Processing Multiple-User Location-Based Keyword Queries*

Yong WANG1(a), Xiaoran DUAN1, Xiaodong YANG1, Yiquan ZHANG1, Nonmembers, and Xiaosong ZHANG1, Member

SUMMARY Geosocial networking allows users to interact with respect to their current locations, which enables a group of users to determine where to meet. This calls for techniques that support processing of Multiple-user Location-based Keyword (MULK) queries, which return a set of Point-of-Interests (POIs) that are ‘close’ to the locations of the users in a group and can provide them with potential options at the lowest expense (e.g., minimizing travel distance). In this paper, we formalize the MULK query and propose a dynamic programming-based algorithm to find the optimal result set. Further, we design two approximation algorithms to improve MULK query processing efficiency. The experimental evaluations show that our solutions are feasible and efficient under various parameter settings.

key words: multiple-user location-based query, location-based keyword query, query processing

1. Introduction

With the proliferation of Geosocial networking applications (e.g., Facebook Place, Meetup, and Google Map), users can easily share their real-time locations to help them decide on a meeting activity. However, doing so calls for techniques that support processing Multiple-user Location-based Keyword (MULK) queries, which return a set of Point-Of-Interests (POIs) such as restaurants, tourist attractions, hotels, and entertainment services that are ‘close’ to a group of users and can provide them with potential options at the lowest extra expense. For instance, suppose a group of Meetup users wish to meet at a restaurant (e.g., one serving their favorite meal) from different locations so that they spend the least time traveling. Further, when the first person to arrive finds it is full, they can conveniently choose another restaurant of interest from the result set that involves the least rerouting time. As another example, two troops position differently and plan to amass at a supply point in a battle field. To avoid risks, they wish to consuming minimum time in concentrating. In addition, when they reach the rally point and reconnoiter it occupied by the enemy, they can march to another supply point as plan B with least switching time.

Traditional location-based keyword queries that return a set of irrelevant POIs, such as the k-Nearest Neighbors (kNN) keyword query [1], the Skyline keyword query [2], [3], and the Top-k keyword query [4], [5], can hardly meet the needs of an entire group of users, nor can their goal be achieved by issuing a location-based top-k keyword query followed by an kNN query.

Answering a MULK query requires a hybrid index that combines information retrieval and location indexes. To verify whether a node is relevant for a set of query keywords, the current approaches evaluate each node; however, doing so incurs a long query processing time. The work closest to processing a MULK query is the spatial group keyword query [6], [7], which is treated as a set-covering problem and has been proved to be NP-complete. The objects in its result set meet a user’s requirements collectively. Put differently, the user must visit all the returned objects to satisfy his needs because no single relatively satisfying object exists that meets his demands. In contrast, a MULK query must return a set of POIs in which each POI meets the needs of a group of users. Hence, unlike a spatial group keyword query, a MULK query cannot be treated as a set-covering problem.

In this paper, we propose technical solutions for processing MULK queries. Our contributions are as follows: (1) we propose the MULK query; (2) we propose a dynamic programming based algorithm to find MULK queries’ optimal result set; (3) we propose two approximation algorithms to improve MULK query processing efficiency; and (4) we conducted a set of experimental evaluations to show the feasibility and efficiency of our solutions under various parameter settings.

The rest of this paper is organized as follows. We propose the MULK query in Sect. 2 and propose the corresponding algorithms in Sect. 3. We present the experimental evaluations in Sect. 4. We review the related works in Sect. 5. We draw brief conclusions in Sect. 6.

2. Problem Statement

Let \( \Psi \) be a set of keywords. Without loss of generality, we consider a set of spatial POIs, \( \Omega, |\Omega| = s \). Each POI \( p \in \Omega \) can be represented as a tuple \( p = \langle id, \lambda, \psi \rangle \), where \( p.id \) is \( p \)'s identifier, \( p.\lambda \) is its location (latitude and longitude), and \( p.\psi \)
is a set of keywords, \( p, \psi \subset \Psi \), that describes the POI (e.g., the recipe of a restaurant). Figure 1 depicts a dataset of nine restaurants \( \Omega = \{p_0, p_1, \ldots, p_9\} \), each of which represents a POI associated with a set of keywords.

### 2.1 Multiple-User Location-Based Keyword Query (MULK)

A multiple-user location-based keyword (MULK) query \( q \) is of the form \( \langle \lambda, \psi \rangle \), where \( q, \lambda \) is a set of the locations of a group’s users, \( q, \lambda \subset \Lambda \), and \( \psi \) is a set of keywords provided by the group’s users, \( |q, \psi| = m \). The MULK query finds a set of \( k \) POIs \( \chi, \chi \subset \Omega \), such that \( \text{Cost}(\chi, q) \) is minimized and \( |\chi| = k \). Figure 2 shows an example of query.

