \subseteq \kappa(m)$ and $\kappa_t \subseteq \kappa(R_{Q,m})$. We denote the complete set of candidate AKR results as A_R . As the aggregate point m could be derived from $R_{Q,m}$, we use $R_{Q,m}$ to represent a candidate AKR result but skip m for brevity.

Definition 5. Given an aggregate distance function f , an Aggregate Keyword Routing (AKR) query $\langle Q, \kappa_a, \kappa_t \rangle$ is to locate the candidate result set R_{result} with the minimum $L_f(R_{result})$ value, i.e., $R_{result} = \arg \min_{R_{Q,m} \in A_R} L_f(R_{Q,m})$.

As mentioned above, processing of AKR query is very time consuming. In fact, it is NP-Hard, as presented in Theorem 1.

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

Proof. The classical Euclidean Traveling Salesman Problem (Euclidean TSP) can be reduced to AKR problem. Given an Euclidean TSP problem, it includes a start point o , and a set of points S which the salesman should travel to, with all the points in an Euclidean space. We can construct an AKR problem as follows. We assign each point in $S \cup \{o\}$ unique keywords with its location unchanged. Let the query point set $Q = \{o\}$, the aggregate keyword set $\kappa_a = \{\kappa(o)\}$, and the task keyword set $\kappa_t = \{\bigcup_{p \in S} \kappa(p)\}$. Clearly, this AKR query solves the Euclidean TSP problem. Thus, the AKR problem is NP-Hard. \square

4 Tree Expansion

As mentioned in Sect. 2, existing algorithm TAR finds the optimal solution of an AKR query. However, the algorithm is time-consuming since its time complexity grows exponentially as the size of input increases. On the other hand, CBA is efficient, but usually results in inaccurate answers. In this section, we propose a new algorithm, which is as efficient as CBA, but much more accurate than CBA.

We first find the aggregate point m , on the basis of the *minimum cover circle*, i.e., a circle that covers all query points and has the smallest radius. Assuming that the circle is centered at the point o , we search for the aggregate point m within the neighborhood of o , because the points near o probably have small maximum distance to the query points. Then, we look for suitable POIs during the process of generating the routes. We consider the selected aggregate point m as the root of a tree, and the query points are the leaves of the tree. Initially, the tree only contains the direct path from all the leaves to the root, without passing any of the task POI. We then expand the tree by adding suitable POIs to the paths. As **maximum** is the aggregate distance function, among the $|Q|$ paths from $q \in Q$ to the root, it strategically selects the shortest one to perform the expansion to avoid the case where one of the path becomes very long, until all the task keywords specified in κ_t have been covered. The algorithm is named as *Tree Expansion (TE)* to reflect the nature of the search.

Before we present the detailed algorithm, we first introduce a min-heap where each element of the heap is in the form of $\langle (s, p, e), d \rangle$. Here, (s, p, e) is a three tuple vector and d indicates the detour caused if a route from s to e needs to take

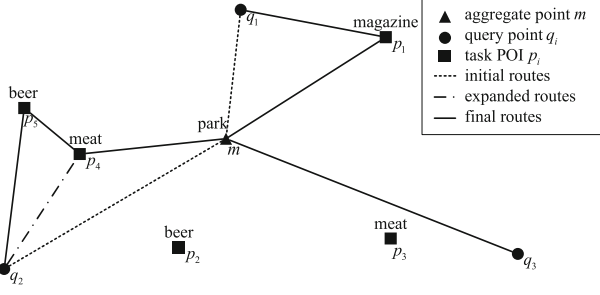


Fig. 2. Example of tree expansion

a detour at point p , i.e., $d = |s, p| + |p, e| - |s, e|$. The notation $|a, b|$ represents the Euclidean distance between two points a and b . All the elements in the heap are sorted based on ascending order of d values. We maintain $|Q|$ min-heaps to facilitate the expansions of the paths from $q \in Q$ to m .

