The VLDB Journal
https://doi.org/10.1007/s00778-019-00583-8

REGULAR PAPER

Efficient processing of moving collective spatial keyword queries

Hongfei Xu[1] · Yu Gu[1] · Yu Sun[2] · Jianzhong Qi[3] · Ge Yu[1] · Rui Zhang[3]

Received: 18 February 2019 / Revised: 11 October 2019 / Accepted: 15 October 2019

Abstract

As a major type of continuous spatial queries, the moving spatial keyword queries have been studied extensively. Most existing studies focus on retrieving single objects, each of which is close to the query object and relevant to the query keywords. Nevertheless, a single object may not satisfy all the needs of a user, e.g., a user who is driving may want to withdraw money, wash her car, and buy some medicine, which could only be satisfied by multiple objects. We thereby formulate a new type of queries named the moving collective spatial keyword query (MCSKQ). This type of queries continuously reports a set of objects that collectively cover the query keywords as the query moves. Meanwhile, the returned objects must also be close to the query object and close to each other. Computing the exact result set is an NP-hard problem. To reduce the query processing costs, we propose algorithms, based on safe region techniques, to maintain the exact result set while the query object is moving. We further propose two approximate algorithms to obtain even higher query efficiency with precision bounds. All the proposed algorithms are also applicable to MCSKQ with weighted objects and MCSKQ in the domain of road networks. We verify the effectiveness and efficiency of the proposed algorithms both theoretically and empirically, and the results confirm the superiority of the proposed algorithms over the baseline algorithms.

Keywords Moving query · Collective spatial keyword query · Safe region · Query processing algorithms

1 Introduction

As a major type of moving queries, the *moving spatial keyword queries* (MSKQ) have been studied extensively [1–4]. Given a set of static spatio-textual objects each with a location and textual description (e.g., points of interest and geo-tagged documents), a typical MSKQ considers the location of a moving object and a set of keywords as arguments and aims to continuously return a spatial-textual object that *best* matches these arguments, e.g., close to the query object and containing all the query keywords. Such a query could come from a tourist who wants to find the nearest "seafood restaurant" while walking in a city.

In some applications, we observe that users' needs may be better satisfied by multiple objects *collectively* instead of a *single* object. Consider a scenario where Alice is driving in a foreign city. She wants to withdraw money, wash her car, and buy some medicine. Her needs can only be satisfied by multiple objects, e.g., ATMs, car-washing facilities, and pharmacies. For convenience, she would prefer locations within walking distance from each other. When her car is being washed, she can walk to an ATM and a pharmacy. As her needs are not very urgent, she wants to see more candidate locations while driving until she finds a satisfactory result (i.e., location set that suits her preference). As another example, nowadays many PC games (e.g., The Legend of Zelda[1] and The Elder Scrolls[2].) have large game

✉ Yu Gu
guyu@mail.neu.edu.cn

Hongfei Xu
xuhongfei_neu@163.com

Yu Sun
ysun@twitter.com

Jianzhong Qi
jianzhong.qi@unimelb.edu.au

Ge Yu
yuge@mail.neu.edu.cn

Rui Zhang
rui.zhang@unimelb.edu.au

1 College of Computer Science and Engineering, Northeastern University, Shenyang, China

2 Twitter, Inc., San Francisco, CA, USA

3 The Department of Computing and Information Systems, The University of Melbourne, Melbourne, Australia

1 https://www.zelda.com/.

2 https://elderscrolls.bethesda.net/.

Published online: 01 November 2019

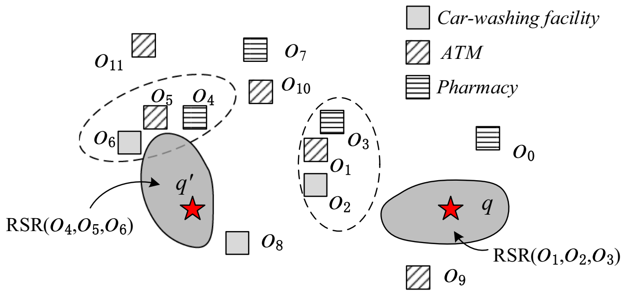


Fig. 1 An MCSKQ query

maps, which include millions of spatial objects. In these games, players need to complete gaming tasks to level-up their gaming avatars. Game producers may provide a location recommendation service for players, especially those who are new to the game, to help them get an easier start and to retain them in the game. While the game is running, that service can continuously recommend nearby task target locations for the players, where they can collectively go for, e.g., gold farming, trading, and skill training.