Next, we present the cost functions. Given a set of POIs \( \chi \), a cost function has two weighted components:

\[
\text{Cost}(\chi, q) = \alpha \cdot C_1(\chi, q) + (1 - \alpha) \cdot C_2(\chi)
\]

where \( C_1(\chi, q) \) represents the distances between the POIs in \( \chi \) and the group’s users, while \( C_2(\chi) \) characterizes the inter-object distance among the POIs in \( \chi \). A smaller \( C_1(\chi, q) \) indicates that the query users are closer to the POIs in the result set, while a smaller \( C_2(\chi) \) indicates that the POIs in the result set are closer to each other. Therefore, users prefer a lower \( \text{Cost}(\chi, q) \) in the result set of a MULK query. The parameter \( \alpha \in [0, 1] \) is used to balance \( C_1(\chi, q) \) and \( C_2(\chi) \). For simplicity, we will not consider \( \alpha \) in the remainder of this paper. Many functions could be used for \( C_1(\chi, q) \) and \( C_2(\chi) \). For example, such a function can be written as

\[
\text{Cost}(\chi, q) = C_1(\chi, q) + C_2(\chi)
\]

\[
= \max_{p \in \chi} \text{mDist}(p, \lambda, q, \lambda) + \max_{p \in \chi} TR(p, \psi, q, \psi)
\]

\[
+ \max_{p, p' \in \chi} (\text{Dist}(p, \lambda, p', \lambda))
\]

\[
= \max_{p \in \chi} \max_{q, \lambda, q, \lambda} \|p, \lambda, q, \lambda\|
\]

where \( TR(p, \psi, q, \psi) \) is the text relevance between \( p, \psi \) and \( q, \psi \). \( \text{Dist}(p, \lambda, q, \lambda) \) denotes the distance between two POIs \( p_i \) and \( p_j \), and \( \text{mDist}(p, \lambda, q, \lambda) \) represents the maximum distance from the POI to the query users’ locations. Here, \( \|p, \lambda, p', \lambda\| \) indicates the Euclidean distance between POI \( p_i \) and \( p_j \), while \( \|p, \lambda, q, \lambda\| \) indicates the Euclidean distance between POI \( p \) and user \( u_i \). \( \text{NGD}(x, y) \) is the Normalized Google Distance (NGD) [8], which calculates the text relevance of two words from web pages based on the Google search engine.

Referring to Fig. 2, the locations of the group’s users \( q, \lambda = \{q_0, \lambda_0, q_1, \lambda_1, q_2, \lambda_2\} \) are also shown in Fig. 1. Each user provides their own keywords; these users tend to find three candidate restaurants \( (k = 3) \) with minimized \( \text{Cost}(\chi, q) \) where they can meet. Considering Eq. (1) as \( \text{Cost}(\chi, q) \), we obtain the final result set \( \chi = \{p_0, p_1, p_2\} \) (also shown in Fig. 1).

### 3. Processing MULK Queries

#### 3.1 Dynamic Programming-Based Solution

The straightforward solution would be to enumerate each combination of POIs and calculate its cost function. With \( k \) POIs selected from \( \Omega \), \( |\Omega| = s \), at the most \( C_s^k \) combinations would need to be considered. Therefore, its time complexity would be \( \max(O(nkC_s^k), O(k^2C_s^k)) \), where the first term is the time complexity of \( C_s^k(\chi, q) \), and the second term is that of \( C_2(\chi) \). However, this yields a factorial running time based on the number of POIs, which is obviously very expensive.

We observed that \( k \) is quite small compared with the dataset cardinality \( s \). Hence, we propose an exact algorithm based on dynamic programming to find the \( \chi \) with the lowest \( \text{Cost} \). The main idea in our algorithm is to check each state to find the lowest \( \text{Cost} \). A state is defined with its sub-problem, namely \( \text{Cost}(i, j) \), which means the minimum \( \text{Cost} \) of selecting \( j \) POIs from the first \( i \) POIs. Let the result set of state \( \text{Cost}(i, j) \) be \( \chi_{i,j} \). Then, the state transition equation is

\[
\text{Cost}(i, j) = \min_{q, \lambda, q, \lambda} \|p, \lambda, q, \lambda\|
\]

...
Where \( W_i \) denotes the \( \text{Cost} \) when \( p_i \) is inserted into the result set \( \chi \). According to Eq. (1), \( W_i \) is calculated as

\[
W_i = \max_{p \in [p_{i-1}, p_i]} m\text{Dist}(p, \lambda, q, \lambda) + \max_{p' \in [p_{i-1}, p_i]} TR(p, \psi, q, \psi) + \max_{p' \in [p_{i-1}, p_i]} \text{Dist}(p, \lambda, p', \lambda) \quad j > 1
\]