Algorithm 1 lists the pseudo code of TE. It locates the minimum circle covering all the query points and uses the center of the circle to find an aggregate point m (line 1), initializes the result set R_{result} with $R_{Q,m}^0$, and initializes min-heap $heap[i]$ and an index array $leng[i]$ (lines 2–4). The index array is to record the length $L(r_{q_i,m})$ for each route $r_{q_i,m}$ in the current R_{result} . Thereafter, for each single query point $q_i \in Q$, it pushes the potential expansion options to the corresponding min-heap $heap[i]$ to enable the tree expansion. To be more specific, for each direct route $\langle q_i, m \rangle$, it could be expanded by adding one POI point. We locate all the POIs points $p \in P$ that cover at least one queried task keyword as the candidate points to enable the expansion, and add $\langle (q_i, p, m), d \rangle$ to the heap $heap[i]$ as a potential expansion option (lines 5–7). The real expansion is guided by routes in R_{result} and the elements in $heap$. Every time, we pick the path $r_{q_i,m} \in R_{result}$ with the shortest distance for expansion, following the top option in $heap[i]$ with the smallest detour. Assume the path $r_{q_{index},m}$ has the shortest route distance value among all the routes in R_{result} , and the top element of $heap[index]$ is e in the format of $\langle (p_s, p_m, p_e), d \rangle$. The corresponding expansion is to expand the direct link $\langle p_s, p_e \rangle$ to $\langle p_s, p_m, p_e \rangle$. Note that p_s and p_e might not be adjacent because other point(s) might have been added between p_s and p_e . Meanwhile, we also check whether $\kappa(p_m)$ is still required by κ with κ recording the task keywords not yet been covered by any route in R_{result} . We then perform the expansion if it is valid (line 12), update $leng[index]$ to reflect the extended length of the route $r_{q_{index},m} \in R_{result}$ and update κ to remove the task keywords covered by new POI p_m (line 13). In addition, the new link between p_s and p_m and that between p_m to p_e provide new expansion options. We update the heap $heap[index]$ to reflect the new expansion options (lines 14–16). The above expansion continues until all the queried task keywords have been fully covered.

We plot one example in Fig. 2 to illustrate the search. The query points $Q = \{q_1, q_2, q_3\}$, the candidate aggregate point is m , and the task keywords $\kappa_t = \{meat, beer, magazine\}$. Initially, the tree contains only three direct routes

Algorithm 1. Tree Expansion (TE) Algorithm

Input: P, Q, κ_a, κ_t
Output: R_{result}

- 1 $o := \text{getCenter}(Q), \alpha := \{p \in P \mid \kappa(p) \supseteq \kappa_a\}, m := \text{NearestNeighbor}(o, \alpha)$
- 2 $R_{result} := \cup_{\forall q_i \in Q} \langle q_i, m \rangle, \kappa := \kappa_t$
- 3 **for each** $i \in |Q|$ **do**
- 4 $\text{heap}[i] := \emptyset, \text{leng}[i] := |q_i \in Q, m|$
- 5 **for each** $p \in P$ **with** $\kappa(p) \cap \kappa_t \neq \emptyset$ **do**
- 6 **for each** $q_i \in Q$ **do**
- 7 $d := |q_i, p| + |p, m| - |q_i, m|$, push element $\langle (q_i, p, m), d \rangle$ to $\text{heap}[i]$
- 8 **while** $\kappa \neq \emptyset$ **do**
- 9 $\text{index} := \text{argmin}_{i \in |Q|} \text{leng}[i]$
- 10 $e \langle (p_s, p_m, p_e), d \rangle := \text{pop}(\text{heap}[\text{index}])$
- 11 **if** p_s **and** p_e **are adjacent in a route in** R_{result} **and** $\kappa \cap \kappa(p_m) \neq \emptyset$ **then**
- 12 update $r_{q_{\text{index}}, m} \in R_{result}$ by including p_m in the middle of p_s and p_e
- 13 $\text{leng}[\text{index}] := \text{leng}[\text{index}] + d; \kappa := \kappa - \kappa(p_m)$
- 14 **for each** $p \in P$ **with** $\kappa(p) \cap \kappa \neq \emptyset$ **do**
- 15 $d_1 := |p_s, p| + |p, p_m| - |p_s, p_m|$, push $\langle (p_s, p, p_m), d_1 \rangle$ to $\text{heap}[\text{index}]$
- 16 $d_2 := |p_m, p| + |p, p_e| - |p_m, p_e|$, push $\langle (p_m, p, p_e), d_2 \rangle$ to $\text{heap}[\text{index}]$
- 17 **return** R_{result}

from query points to m , i.e., $R_{Q,m} = \{\langle q_1, m \rangle, \langle q_2, m \rangle, \langle q_3, m \rangle\}$. At first, $\langle q_1, m \rangle$ is the shortest route in $R_{Q,m}$, so we expand $\langle q_1, m \rangle$ to $\langle q_1, p_1, m \rangle$. Then, $\langle q_2, m \rangle$ becomes the shortest route, and we expand it to $\langle q_2, p_4, m \rangle$. After expansion, $\langle q_2, p_4, m \rangle$ is still the shortest, and we further expand it to $\langle q_2, p_5, p_4, m \rangle$ to complete the search.