To address the need for such collective answers to MSKQ, in this paper, we propose a new type of queries, the *moving collective spatial keyword query* (MCSKQ). As the query object is moving, this query *continuously* returns a set of objects satisfying the following conditions: (1) the union of textual descriptions of the objects covers (i.e., contains all) the query keywords; (2) the objects are spatially close to the query object; and (3) the objects are also spatially close to each other.

Figure 1 illustrates an example of the MCSKQ q for Alice. There are 12 spatio-textual objects $o_0, o_1, \ldots, o_{11}$ represented by squares. The set $\{o_1, o_2, o_3\}$ will be returned as the answer to the query q since it contains all the query keywords, i.e., ATM, car-washing facility, and pharmacy, all close to q and close to each other. If Alice is not satisfied with the current answer, she could continue driving. When she moves to q', the answer will become $\{o_4, o_5, o_6\}$.

Efforts [5–8] have been made to solve the *static* collective spatial keyword query (CSKQ) problem. Given a query $q = \langle q.\lambda, q.\psi \rangle$, where $q.\lambda$ is a location and $q.\psi$ is a set of keywords, CSKQ is to find a set S of objects such that S covers $q.\psi$ and has the minimum cost measured by a given function. However, existing techniques for static CSKQ do not suit MCSKQ. This is because when a static CSKQ algorithm is used, keeping the result set up-to-date when the query object is moving requires constant recomputation, incurring expensive computation costs.

To overcome the drawbacks, our initial focus is to reduce the frequency of recomputing a static CSKQ as much as possible, which has the following challenges:

Challenge 1 Existing *safe region* models for moving queries are inapplicable. The safe region technique is commonly used in processing moving queries as it can reduce the recomputation frequency effectively [9–11]. As long as the query is still in the safe region, it is guaranteed that the current query answer remains correct. However, to the best of our knowledge, existing safe region models are defined for individual objects. They are not suitable for the sets of spatio-textual objects collectively covering the query keywords. Specifically, existing approaches use the concept of *dominant region*[3] to compute the safe region. Suppose that object o^* is the top-1 query answer. The safe region of o^* is the intersection of the dominant region of o^* to all other objects (think of a Voronoi cell). Nevertheless, it is much expensive to compute the dominant region of two object sets when each set contains several objects (think of a higher-order Voronoi cell). Therefore, it may be too expensive to compute the safe region for MCSKQ following existing approaches straightforwardly.

Challenge 2 The MCSKQ algorithms are required to work with both exact and approximate static CSKQ algorithms. Recently, many effective algorithms for static CSKQ [7,12] have been proposed to cope with different application priorities (reviewed in Sect. 2.1), which can be divided into exact algorithms and approximate algorithms. An exact algorithm aims to compute the exact result set while an approximate algorithm can efficiently compute an approximate result set with precision bounds (e.g., 3-, 1.8-, and 1.375-approximation ratios). Since exact results and approximate results are generated by different strategies, to reduce the recomputation frequency, both exact and approximate incremental approaches for MCSKQ are required to maintain these results (i.e., to retain exactness of the result). Also, the approximate incremental algorithm needs to be sufficiently generic to be adapted to various approximate algorithms with different approximation ratios.

Challenge 3 Object weights and the underlying road networks add further challenges. The object weight can be a user-contributed rating or the text relevance of the object to the query. In this case, we need to consider object weights when evaluating a set of objects. In many real-life applications, the objects are located on a road network where we need to use the road network distance to object distance computation. Therefore, the approaches for MCSKQ need to be general and flexible and can be easily adapted to these two variants.