According to Eq. (2), each state \( \text{Co}[i, j] \) must be checked before the optimal result set \( \chi \) can be found. The computational complexity of \( W_i \) is \( \max(\mathcal{O}(n), \mathcal{O}(k^2)) \) because the maximum value of \( j \) is \( k \). The computational complexity of the dynamic programming approach, denoted as \( C_{dp} \), is

\[
C_{dp} = k \cdot C_{ro} + (s - 1) \cdot C_{co} + (s - 1)(k - 1) \cdot C_{plus}
\]

where \( C_{ro} \), \( C_{co} \) and \( C_{plus} \) are the time costs involved in calculating \( \text{Co}[j, \ell] \) and are \( O(n), O(s) \) and \( \max(O(n), O(k^2)) \), respectively.

The computational complexity of the state transition Eq. (2) is \( \max(O(nsk), O(sk^2)) \). Obviously, this requires pseudo polynomial time; therefore, it is intolerable when \( s \) is large, which is typical in a real spatial dataset. To solve this problem, we propose two approximation algorithms to retrieve the result set without examining each possible state.

3.2 Approximation Algorithms

3.2.1 Preliminaries: The IR-Tree

The IR-tree [9], [10] is essentially an R-tree [11] extended with inverted files. Each POI \( p \), termed as a data entry, consists of its identifier \( p, \text{pid} \), its location \( p, \lambda \), and its keyword set \( p, \psi \). Further, we group the POIs by their locations (their latitude and longitude). Each group, termed as a minimum bounding rectangle (MBR), is indexed as a leaf node. Each leaf node \( e_l \) contains a set of data entries and a pointer to an inverted file. An inverted file, termed as \( \text{IFL}_{e_l} \), is a vocabulary of all the distinct keywords that appear in its data entries. Each internal node of an IR-tree is a larger MBR containing a set of child MBRs and a pointer to an inverted file, termed as \( \text{IFL}_{e_i} \), which is the union of its children’s inverted files.

Figure 4 shows an IR-tree built from the dataset in Fig. 1. For example, the leaf node \( e_3 \) stores the MBR \( N_1 \), containing the data entries \( p_5, p_4 \) in \( N_3 \), and a pointer to its inverted file \( \text{IFL}_{e_3} \). The internal node \( e_6 \) stores the MBR \( N_6 \), the child MBRs \( N_3, N_4 \), and a pointer to its inverted file \( \text{IFL}_{e_6} \). The associated inverted file indexes are shown in Fig. 5. For simplicity, we utilize the symbols shown in Fig. 3 to represent the POIs’ keywords in Fig. 1.

3.2.2 Approximation Algorithms 1: MULK-Appr01

The main idea of the approximation algorithm named \textit{MULK-Appr01} is to use the best-first search to find the result set \( \chi \) with the lowest \( \text{Cost} \) according to Eq. (1).

We calculate the value of each data entry \( p_j \) in the IR-tree as follows:

\[
\text{Val}_{p_j} = \begin{cases} 
TR(p_j, \psi, q, \psi) + m\text{Dist}(p_j, \lambda, q, \lambda) & |\chi| = 0 \\
TR(p_j, \psi, q, \psi) + m\text{Dist}(p_j, \lambda, q, \lambda) + \max_{p, q \in \chi} \text{Dist}(p, \lambda, p, \lambda) & |\chi| > 0
\end{cases}
\]

where \( \chi \) is the current partial result set, and \( |\chi| \) is the cardinality of \( \chi \).

There are various approaches [12] to computing the...
distance between an MBR and a POI. However, because these approaches are not the major concern of this paper, for simplicity, we present our solution as follows. For each node \( e_j \) in IR-tree, suppose the corresponding MBR is \( N_j \), and its rectangular vertices are \( N_j^1, N_j^2, N_j^3 \) and \( N_j^4 \). These four vertices allow us to express \( N_j \) conveniently even though they may not be the locations of actual POIs. Figure 6 shows an example of MBR \( N_6 \) (the red points).

The value of node \( e_j \) is computed as follows:

\[
Val_e_j = \begin{cases} 
TR(e_j, \psi, q, \psi) & |\psi| = 0 \\
TR(e_j, \psi, q, \psi) \cdot mDist(N_j^k, \lambda, q, \lambda) & |\psi| > 0
\end{cases}
\]

where

\[
TR(e_j, \psi, q, \psi) = \min_{o \in e_j} TR(o, \psi, q, \psi)
\]

in which \( o_i \) is either a child node of \( e_j \) or is a POI when \( e_j \) is a leaf node.