TE only considers one aggregate point and uses this point to compute the route set. However, the aggregate point nearest to the center of the circle that covers all the query points may not be the best choice. As a result, TE may suffer from high error rate because of this not-ideal aggregate point. In order to reduce the side effect of selecting a not-ideal aggregate point on the accuracy of the approximate result, we can extend TE via evaluating multiple aggregate points. That is, we can evaluate the aggregate points according to their proximity to the center of the circle (supported by ANN algorithms). For each of the evaluated aggregate points, we generate the result routes. Parameter $R_{candidate}$ maintains the best result route found so far. The evaluation continues until we reach an aggregate point m such that $L_{max}(R_{Q,m}^0)$ is longer than $L_{max}(R_{candidate})$, or all the aggregate points have been evaluated. To differentiate from the original TE algorithm, we name the extended approximation algorithm that evaluate multiple aggregate points as TE-ext. It is noted that TE-ext is able to find a result set with higher accuracy, as compared with TE. However, it requires longer running time in most, if not all, cases. We will report the performance comparison between TE and TE-ext in the experimental study.

Following the same idea, we extend the CBA algorithm originally proposed in [1] as well, and discover that the accuracy is improved after the extension.

In the following, we use CBA-ext to represent the algorithm after extension for convenience.

5 Average Aggregate Distance

In Sect. 4, we presented the TE algorithm, based on **maximum** aggregate distance function. In this section, we introduce **average** as another common and useful aggregate distance function, and illustrate how TE algorithm and its extension could be adapted to different distance functions.

First, we explain the necessary changes we have to make to TE in order to support AKR queries when **average** is adopted as the aggregate distance function. We need to change the routing approach in order to have the average distance of the answer route set as small as possible. The original routing approach is to assign a new task POI to the shortest route so its impact on the maximum distance of the route set could be minimized. When considering **average**, we want to look for a route with the smallest increase in terms of its distance after adding a new POI. Hence, we only need to maintain one heap instead of $|Q|$ heaps for supporting **average** aggregate distance function, and all the elements $\langle (q_i, p, m), d \rangle$ in the heap are still sorted based on ascending order of the d values. When we perform the expansion, we pop out the top element from the heap as the corresponding expansion incurs the smallest detour.

Next, we explain how we adjust the extension of TE to support AKR under **average** aggregate distance function. The main idea behind the extension remains valid regardless of the aggregate distance function adopted, as checking multiple aggregate points will not bring any harm to the accuracy. However, we want to highlight that aggregate points m shall be evaluated based on $L_{avg}(R_{Q,m}^0)$ but not $L_{max}(R_{Q,m}^0)$.

We also adapt TAR and CBA to support **average** aggregate distance, and the performance of all algorithms with respect to two distance functions is evaluated in the following experimental study.

6 Complexity Evaluation

In this section, we will analyze the worst-case time complexity of our algorithms, as well as TAR and CBA, with respect to two different aggregate distance functions.

6.1 Maximum Aggregate Distance Function

For our algorithm TE, it takes $O(|P|)$ to decide m and $O(|Q||P|\log(|P|))$ to initialize $|Q|$ min-heap $heap[i]$. Then, it invokes the **while**-loop to perform the expansion, up to $|\kappa_t|$ times. For each execution of the loop, it takes $|Q|$ to locate the index of the route $r_{q_i,m} \in R_{result}$ with the shortest length, and it inserts at most $|P|$ elements to the heap. Therefore, the maximum size of the heap

is $|\kappa_t||P|$, and the time complexity is $O(|\kappa_t||P| \log(|\kappa_t||P|))$. The overall time complexity is $O(|\kappa_t|(|Q| + |P| \log(|\kappa_t||P|)))$.