To address the above challenges, we propose novel approaches including the following key techniques:

(1) We propose the concept of *relaxed safe region* (RSR), based on which we devise two exact algorithms for

[3] Given a query q and two objects, o_i and o_j, the dominant region of o_i to o_j is a region such that if q is in the region, o_i is a better answer than o_j.

MCSKQ. We call a set of objects a feasible set if it covers the query keywords. Given a query q, we derive the RSR of the exact result set using the top-k feasible sets of q. When q moves to q', as long as q' is in this RSR, the exact result set of q' remains to be one of the top-k feasible sets of q. It is only when the query object moves out of the RSR that a query recomputation is needed. This greatly reduces the recomputation frequency. We present an example of RSR in Fig. 1 where the gray elliptic regions are the RSRs for $\{o_1, o_2, o_3\}$ and $\{o_4, o_5, o_6\}$, respectively. Based on RSR, we devise two exact algorithms for MCSKQ which adopt the exact algorithms in recomputation and return the exact result set continuously when the query moves: One uses a single RSR to maintain the exact result set and the other uses multiple RSRs to collectively maintain the exact result set which further reduce the recomputation frequency. The two algorithms also work for MCSKQ with weighted object and MCSKQ on road networks.

(2) We propose the concept of *keyword-based Voronoi neighbor set* (KVNS), based on which we devise two approximate algorithms for MCSKQ with precision bounds. Given a query q, an approximate CSKQ algorithm computes a result set S that consists of the nearest neighbors of q for each query keyword and gives a 3-approximation result. We use KVNS to maintain these nearest neighbors when the query moves. Specifically, for an object o containing keyword t, the t-Voronoi neighbor set of o consists of the objects which are near to o and containing keyword t. When the query moves, as long as o is closer to the query object than the objects in the t-Voronoi neighbor set of o, o is still the nearest neighbor for keyword t. Conceptually, the t-Voronoi neighbor set of o defines a safe region as large as the order-1 Voronoi cell on keyword t for o. Thus, when q moves to q', if each object in S is closer to q' than the object's KVNS, S is still valid. Based on KVNS, we propose an approximate algorithm for MCSKQ adopting the aforementioned approximate CSKQ algorithm in recomputation. This algorithm uses an incremental maintenance strategy to continuously maintain an approximate result with a precision bound. This algorithm also works for the two variants of MCSKQ (i.e., weighted object and road networks). In addition, combining KVNS and RSR techniques, we design another approximate algorithm with higher precision bounds (i.e., 1.8- and 1.375-approximation ratios), which uses other approximate CSKQ algorithms in recomputation. The two approximate algorithms can simultaneously reduce the cost of recomputation and the recomputation frequency, and hence achieve even higher efficiency.

In summary, we make the following contributions:

– We propose a new query type, the moving collective spatial keyword query (MCSKQ). This query continuously returns a set of objects that collectively cover the query keywords when the query moves. The returned objects are also close to the query object and meanwhile close to each other.
– We approach MCSKQ by reducing the frequency of recomputing static CSKQ. We propose two exact algorithms and two approximate algorithms comprising the aforementioned key techniques to achieve this goal.
– We adapt the proposed algorithms to further handle MCSKQ with weighted objects and MCSKQ on road networks, respectively.
– We conduct extensive experiments using real-world data sets to evaluate the performance of the proposed algorithms. The results confirm the effectiveness and efficiency of the proposed algorithms.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 formulates the MCSKQ problem. Sections 4 and 5 present the proposed algorithms for MCSKQ. Section 6 adapts the proposed algorithms to variants of MCSKQ. Section 7 reports experiments and Sect. 8 concludes the paper.

2 Related work

2.1 Collective spatial keyword queries

The collective spatial keyword query (CSKQ) finds a set of the objects that collectively cover the query keywords and have the minimum cost measured by a case-specific cost function [5,8,13,14]. The *MaxMax* cost function [5] is one of the most popular used in CSKQ, which defines the cost to be a linear combination of the maximum distance between the query object and any returned object and the maximum pairwise distance among the returned objects. In this paper, we mainly focus on the MaxMax cost function. Effective algorithms for CSKQ with MaxMax have been proposed, which can be divided into three categories: *exact* algorithms, *low-approximation* (L-Appro) algorithms, and *high-approximation* (H-Appro) algorithms.