Note that the two Eqs. (4) and (5) calculate the value of a POI or a node, which is different from the \( Cost \) in Eq. (1). The latter is used to calculate the \( Cost \) of the final result set \( \chi \) with \( k \) POIs.

**Lemma 1:** Given an IR-tree, the value \( Val \) of a leaf or internal node is the lower bound of the value \( Val \) of any of its children.

**Proof:** Given a node \( e_j \) and any of its child objects \( o_i \),

1. if object \( o_i \) is a leaf or internal node, its rectangular vertices can be denoted as \( N_i^k \) (\( k = 1, 2, 3, 4 \)). We have \( TR(e_j, \psi, q, \psi) \leq TR(o_i, \psi, q, \psi) \), \( \min mDist(N_i^k, \lambda, q, \lambda) \leq \min mDist(N_j^k, \lambda, q, \lambda) \) and \( \min_{p_i, e_j} Dist(N_j^k, \lambda, p_i, \lambda) \leq \min_{p_i, e_j} Dist(N_i^k, \lambda, p_i, \lambda) \) from Eqs. (5) and (6). Hence, \( Val_{e_j} \leq Val_{o_i} \).
2. if object \( o_i \) is a POI, we have

\[
TR(e_j, \psi, q, \psi) \leq TR(o_i, \psi, q, \psi), \min mDist(N_j^k, \lambda, q, \lambda) \leq mDist(o_i, \lambda, q, \lambda) \text{ and } \min_{p_i, e_j} Dist(N_j^k, \lambda, p_i, \lambda) \leq \max_{p_j, e_k} Dist(o_k, \lambda, p_j, \lambda) \text{ based on Eqs. (4), (5) and (6).}
\]

Similarly, \( Val_{e_j} \leq Val_{o_i} \).

Ultimately, we obtain \( Val_{e_j} \leq Val_{o_i} \).

**Figure 6** The distance between MBR \( N_6 \) and user’s location \( q, \lambda \)

Lemma 1 guarantees the correctness of the best-first strategy for retrieving the optimal \( \chi \) with the lowest \( Cost \). Algorithm 1 shows the pseudocode for the \textit{MULK-Appro1} algorithm. The steps are explained in more detail below.

We maintain a min-priority heap \( \mathcal{H} \) to hold the index nodes that are to be visited in ascending order of the objects’ values. \( \mathcal{H} \) is initially empty, and the root node of the IR-tree is added to \( \mathcal{H} \) first. Then, in each step, the top value from \( \mathcal{H} \) pops up to be checked.

- (1) When the top value from \( \mathcal{H} \) is an internal or a leaf node of the IR-tree, its child objects’ values are calculated using Eqs. (5) and (4). Then, these objects are added to \( \mathcal{H} \) based on their values.
- (2) When the top value of \( \mathcal{H} \) is a POI (which means it has the lowest value among all the objects in the \( \mathcal{H} \) heap), it is added directly into \( \chi \). The values of all the objects in \( \mathcal{H} \) are upgraded with respect to new \( \chi \). Then, the min-priority heap \( \mathcal{H} \) is also upgraded with a new value.

The recursion stops when the cardinality of \( \chi \) meets \( k \), namely \( |\chi| = k \). Using the result set \( \chi \), the \( Cost(\chi, q) \) is ultimately calculated by Eq. (1).

Considering the example in Fig.1, the corresponding IR-tree is demonstrated in Fig.4. After initialization, the min-priority heap \( \mathcal{H} \) is loaded with the root node \( e_{root} \). At step 6, we calculate the values of \( e_5 \) and \( e_6 \) respectively, and insert them into heap \( \mathcal{H} \). At step 10, \( p_0 \) is added into the result set \( \chi \). Then, the values of the objects in \( \mathcal{H} \), namely \( p_1, p_2, e_2 \) and \( e_6 \), are upgraded based on the new result set \( \chi \) at step 11. As the algorithm proceeds, more POIs are popped from \( \mathcal{H} \) and inserted into \( \chi \). Finally, after several iterations, \( \chi = \{p_0, p_1, p_2\} \) is obtained. Its cost function is computed as: \( Cost_\chi = C_1(\chi, q) + C_2(\chi) = \|p_0, q, \lambda\| + NGD(“shishimi”, “lobster”) + \|p_0, p_3, \lambda\| = 8.95 + 0.68 + 5 = 14.63 \).