Next, we analyze the time complexity of TAR. In Search and Assignment Phase, it takes $O(|P|)$ to decide m and it checks at most $|\alpha|$ aggregate points. In Task Finishing Phase, there are $|Q|$ heaps, and most time is spent on pushing and popping heap elements. It pushes elements into heap at most $2^{|\kappa_t|}|P|$ times, and each push involves $|P|$ elements. Hence, the time complexity is $O(2^{|\kappa_t|}|P|^2|\kappa_t| \log(|P|))$. In total, the time complexity of TAR is $O(|P| + |\alpha|(2^{|\kappa_t|}|P|^2|\kappa_t||Q| \log(|P|)))$.

For CBA, it takes $O(|P|)$ to locate the aggregate point m , too. The time complexity of every execution of the loop is $O(|Q| + |\kappa_a|)$, while it is repeated for $|\kappa_t|$ times. Consequently, the total time complexity is $O(|P| + |\kappa_t|(|Q| + |\kappa_t|))$.

When we also consider extensions, we need to multiply the routing cost by $|\alpha|$. As the result, the time complexity of TE-ext is $O(|\alpha||\kappa_t|(|Q| + |P| \log(|\kappa_t||P|)))$ and the time complexity of CBA-ext is $O(|P| + |\alpha||\kappa_t|(|Q| + |\kappa_t|))$.

6.2 Average Aggregate Distance Function

For TE, it only needs to use one heap during the search and the expansion is guided by the top element of the heap. Accordingly, the time complexity is changed to $O(|\kappa_t||P| \log(|\kappa_t||P|))$. Based on the worst-case time complexity, TE runs faster with **average** aggregate distance function.

For TAR, its time complexity remains unchanged when the aggregate distance function is changed from **maximum** to **average**. However, we want to highlight that its real performance under **maximum** is better than that under **average**, as some of its optimizations become less stronger under **average**.

For CBA, it needs to check all $|Q|$ routes in order to find the one with the smallest detour to accommodate a new POI, so its time complexity is changed to $O(|P| + |Q||\kappa_t|^2)$, and it becomes slower when aggregate distance function is changed from **maximum** to **average**.

We should point out that TAR and the extension version of approximate algorithms use ANN algorithm to enumerate aggregate points. As ANN is not the focus of our paper, we skip the time complexity of the ANN algorithms.

7 Experimental Evaluation

In this section, we evaluate the performance of all algorithms using real datasets. In the following, we first introduce the experimental settings and then report the experimental results. All the experiments are performed on a Windows 10 machine with an Intel Core i7-4790 CPU and 32 GB memory.

7.1 Experimental Setup

Dataset. We use two real POI datasets, namely **Shanghai** and **New York**, with their main characteristics presented in Table 2. Shanghai dataset has in total

Table 2. Characteristics of the dataset

Dataset	# of POIs	# of keywords	# of distinct keywords
Shanghai from Amap	1,229,313	2,950,336	1,361
New York from FourSquare	132,263	249,918	711

Table 3. Query parameters

Parameter	Range
Query Point Number ($ Q $)	2, 4 , 6, 8, 10, 12
Aggregate Point Number ($ \alpha $)	1, 10, 100, 1000 , 10000
Task Keyword Number ($ \kappa_t $)	2, 4, 6 , 8, 10
Task Keyword Popularity (ρ_{κ_t})	1, 10, 100, 1000 , 10000
Area Size (AS)	0.3%, 0.9%, 3%, 9% , 30%, 90%

1, 229, 313 POIs, covering 2, 950, 336 keywords (i.e., 2.4 keywords per POI). The number of distinct keywords is 1,361, so every keyword corresponds to 903 POIs on average. New York dataset has in total 132, 263 POIs, covering 249, 918 keywords (i.e., 1.89 keywords per POI). The number of distinct keywords w.r.t. New York dataset is 711. The number of POIs corresponding to one keyword w_i could be very different from that of another keyword w_j , e.g., the number of convenience stores is much larger than the number of museums. In addition, the number of keywords associated with one POI could be also very different from that associated with another POI, e.g., a shopping mall POI could have a long list of keywords such as *shopping*, *dinning*, *bank*, *cinema*, *supermarket*, while a rail station could have only one keyword.