Exact algorithms The state-of-the-art exact algorithms are MaxMax-Exact [12] and MaxSum-Exact [7], which use different branch-and-bound strategies to enumerate feasible sets in the object space.

L-appro algorithms The MaxMax-Appro1 algorithm [12] is the only L-Appro algorithm. It computes query's nearest neighbors (NNs) for each query keyword and returns the result set containing all these NNs as an approximate solution, which gives a 3-approximation result.

H-appro algorithms The state-of-the-art approximate algorithms for CSKQ are MaxMax-Appro2 [12] and MaxSum-Appro [7], which produce approximate result sets under 1.8- and 1.375-approximation ratios, respectively. The two algorithms both mainly consist of two procedures. The first procedure is to use the aforementioned L-Appro algorithm to find a feasible set as a candidate. The second procedure finds the set having the minimum cost among all *special feasible sets* and treats it as another candidate. For MaxMax-Appro2, a special feasible set consists of an object containing the most infrequent query keyword and its NNs covering the other query keywords. For MaxSum-Appro, a special feasible set instead consists of an object containing at least one query keyword and its NNs. The final result set is the better one of the two candidates. Therefore, this result set is guaranteed to be no worse than that returned by the L-Appro algorithm.

Many studies [8,13,15,16] have worked on CSKQ with other cost functions. Long et al. [7] investigate a new instantiation of CSKQ with the cost function *diameter*, which defines the cost to be the diameter of the query object and the returned objects. Chan et al. [13] propose a new type of CSKQ with the cost function *maximum dot size* which captures both some spatial distances between objects and a query and the *inherent* costs of the objects (e.g., the admission fee of a POI). Zhang et al. [17] study the *level-aware collective spatial keyword* (LCSK) query. The keyword level can be used to capture the level of tourist attractions, hotels, or the rescue ability of equipments. The LCSK query asks for a group of objects that cover the query keywords collectively with keyword level constraints and minimize the cost function, which takes into account both the cost of objects and the spatial distance. Gao et al. [14] define CSKQ on road networks. Based on connectivity-clustered access method (CCAM) index [18], they develop two approximate algorithms with approximation bounds and one exact algorithm to support CSKQ processing. Su et al. [8] address a *group-based collective keyword* (GBCK) query problem on road networks. It aims to find a region containing a set of POIs that covers the query keywords where the POIs are close to the group of users and also close to each other.

All these studies related to CSKQ are static queries. None of the above techniques takes into account continuous queries, and our focus, i.e., effectively reducing the recomputation frequency for a continuously moving query, has not been studied.

2.2 Moving queries

Moving queries or continuous queries have attracted much attention with the popularity of location-based services. For a continuous query, the query position is moving while the objects can be either static or moving. Our study focuses on the former case.

Previous studies [9,19–21] have worked on *moving k nearest neighbor* (MkNN) query, which handles a moving query object q and a set of static data objects $\mathcal{O}$. When q is moving, the MkNN query reports its k-nearest neighbors (kNN) continuously. Nutanong et al. [9] exploit both the query location and data objects to construct a safe region. Li et al. [20] propose the *influential neighbor set* approach, which is the state-of-the-art MkNN algorithm. This approach uses *safe guarding objects* rather than safe regions, which are a small set of data objects surrounding the current kNN set. Other studies [22,23] investigate the continuous *reverse kNN* queries, which continuously retrieve all the data objects that have the query object as one of their closest object when the query and data objects move freely. The *continuous range* query [24–26] is also an important type of continuous queries, which continuously retrieves all the data objects in a query region. However, these previous studies do not take into account the keyword information of objects as in MCSKQ.