We proceed to show that \textit{MULK-Appro1} is within an approximation factor of 3. We denote the \( Cost \) of the result set \( \chi \) returned by \textit{MULK-Appro1} as \( Cost_{\chi_{approx}} \), and denote the cost of the optimal result set as \( Cost_{\chi_{opt}} \).
Algorithm 1: Approximation algorithm 1: MULK-Appro1

Input: q: the query
Ω: the POI dataset
k: the required number of result POIs

Output: χ: the result set
1: initiate the result set and the min-priority heap: χ = 0, H = ∅
2: H ← root node of IR-tree
3: while |χ| < k do
4: current ← H.pop
5: if current is an internal node or a leaf node then
6: calculate the values of its child objects using Eqs. (4) and (5)
7: H ← the children of current
8: else
9: if current is a POI then
10: χ ← current
11: upgrade the values Val of the nodes and POIs in H with respect to the updated χ using Eqs. (4) and (5)
12: end if
13: end if
14: end while

Theorem 1: For a given query q, the Cost of the result set returned by MULK-Appro1 is at most 3 times the Cost of the optimal result set (i.e., Cost_{Appro1} ≤ 3Cost_{OPT}).

Proof: In result set χ returned by MULK-Appro1, we assume that POI p_j has the greatest spatial distance from users, and we let d_j = mDist(p_j, λ, q, λ). Then, the largest possible inter-object distance between POIs in χ is 2d_j. Suppose that POI p_j has the largest text relevance based on the users’ submitted keywords, namely tr_j = TR(p_j, φ, q, φ). Obviously, Cost_{OPT} ≥ (d_j + tr_j). Therefore, referring to Eq. (1), we can derive the following:

Cost_{Appro1} ≤ d_j + tr_j + 2d_j
≤ 3(d_j + tr_j) ≤ 3Cost_{OPT}

The proof is completed.

3.2.3 Approximation Algorithm 2: MULK-Appro2

Based on MULK-Appro1 and the IR-tree, we also propose a more advanced approximation algorithm, denoted as MULK-Appro2, whose main idea is to repeatedly invoke MULK-Appro1 for the result sets that have a lower Cost.

The pseudo-code of MULK-Appro2 is shown in Algorithm 2. First, we execute MULK-Appro1 to obtain the initial result set, χ, and its Cost_χ. Then, we select the POI p_k with the highest value Val from χ and remove p_k from Ω. Further, we insert p_k into the set Δ_χk as the initial result for a new round of MULK-Appro1 on the dataset Ω with the parameters q and k−1. Finally, we compare Cost_χ with Cost_{Δχ_k} using Eq. (1).

- (1) if Cost_{Δχ_k} is smaller than Cost_χ, we replace χ with Δ_χk. Then, we start a new iteration;
- (2) otherwise, the iteration stops, and χ is returned as the final result.

Recall the example in Fig.1. MULK-Appro1 is first invoked and returns χ = {p_0, p_1, p_2} in step 1; p_0 is selected because it has the highest value Val in χ and is inserted into X_{p_0} in steps 2 to 5. By performing MULK-Appro1 for the result set X_{p_0} in step 7, X_{p_0} = {p_0, p_1, p_2} and its cost function are obtained in step 8. We have Cost_{X_{p_0}} = C_1(χ_{p_0}, q) + C_2(χ_{p_0}) = ∥p_0, λ, q, λ∥ + NGD("mutton shashlik", "lobster") = 8.95 + 0.81 + 2.83 = 12.59. However, because Cost_{X_{p_0}} ≤ Cost_χ = 13.68, we re-execute recursively from step 3 to obtain a new result set Δ_χ_{p_0}. Finally, we return χ = {p_0, p_1, p_2} as the final result at step 13.

Next, we show that the result of MULK-Appro2 is within an approximation factor of 1.8. We denote the Cost of the result set χ returned by MULK-Appro2 as Cost_{Appro2}, and the Cost of optimal result set as Cost_{OPT}.

Theorem 2: For a given query q, the Cost of the result set returned by MULK-Appro2 is at most 1.8 times the cost of the optimal result set (i.e., Cost_{Appro2} ≤ 1.8Cost_{OPT}).

Proof: We give the detailed proof in Appendix.

4. Experimental Evaluations

4.1 Experimental Settings

Dataset. We employed both synthetic and real datasets in the experimental evaluations.

The synthetic dataset, OM, was constructed using newly generated virtual POIs. Each virtual POI is equipped with a random location whose latitude and longitude are within the range [0, 1000]. The tags of the buildings in Atlantic City were taken from OpenStreetMap † and randomly attached to these virtual POIs to provide textual descriptions. The OM dataset consists of 1,000,000 POIs with 41,974 distinct keywords.