Queries and Parameters. We randomly generate queries with selected parameters to evaluate the performance of different algorithms on various settings. Each AKR query has three input parameters, the query points Q , the aggregate keyword set κ_a , and the task keyword set κ_t . Based on these three input parameters, we set five parameters to control query generation, as listed in Table 3. The values with bold numbers represent the default settings. Parameter $|Q|$ determines the number of query points, ranging from 2 to 12; parameter $|\alpha|$ specifies the qualified aggregate points which is dependent on κ_a ; parameter $|\kappa_t|$ represents the number of queried task keywords; parameter ρ_{κ_t} indicates the popularity of a query task keyword with its value representing the total number of POIs in P that cover this keyword; parameter AS determines a square-shaped subarea S_{sub} within which query points are randomly generated, and it is represented as the ratio of the size of the subarea S_{sub} to that of the whole search space. For simplicity, we assume all the query task keywords in κ_t share the same popularity.

Algorithms. We implement our proposed TE and TE-ext algorithms, as well as TAR, CBA and CBA-ext, which are proposed in [1], in total 5 algorithms.

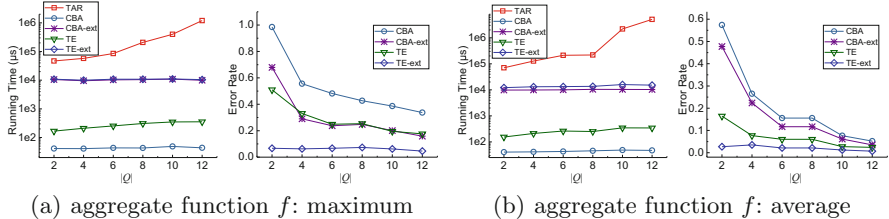


Fig. 3. Search performance vs. parameter $|Q|$

We test their performance for both **maximum** aggregate distance function and **average** aggregate distance function.

7.2 Experimental Results

For every parameter setting, we randomly generate 100 queries for testing and report their average performance. All the algorithms need to form the candidate set α for the aggregate points. Therefore, we exclude the cost of forming α from all the experimental figures. We adopt **running time** and **error rate** as the main performance metrics with error rate set to $\frac{L_f(R_{\text{appro}}) - L_f(R_{\text{exact}})}{L_f(R_{\text{exact}})}$. R_{appro} refers to the result set returned by an approximate algorithm, while R_{exact} refers to the exact search result. Let the error rate be E , the accuracy is $\frac{1}{(1+E)}$. In other words, a lower error rate is equivalent to a higher accuracy. When we investigate the impacts of different parameters, we only report the results corresponding to Shanghai dataset, as the observations made from New York datasets are similar. We will briefly present the results of New York dataset at the end of this section.

Impact of $|Q|$. We first evaluate the impact of the size of the query point set on the performance via changing $|Q|$ from 2 to 12. Figure 3 depicts the result. The performance gap in different algorithms is obvious and consistent across all the testing cases. TAR’s running time grows exponentially when $|Q|$ increases, and we also observe that the running time of TAR grows even faster with **average** function than **maximum** function. On the other hand, $|Q|$ does not change the running time of CBA, TE and their extensions much; while the increase of $|Q|$ does help to reduce the error rate of approximate algorithms and their extended versions. This is because with more query points, the average number of POIs passed by each route from a query point to the aggregate point decreases, which makes it easier for the approximate algorithms to find a better answer. However, there is not much room for further improvement in TE-ext, since the accuracy has already been very high when $|Q|$ is small. In general, the approximate algorithms run much faster than their extended versions but their error rates are also higher.

Impact of $|\alpha|$. Secondly, we change the parameter $|\alpha|$ from 1 to 10,000 and report the result in Fig. 4. For TAR, CBA-ext and TE-ext, $|\alpha|$ mainly affects the number of aggregate points enumerated from α . On the other hand, both CBA and TE only evaluate one aggregate point, so $|\alpha|$ only effects the time required