Previous studies [27–29] have worked on continuous queries over spatial-textual streams, which continuously report the objects from streaming spatial-textual data that satisfy the query's spatial and textual predicates. Mahmood et al. [27] propose a distributed system *Tornado* to process spatial keyword data streams in real time. Salgado et al. [30] propose the continuous range spatial keyword queries over moving spatio-textual objects (CRSK-mo queries), which continuously monitor moving spatio-textual objects (e.g., customers) for multiple long running range queries with respect to query objects (e.g., restaurants and hospitals). The authors use the spatial and textual upper bounds between queries and objects to form safe zones (at the client-side) and buffer regions (at the server-side) to reduce both communication and computational overhead. Guo et al. [31] study the continuous moving range queries over dynamic event streams which are continuously published by local businesses. In these studies, the spatio-textual objects are incoming or moving. This differs from our work where the query object is moving and the spatio-textual objects are static.

Wu et al. [4] propose the *moving top-k spatial keyword* (MkSK) query, which considers both spatial locations and keywords, and enables a mobile user to be continuously aware of the k spatial web objects that best match a query with respect to both location and text relevancy. Huang et al. [2] propose a more general ranking function for MkSK query. They use hyperbolas to represent the safe region and devise efficient incremental algorithms to compute the safe region with effective pruning techniques and indexing structures, e.g., IR-tree. Guo et al. [1] propose the notion of *safety road segment* and design a framework to process MkSK on road networks using a similarity function formed by spatial proximity and textual relevance. Zheng et al. [32] study the *keyword-aware continuous k nearest neighbor* (CkNN) on road networks, which is to find the kNN results that sim-

Table 1 Frequently used symbols

Notation	Description
q	An MCSKQ
q'	A new MCSKQ
S_e	The exact result set
S_a	An approximation result set
S_k	The k-th feasible set of q
$\mathcal{C}(q, S)$	The cost value for S in q
$\mathcal{P}(S)$	The maximum distance between two objects in S
$C_o^{\mathcal{P}(S)}$	The corresponding circle of o for a set S
γ_e	The value equals to $\mathcal{C}(q, S_e)$
γ_k	The value equals to $\mathcal{C}(q, S_k)$
$C_q^{\gamma_e}$	The circle centered at q with a radius of γ_e
$C_q^{\gamma_k}$	The circle centered at q with a radius of γ_k
RSR(S)	The relaxed safe region of a set S
o_f	The furthest object from q' in S_a
$N_t(o)$	The t-Voronoi neighbor set of o

ply contain the query keywords and return the results in a continuous manner.

These studies retrieve single objects that are close to the query object and are relevant to the query keywords when the query moves. In contrast, we retrieve a set of objects that are close to the query object and collectively meet the keywords requirement. Hence, the above methods cannot be directly applied to our MCSKQ problem.

3 Preliminaries

We define the moving collective spatial keyword query. We summarize the frequently used symbols in Table 1.

Let $\mathcal{O}$ be a set of two-dimensional static spatio-textual objects. Each object $o \in \mathcal{O}$ has a location $o.\lambda$ and a set of keywords $o.\psi$. We denote by $d(o_1, o_2)$ the Euclidean distance between two objects o_1 and o_2.

A query $q = \langle q.\lambda, q.\psi \rangle$ also has a location $q.\lambda$ and a set of keywords $q.\psi$. Given a subset S of objects in $\mathcal{O}$, if the union of all keywords in S *covers* $q.\psi$, i.e., $q.\psi \subseteq \bigcup_{o \in S} o.\psi$, we call S a *feasible set*. The collective spatial keyword query (CSKQ) is to find a feasible set S having the minimum cost measured by a given cost function $\mathcal{C}(q, S)$. In this paper, we mainly focus on the $MaxMax$ cost function $\mathcal{C}_{max^2}(q, S)$ [5], which computes a weighted sum of the maximum distances (i) between q and objects in S and (ii) between any two objects in S, i.e.,

$$\mathcal{C}_{max^2}(q, S) = \alpha \cdot \max_{o \in S} d(q, o) + (1 - \alpha) \cdot \max_{o_1, o_2 \in S} d(o_1, o_2), \quad (1)$$