The real dataset, GN, was extracted from the U.S. Board on Geographic Names ‡, in which each object is a location with a geographic name (e.g., valley). Each record is regarded as a POI. We selected several items from each record as keywords to describe the POIs (e.g.,

†www.openstreetmap.org
‡geonames.usgs.gov


FEATURE\_NAME, COUNTY\_NAME). The GN dataset consists of 1,868,821 POIs with 222,409 distinct keywords.

**Queries.** We randomly picked \(n\) POIs from a dataset and regard their locations as those of the query users, \(q, \lambda\). Then we randomly chose a specified number of keywords from these POIs as the query keywords \(q, \psi\). The selected POIs are temporarily excluded from the dataset during query execution. We executed each of the three algorithms 100 times and then evaluated their performances using their average execution times and average approximation ratios. The average approximation ratio is the average value of the approximation ratio and demonstrates the accuracy of approximate algorithms. In addition, because the IR-tree uses a disk-based implementation, it must be swapped in and out of memory frequently during the experiments. Because this swapping time cost is not a major consideration in this paper, we have excluded it from the execution times for these experiments.

**Environment.** The experiments were executed on a personal computer equipped with an Intel Core i5-3450 processor, 16 GB RAM and running Windows 7 x64 Server Pack 1. The code for the experiments was written and executed using JDK 1.7 64-bit.

### 4.2 Experimental Evaluations

Figure 7 shows the scalability of the three algorithms with respect to an increasing \([\Omega]\) for each dataset. In the following figures, “DP” represents the dynamic programming. On both datasets, as the number of POIs increases, the gap between the average execution time of MULK-Appro1, MULK-Appro2 and dynamic programming becomes clearer. The execution time of each three algorithm increases based on its computational complexity. Referring to Fig.7, when the number of POIs is small (e.g., no larger than 10 K), the execution time of the dynamic programming approach is acceptable (less than 10 seconds).

However, MULK-Appro1 and MULK-Appro2 represent better options when dealing with large data sets (e.g., larger than 100 K). We believe that this work will help enable practical applications that involve large data set (e.g., larger than 100 K) in the future.

Figure 8 shows that the average execution time changes with \([q, \lambda]\) on both datasets. The average execution time rises as \([q, \lambda]\) increases, which conforms with the computational complexity analysis of dynamic programming in Sect.3.1. The execution times of MULK-Appro1 and MULK-Appro2 also increase as \(q, \lambda\) increases based on Eqs. (5) and (4). In Fig.9, dynamic programming performs best in retrieving the lowest average Cost; however, its execution time is excessive. MULK-Appro1 constructs the result set \(\chi\) most efficiently. Nonetheless, it has the highest average Cost. Compared with the other two algorithms, by invoking MULK-Appro1, MULK-Appro2 can achieve a better average Cost in an acceptable execution time.

As shown in Fig.10, the average execution time rises as \([p, \psi]\) increases, which conforms with Eqs. (4) and (5) in Sect.2. In Fig.11, we can observe approximation ratio results similar to those in Fig.9, which shows the effectiveness of MULK-Appro2.

Figure 12 shows how the average execution time changes with \([q, \psi]\) on both datasets. When 24 users each provide between 1 and 5 keywords, \([q, \psi]\) increases from 24 to 120, respectively. As \([q, \psi]\) increases, more time is required to compute \(TR(p, \psi, q, \psi)\) based on Eqs. (4) and (5). Thus, the average execution time of the three algorithms increases as \([q, \psi]\) increases. In Fig.13, we can observe approx-
the datasets do not influence the effectiveness of our algorithms. Put differently, these results validate the extensive applicability of our algorithms.

We also compared MULK-Appro1 and MULK-Appro2 against MAXM-A1[7] and T2A2[6]. Because we obtained qualitatively similar results on both datasets in the preceding experiments, here, we perform comparison experiments only on dataset GN due to space limitations. Note that MAXM-A1 and T2A2 permit only single users to perform queries, while MULK-Appro1 and MULK-Appro2 queries are intended for use by a group of users. We evaluated MULK-Appro1 and MULK-Appro2 using different POI and user-location distributions.

Figure 14 shows the average execution times of the compared algorithms in these settings. Uniform distribution means that POIs and users are randomly distributed in an area. The centralized distribution denotes that most POIs and users are clustered within a limited area. Note that a centralized distribution sharply reduces the average execution times of the algorithms. This result occurs because when the POIs are centralized, it is easy to find a nearby group of POIs to satisfy the users’ requirements. For example, the restaurants in Wangfujing Street have a centralized distribution; therefore, we can conveniently retrieve the results that answer users’ requests. When the group users are centralized, a MULK query can be considered as a single user query, which also consumes less time than does a multi-user query.