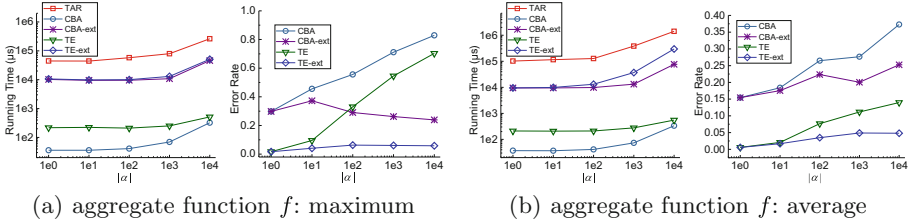


Fig. 4. Search performance vs. parameter $|\alpha|$

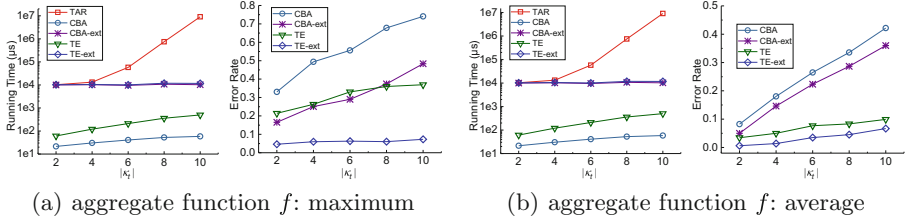
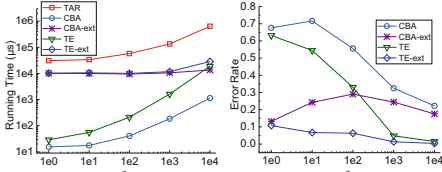


Fig. 5. Search performance vs. parameter $|\kappa_t|$

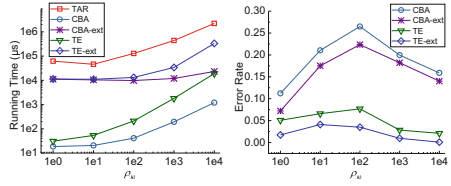
to find the aggregate point nearest to the center of query points. We can observe that when $|\alpha|$ values are small, an increase of $|\alpha|$ value does not necessarily increase the running time, as many aggregate points could be filtered out. Note that we select the query points within a small subarea while the aggregate points are distributed across the entire search space. However, as $|\alpha|$ becomes much larger (e.g., $|\alpha| = 1000$), a further increase of $|\alpha|$ actually increases the number of aggregate points that require evaluation and hence the running time. On the other hand, unlike $|Q|$, the increase of $|\alpha|$ does not decrease but increase the error rate.

Impact of $|\kappa_t|$. We now study the impact of parameter $|\kappa_t|$. The result is plotted in Fig. 5. Compared with $|Q|$, $|\kappa_t|$ has an even bigger impact on the performance of TAR. However, its impact on CBA and TE is less significant. This is because both algorithms only evaluate one route set, for the selected aggregate point, without enumerating all the possible route sets. As $|\kappa_t|$ becomes bigger, the route set needs to pass by more POI points which increases the cost of generating one route. In general, TE and TE-ext are able to achieve a much lower error rate.

Impact of ρ_{κ_t} . Next, we study the impact of the popularity ρ_{κ_t} of query task keywords with the result plotted in Fig. 6. It is observed that ρ_{κ_t} value affects the performance of CBA and TE significantly. In addition, when ρ_{κ_t} reaches very big values, CBA and TE do not have much advantage over their extended versions in terms of running time. Similarly, their extended versions do not demonstrate much advantage over their original algorithms in terms of error rate. This is because when the density of task POIs becomes very high, there are always task POIs around a given aggregate point. In addition, we also observe that when **maximum** is selected as the aggregate distance function, the increase of ρ_{κ_t}