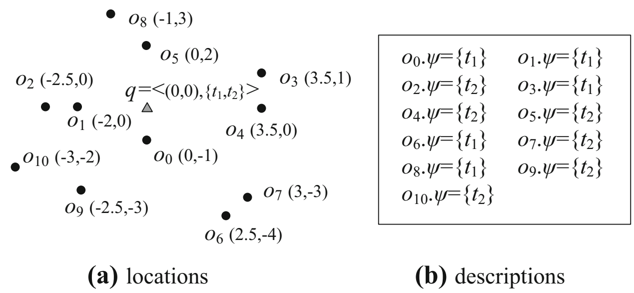


Fig. 2 An example of MCSKQ

where $\alpha \in [0, 1]$ balances the relative importance of the two distances. The CSKQ problem using the MaxMax cost function is called *MaxMax-CSKQ*. For ease of presentation, we will simply use $\mathcal{C}(\cdot)$ to denote $\mathcal{C}_{max^2}(\cdot)$, CSKQ to denote MaxMax-CSKQ, and omit the parameter α for the rest of this paper. The MaxMax cost function thus can be simply denoted as:

$$\mathcal{C}(q, S) = \max_{o \in S} d(q, o) + \max_{o_1, o_2 \in S} d(o_1, o_2). \quad (2)$$

Note that the algorithms introduced in this paper are valid for different α values.

We use an example in Fig. 2 to illustrate the above concepts. Figure 2a shows the locations of a set of objects $\mathcal{O} = \{o_0, o_1, \ldots, o_{10}\}$, and Fig. 2b shows the keywords of each object. For ease of illustration, each object is associated with one keyword, although our proposed algorithms can support multiple keywords. Given a CSKQ $q = \langle (0, 0), \{t_1, t_2\} \rangle$ as represented by the small gray triangle in Fig. 2a, the query result set is $\{o_1, o_2\}$, because the union of keywords from o_1 and o_2 is $\{t_1, t_2\}$ which covers the keyword set $q.\psi = \{t_1, t_2\}$ of the query, and o_1 and o_2 have close locations with $d(o_1, o_2) = 0.5$ and the maximum distance between q and o_1 or o_2 is only $d(q, o_2) = 2.5$. This gives the minimum cost $\mathcal{C}(q, \{o_1, o_2\}) = 2.5 + 0.5 = 3$ among any feasible sets of $\mathcal{O}$.

CSKQ assumes that the query location is static. When the query q moves, q is at different locations at different time. This essentially constitutes a *moving collective spatial keyword query*, i.e., moving CSKQ or MCSKQ. We formally define the MCSKQ problem as follows.

Definition 1 *[Moving Collective Spatial Keyword Query (MCSKQ)]* Given a set of static objects $\mathcal{O}$ and a moving query $q = \langle q.\lambda, q.\psi \rangle$. As the query is moving to a new location $q'.\lambda$, the MCSKQ continuously returns the result of the CSKQ $q' = \langle q'.\lambda, q.\psi \rangle$.

When an MCSKQ q is issued, we first process the query as if it were a static CSKQ. Since the static CSKQ problem is NP-hard [5], it will result in high computation costs

Table 2 Summary of algorithms

Problem	Static algorithm	Approximation ratio	Our maintenance algorithm	Section
MCSKQ	MaxMax-exact [12], MaxSum-exact [7]	N.A.	SRSR/MRSR$_{ad}$	4
(Euclidean space)	MaxMax-appro1 (L-Appro) [12]	3	AIM	5.1
	MaxMax-appro2 [12], MaxSum-appro [7]	1.8, 1.375	AAM	5.2
MCSKQ with	wMaxMax-exact [12]	N.A.	wMRSR$_{ad}$	6.1
weighted objects	wMaxMax-appro [12]	$1+2e$	wAIM	6.1
MCSKQ	Sliding window algorithm [14]	N.A.	rMRSR$_{ad}$	6.2
on road networks	NEB algorithm [14]	3	rAIM	6.2

if we recompute and process one CSKQ at every timestamp to return the query result continuously. Therefore, we propose efficient algorithms for MCSKQ, which achieve high query efficiency by maintaining the query result with auxiliary information and reducing the recomputation frequency. Specifically, when q moves to q', we first try to recompute the result using the auxiliary information. Only when this recomputation fails, we recompute a new CSKQ q' and update the auxiliary information.