Considering that the detailed execution circumstances of MAXM-A1 and T2A2 were not reported, and are therefore unknown, MULK-Appro1 and MULK-Appro2 consume less time than T2A2 when the users and POIs are centralized. Moreover, when both users and POIs are centralized, the time costs of MULK-Appro1 and MULK-Appro2 are acceptable compared with that of MAXM-A1.

5. Related Work

Cao et al. [6], [7] defined the problem of retrieving a group of spatial web objects such that the group’s keywords collectively cover the query’s keywords and such that the retrieved objects are nearest to the query location and have the lowest inter-object distances. Gao et al. [13] extended this problem into road networks.

Compared with the works in [6], [7], [13], the cost functions in [6], [7] consider only spatial distance on the condition that the union of the objects’ keywords covers that of the query. In contrast, our cost function takes both spatial distance and textual relevance into consideration. Hence, the MULK query cannot be reduced to a set-covering problem.

Specifically, the dynamic programming algorithm proposed in the work [6], [7] divides their states based on a subset of \( q, \psi \). It has exponential running time in terms of \( |q, \psi| \). As for answering the MULK query, its state is defined by a subset of \( \Omega \) and consumes exponential running time with regard to \( |\Omega| \). Further, the two approximation algorithms T2A1 and T2A2 proposed in the work [6], [7] cannot be applied to answering the MULK query either. Our approximation algorithms differ from theirs in two aspects:

- Our parameters for retrieving POIs are different from theirs. Our parameters are \( q, \lambda \) and \( q, \psi \), which are constant during query processing, while in theirs, the key-
word component \( q, \psi \) changes as POIs are added into result set.

- Our method of computing a node’s value is also different from theirs. Our method considers the text relevance \( TR(p, \psi, q, \psi) \) between the POIs’ keywords and the query keywords, while theirs focuses only on the number of POI keywords that overlap with the query keyword, namely \( |p, \psi \cap q, \psi| \).

6. Conclusion

A MULK query can be applied in many scenarios, such as group travel, party planning, etc. It would definitely be interesting to apply these techniques to disaster planning (e.g., evacuation procedures or search-and-rescue operations). In this paper, we formalize the MULK query and propose a dynamic programming-based algorithm to find the optimal result set. To improve query efficiency, we further propose two approximation algorithms. The results of comprehensive experimental evaluations show that our solutions are not only feasible, but also efficient and robust under various parameter settings. We believe that this work will help enable practical applications involving multiple-user query processing on spatial datasets.

References


Appendix: The Proof of Theorem 2

Lemma 2: Given a query \( q \) and POI \( p_k \) with worst text relevance in \( \chi \), the Cost of result set \( \chi \) returned by MULK-Appro1 \( p_k \) in MULK-Appro2, denoted by Cost\(_{Appro1-p_k}\), is in the following range: \( mDist(p_k, \lambda, q, \lambda) + Dist(p_k, \lambda, p_{max-k}, \lambda) + TR(p_k, \psi, q, \psi) \leq Cost_{Appro1-p_k} \leq mDist(p_k, \lambda, q, \lambda) + 3Dist(p_k, \lambda, p_{max-k}, \lambda) + TR(p_k, \psi, q, \psi) \), where \( p_{max-k} \) is the furthest POI from \( p_k \) in the result set \( \chi \).

Proof: (1) Since \( p_k \) is included in result set \( \chi \) returned by MULK-Appro1 \( p_k \), \( mDist(p_k, \lambda, q, \lambda) \) is the lower bound on the spatial distance, \( Dist(p_k, \lambda, p_{max-k}, \lambda) \) is the lower bound on the inter-object distance of \( \chi \), and \( TR(p_k, \psi, q, \psi) \) is also the lower bound on the text relevance of \( \chi \). Hence, \( mDist(p_k, \lambda, q, \lambda) + Dist(p_k, \lambda, p_{max-k}, \lambda) + TR(p_k, \psi, q, \psi) \leq Cost_{Appro1-p_k} \).

(2) \( mDist(p_k, \lambda, q, \lambda) + Dist(p_k, \lambda, p_{max-k}, \lambda) \) is the maximum spatial distance between \( q \) and \( \chi \), the inter-object distance of \( \chi \) will not exceed \( 2Dist(p_k, \lambda, p_{max-k}, \lambda) \), and \( TR(p_{trk}, \psi, q, \psi) \) is also the lower bound on the text relevance of \( \chi \). Hence, \( mDist(p_k, \lambda, q, \lambda) + 3Dist(p_k, \lambda, p_{max-k}, \lambda) + TR(p_{trk}, \psi, q, \psi) \leq Cost_{Appro1-p_k} \).