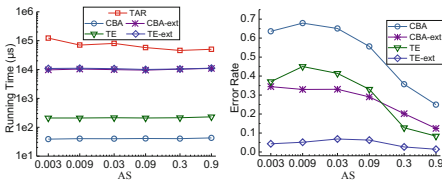


(a) aggregate function f : maximum

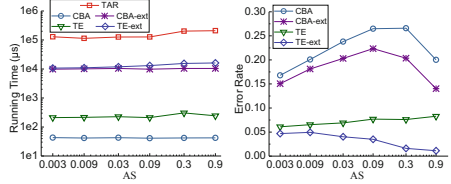


(b) aggregate function f : average

Fig. 6. Search performance vs. parameter ρ_{κ_t}



(a) aggregate function f : maximum



(b) aggregate function f : average

Fig. 7. Search performance vs. parameter AS

value helps to reduce the error rate. This is because as the density of task POIs increases, the position of aggregate point becomes less sensitive and the impact of a wrong aggregate point becomes smaller. However, when **average** is selected as the aggregate distance function, the error rate corresponding to very small ρ_{κ_t} values is low. The reason behind is that as the density of task POIs is very low, the average length of the exact route is expected to be very large which becomes less sensitive to the errors. As ρ_{κ_t} increase its values, the length of the result set decreases and the error rate increases. However, once ρ_{κ_t} reaches a large value, a further increase of its value helps to decrease the error rate.

Impact of AS. Last but not least, we analyze the impact of parameter AS, which reflects the distance between query points. The results are plotted in Fig. 7. We observe that the values of AS do not change the algorithms' running time much. As for the error rate, an increase of AS value helps to improve the error rate performance when **maximum** is selected as the aggregate distance function. This is because as points are further from each other, the aggregate point that is nearest to the center of the query points is expected to have a better accuracy.

Performance Evaluation on Different Datasets and Statistical Results.

We test all algorithms over two different datasets, Shanghai and New York dataset. Note Shanghai dataset is around 10 times larger than New York dataset. Table 4 reports the average performance of different algorithms under two different datasets. We report the results based on four metrics, including *average running time (ART)*, *average error rate (AER)*, *maximum running time rate (MRTR)*, and *maximum error rate (MER)*. Let QU be the set of queries in our evaluation, and let t_i and e_i be the running time and error rate of each query

Table 4. Statistical data in Shanghai and New York

Metrics		ART (ms)		AER		MRTR		MER	
		Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.
Shanghai	TAR	590.46	3153.53	-	-	31.39	18.90	-	-
	CBA	0.29	0.30	0.50	0.22	2.91	3.03	2.32	1.31
	CBA-ext	43.96	79.17	0.26	0.18	8.26	8.96	1.91	1.00
	TE	3.09	3.57	0.28	0.07	2.99	3.21	1.71	0.74
	TE-ext	27.13	191.93	0.05	0.02	2.71	7.03	0.56	0.19
New York	TAR	34.44	181.76	-	-	51.73	32.34	-	-
	CBA	0.01	0.01	0.62	0.29	8.33	5.81	2.25	1.22
	CBA-ext	0.37	0.53	0.37	0.24	8.60	6.04	2.01	1.22
	TE	0.19	0.18	0.35	0.09	5.69	7.27	1.88	0.77
	TE-ext	0.76	3.25	0.09	0.03	6.89	5.33	1.07	0.52

$qu_i \in QU$ respectively. ART is set to $\frac{\sum_{qu_i \in QU} t_i}{|QU|}$; AER is set to $\frac{\sum_{qu_i \in QU} e_i}{|QU|}$; MRTR is set to $\max_{qu_i \in QU}(\frac{t_i}{\bar{t}})$, where \bar{t} denotes the ART of queries which have same parameters as qu_i ; and MER is set to $\max_{qu_i \in QU}(e_i)$. In general, the algorithms become efficient but inaccurate if maximum aggregate distance is used. TAR is 15 times slower than TE-ext, and its MRTR is high so its performance is not stable. CBA is the most efficient, but the least effective with the accuracy far below that of TE-ext algorithm. Compared with TAR, TE-ext has high accuracy (ranging from 92% to 98%) too, with respect to all two aggregate distance functions.

8 Conclusion

In this paper, we study the AKR problem, and propose novel approximate algorithms to process AKR queries. The algorithms support both maximum and average aggregate distance functions, two commonly used distance functions in practice. Our TE-ext algorithm is efficient since it is 15 times faster than the exact algorithm, whereas it is effective and achieves the accuracy of at least 91%. In this paper, our AKR query only returns one result, while we plan to extend the search to return top- k results in the near future. In addition, we are also exploring AKR query on road networks.

Acknowledgment. This research is supported in part by the National Natural Science Foundation of China under grant 61772138, the National Key Research and Development Program of China under grant 2018YFB0505000, and the National Research Foundation, Prime Ministers Office, Singapore under its International Research Centres in Singapore Funding Initiative.

References

1. Chen, K., Sun, W., Tu, C., Chen, C., Huang, Y.: Aggregate keyword routing in spatial database. In: SIGSPATIAL, pp. 430–433. ACM (2012)
2. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. *Proc. VLDB Endowment* **2**(1), 337–348 (2009)
3. De Felipe, I., Hristidis, V., Rishé, N.: Keyword search on spatial databases. In: ICDE, pp. 656–665. IEEE (2008)
4. Deng, K., Sadiq, S., Zhou, X., Xu, H., Fung, G.P.C., Lu, Y.: On group nearest group query processing. *TKDE* **24**(2), 295–308 (2012)
5. Li, F., Yao, B., Kumar, P.: Group enclosing queries. *TKDE* **23**(10), 1526–1540 (2011)
6. Li, G., Feng, J., Xu, J.: Desks: direction-aware spatial keyword search. In: ICDE, pp. 474–485. IEEE (2012)
7. Li, Z., Xu, H., Lu, Y., Qian, A.: Aggregate nearest keyword search in spatial databases. In: APWEB, pp. 15–21. IEEE (2010)
8. Li, Z., Lee, K.C., Zheng, B., Lee, W.C., Lee, D., Wang, X.: Ir-tree: an efficient index for geographic document search. *TKDE* **23**(4), 585–599 (2011)
9. Lian, X., Chen, L.: Probabilistic group nearest neighbor queries in uncertain databases. *TKDE* **20**(6), 809–824 (2008)
10. Luo, Y., Chen, H., Furuse, K., Ohbo, N.: Efficient methods in finding aggregate nearest neighbor by projection-based filtering. In: Gervasi, O., Gavrilova, M.L. (eds.) ICCSA 2007. LNCS, vol. 4707, pp. 821–833. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74484-9_70
11. Luo, Y., Furuse, K., Chen, H., Ohbo, N.: Finding aggregate nearest neighbor efficiently without indexing. In: Proceedings of the 2nd international conference on Scalable information systems. p. 48. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (2007)
12. Papadias, D., Shen, Q., Tao, Y., Mouratidis, K.: Group nearest neighbor queries. In: ICDE, pp. 301–312. IEEE (2004)
13. Papadias, D., Tao, Y., Mouratidis, K., Hui, C.K.: Aggregate nearest neighbor queries in spatial databases. *TODS* **30**(2), 529–576 (2005)
14. Sharifzadeh, M., Shahabi, C.: Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries. *Proc. VLDB Endowment* **3**(1–2), 1231–1242 (2010)
15. Sun, W.W., Chen, C.N., Zhu, L., Gao, Y.J., Jing, Y.N., Li, Q.: On efficient aggregate nearest neighbor query processing in road networks. *JCST* **30**(4), 781–798 (2015)
16. Sun, W., et al.: Merged aggregate nearest neighbor query processing in road networks. In: CIKM, pp. 2243–2248. ACM (2013)
17. Sun, W., Chen, C., Zheng, B., Chen, C., Zhu, L., Liu, W., Huang, Y.: Fast optimal aggregate point search for a merged set on road networks. *Inform. Sci.* **310**, 52–68 (2015)
18. Yao, B., Tang, M., Li, F.: Multi-approximate-keyword routing in GIS data. In: SIGSPATIAL, pp. 201–210. ACM (2011)
19. Yiu, M.L., Mamoulis, N., Papadias, D.: Aggregate nearest neighbour queries in road networks. *TKDE* **17**(6), 820–833 (2005)
20. Zhang, C., Zhang, Y., Zhang, W., Lin, X.: Inverted linear quadtree: efficient top K spatial keyword search. *TKDE* **28**(7), 1706–1721 (2016)

21. Zhang, D., Chee, Y.M., Mondal, A., Tung, A.K., Kitsuregawa, M.: Keyword search in spatial databases: towards searching by document. In: ICDE, pp. 688–699. IEEE (2009)
22. Zhang, P., Lin, H., Gao, Y., Lu, D.: Aggregate keyword nearest neighbor queries on road networks. *GeoInformatica* **22**(2), 237–268 (2018)
23. Zhu, L., Jing, Y., Sun, W., Mao, D., Liu, P.: Voronoi-based aggregate nearest neighbor query processing in road networks. In: SIGSPATIAL, pp. 518–521. ACM (2010)