Lemma 3: Given a query \( q \), let \( \chi_{OPT} \) be the optimal result set and POI \( p_j \in \chi_{OPT} \). Let \( p_{max-j} \) denote the furthest POI to \( p_j \) in MULK-Appro1 \( p_j \). It holds that \( Cost(\chi_{OPT}) \geq mDist(p_j, \lambda, q, \lambda) + Dist(p_j, \lambda, p_{max-j}, \lambda) + TR(p_j, \psi, q, \psi) \).

Proof: Since POI \( p \in \chi_{OPT} \), \( mDist(p_j, \lambda, q, \lambda) \) and \( TR(p_j, \psi, q, \psi) \) can serve as the lower bound on spatial distance and text relevance between \( q \) and \( \chi_{OPT} \) respectively. Since \( p_{max-j} \) is the furthest POI to \( p_j \) in MULK-Appro1 \( p_j \), apart from POIs in \( \chi_{OPT} \), any other POI whose spatial distance and text relevance are better than or equal to \( mDist(p_{max-j}, \lambda, q, \lambda) \) and \( TR(p_{max-j}, \psi, q, \psi) \), has worse inter-object distance than \( Dist(p_j, \lambda, p_{max-j}, \lambda) \). We examine whether POI \( p_{max-j} \) is included in \( \chi_{OPT} \). If it is, \( Dist(p_j, \lambda, p_{max-j}, \lambda) \) can serve as the lower bound on spatial distance between POIs in \( \chi_{OPT} \). Otherwise, a POI \( p_k (p_k \notin \chi_{OPT}) \) with better spatial distance and text relevance is inserted into \( \chi_{OPT} \) to replace \( p_{max-j} \). Since POI \( p_k \)’s spatial distance and text relevance are better than that of \( p_{max-j} \), its inter-object distance is worse than \( Dist(p_j, \lambda, p_{max-j}, \lambda) \). Put differently, \( Dist(p_j, \lambda, p_{max-j}, \lambda) \) can also serve the lower bound of inter-object distance in \( \chi_{OPT} \).

We present the proof of theorem 2 as:

Proof: Let POI \( p_k \) and \( p_{trk} \) be the POI with the furthest-away spatial distance and the worst text relevance from \( q \)
in MULK-Appro1. Let POI $p_j$ be a POI contained in the optimal result set $p_j \in \chi_{opt}$. Let POI $p_{max-j}$ be the furthest-away POI from $p_j$ in MULK-Appro1.

(1) Consider the case where $mDist(p_j, \lambda, q, \lambda) + TR(p_j, \psi, q, \psi) \geq mDist(p_k, \lambda, q, \lambda) + TR(p_k, \psi, q, \psi)$

According to lemma 3, we get $Cost_{opt} \geq mDist(p_j, \lambda, q, \lambda) + Dist(p_j, \lambda, p_{max-j}, \lambda) + TR(p_j, \psi, q, \psi)$.

Since MULK-Appro2 invokes MULK-Appro1 iteratively for lower Cost, we obtain that $Cost_{Approx2} \leq Cost_{Approx1}$ and $Cost_{Approx2} \leq Cost_{Approx1}$. In terms of lemma 2, we get $Cost_{Approx2} \leq Cost_{Approx1}$.

$$\frac{Cost_{Approx2}}{Cost_{opt}} \leq \frac{mDist(p_j, \lambda, q, \lambda) + 3mDist(p_k, \lambda, q, \lambda) + 3TR(p_k, \psi, q, \psi)}{mDist(p_j, \lambda, q, \lambda) + Dist(p_j, \lambda, p_{max-j}, \lambda) + TR(p_j, \psi, q, \psi)} \leq 1.8 \text{ with Eqs. (A-3) and (A-4).}$$

$$\frac{Cost_{Approx2}}{Cost_{opt}} \leq 1.8$$

$\square$

Yong Wang is an associate professor of University of Electronic Science and Technology of China. His research interests include P2P network measurements and modeling, spatial database and query authentication.

Xiaoran Duan is a M.S. candidate in University of Electronic Science and Technology of China. He received his bachelor in 2015 from the Harbin Engineering University. His major research interests include query processing, and query authentication, etc.

Xiaodong Yang is a M.S. candidate in University of Electronic Science and Technology of China. He received his bachelor in 2016 from the University of Electronic Science and Technology of China. His major research interests include query processing, and query authentication, etc.
Yiquan Zhang is a M.S. candidate in University of Electronic Science and Technology of China. He received his bachelor in 2016 from the University of Electronic Science and Technology of China. His major research interests include query processing, and query authentication, etc.

Xiaosong Zhuang is a professor of University of Electronic Science and Technology of China. His research interests include network security, software security and reliability, etc.