In the following sections, we present the proposed algorithms for MCSKQ, which can use either the exact algorithms, or the L-Appro algorithm, or the H-Appro algorithms (cf. Sect. 2.1) in recomputation. We also adapt the proposed algorithms to solve MCSKQ with weighted objects and MCSKQ on road networks, etc. Table 2 summarizes the proposed algorithms.

4 MCSKQ algorithms for exact result maintenance

In this section, we discuss how to use the safe region-based techniques to maintain the exact result set. In particular, we first introduce the concept of *relaxed safe region* in Sect. 4.1, based on which two algorithms *SRSR* and $MRSR_{ad}$ are proposed in Sect. 4.2. SRSR uses a single relaxed safe region to maintain the exact result set S_e, while MRSR$_{ad}$ computes multiple relaxed safe regions and uses them to maintain S_e collectively.

4.1 Relaxed safe region

A straightforward method to compute a safe region for the exact result set S_e is that: For any feasible set S_i (except S_e), we compute a dominant region of S_e for S_i, within which when the query moves the cost of S_e is still not larger than that of S_i. Thus, the intersection of all the dominant regions is a safe region of S_e. However, this method will bring in two overheads: *construction overhead* and *validation overhead*. Construction overhead: Let us assume that both S_e and S_i contain two data objects, i.e., $S_e = \{o_{e1}, o_{e2}\}$ and $S_i = \{o_{i1}, o_{i2}\}$. The dominant region $D(S_e, S_i) = \{q' | \max_{o\in S_e} d(q', o) + \max_{o_1,o_2\in S_e} d(o_1, o_2) \leq \max_{o\in S_i} d(q', o) + \max_{o_1,o_2\in S_i} d(o_1, o_2)\}$. Given two feasible sets S_m and S_n, and two objects $o_i \in S_m$ and $o_j \in S_n$, we define the dominant region of o_i to o_j, denoted by $D(o_i, o_j)$, as $D(o_i, o_j) = \{q' | d(q', o_i) + \max_{o_1,o_2\in S_m} d(o_1, o_2) \leq d(q', o_j) + \max_{o_1,o_2\in S_n} d(o_1, o_2)\}$. Then, the dominant region $D(S_e, S_i)$ can be derived as:

$$D(S_e, S_i) = (D(o_{e1}, o_{i1}) \cap D(o_{e2}, o_{i1})) \cup (D(o_{e1}, o_{i2}) \cap D(o_{e2}, o_{i2})).$$

It will be costly to compute $D(S_e, S_i)$ because of the intersection and union operations included in above function, and even more costly to intersect all the dominant regions to compute the safe region; Validation overhead: the safe region that we obtain adopting this method will have an irregular shape result from intersection and union operations. The irregular shape adds extra costs to check whether the query object is still in this region.

To overcome these limitations, in this section, we first discuss how to compute a *local* safe region of the exact result set S_e only using the objects in S_e. The underlying idea is to transform the maintenance of S_e into maintaining each object in S_e separately. As we will see, the safe region forms a line segment. Since the query can move along any possible directions, it is very easy for the query to exit this safe region and thus invoke recomputation frequently. Therefore, we further propose to use the top-k feasible sets to extend the local safe region, which we call *relaxed safe region*. When the query q moves to q', as long as q' is in the relaxed safe region, the exact result set of q' remains to be one of the top-k feasible sets of q.

Local safe region Given a feasible set S of query q, let $\mathcal{P}(S)$ denote the maximum distance between any two objects in S, which does not change when the query moves. The cost function (2) can then be rewritten as:

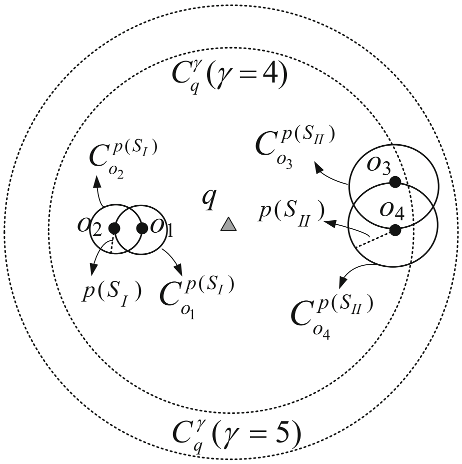


Fig. 3 An example for Lemma 1

$$\mathcal{C}(q, S) = \max_{o \in S}\{d(q, o) + \mathcal{P}(S)\}. \tag{3}$$

We can see that if we treat each object $o \in S$ as a circle $C_o^{\mathcal{P}(S)}$ that centers at o and has a radius of length $\mathcal{P}(S)$, then $d(q, o) + \mathcal{P}(S)$ is the furthest distance from q to circle $C_o^{\mathcal{P}(S)}$. Thus, the cost of S is the *maximum* furthest distance, i.e., $\max\{d(q, o) + \mathcal{P}(S) | \ o \in S\}$. Based on this property, we propose the following three lemmas to compute a safe region for S_e, which we call the local safe region.

Lemma 1 *Given a feasible set S of query q and let C_q^{γ} be the circle centered at q with a radius of γ. We have $\mathcal{C}(q, S) \le \gamma$ if and only if the circle $C_o^{\mathcal{P}(S)}$ for each $o \in S$ is inside C_q^{γ}.*

Proof According to Eq. (3), we know that $\mathcal{C}(q, S) \le \gamma$ means $\max_{o \in S}\{d(q, o) + \mathcal{P}(S)\} \le \gamma$, i.e., $\forall o \in S, d(q, o) \le \gamma - \mathcal{P}(S)$. Therefore, the circle $C_o^{\mathcal{P}(S)}$ for each $o \in S$ is inside C_q^{γ}, and vice versa. □

For example, in Fig. 3, using the example illustrated in Fig. 2, for the query $q = \langle(0, 0), \{t_1, t_2\}\rangle$, we have two feasible sets $S_{\text{I}} = \{o_1, o_2\}$ and $S_{\text{II}} = \{o_3, o_4\}$. The cost $\mathcal{C}(q, S_{\text{I}})$ of S_{I} is 3 with $\mathcal{P}(S_{\text{I}}) = d(o_1, o_2) = 0.5$. The cost $\mathcal{C}(q, S_{\text{II}})$ of S_{II} is 4.64 with $\mathcal{P}(S_{\text{II}}) = d(o_3, o_4) = 1$. For the four circles $C_{o_1}^{\mathcal{P}(S_{\text{I}})}$, $C_{o_2}^{\mathcal{P}(S_{\text{I}})}$, $C_{o_3}^{\mathcal{P}(S_{\text{II}})}$, and $C_{o_4}^{\mathcal{P}(S_{\text{II}})}$, given a circle C_q^{γ}, when $\gamma = 5$, we have $\mathcal{C}(q, S_{\text{I}}) < \mathcal{C}(q, S_{\text{II}}) < \gamma$. From Fig. 3 we see that the four circles are all inside C_q^{γ}. When $\gamma = 4$, we have $\mathcal{C}(q, S_{\text{I}}) < \gamma < \mathcal{C}(q, S_{\text{II}})$, and only $C_{o_1}^{\mathcal{P}(S_{\text{I}})}$ and $C_{o_2}^{\mathcal{P}(S_{\text{I}})}$ are inside C_q^{γ}.

For ease of presentation, in the rest of this paper, we will use the dataset $\mathcal{O}$ and query $q = \langle(0, 0), \{t_1, t_2\}\rangle$ presented in Fig. 2 as a running example. We denote the set $\{C_o^{\mathcal{P}(S)} | o \in S\}$ of circles as $Circles(S)$. Also, we say a feasible set S is *inside* the circle C_q^{γ} if all the circles in $Circles(S)$ are inside C_q^{γ}.

Here, we suppose that for any feasible set $S \neq S_e$, $\mathcal{C}(q, S) > \mathcal{C}(q, S_e)$. Let $C_q^{\gamma_e}$ be the circle centered at q with a radius of $\gamma_e = \mathcal{C}(q, S_e)$. The following Lemma 2 shows that the exact result S_e is the only feasible set inside $C_q^{\gamma_e}$.

Lemma 2 *S_e is the only feasible set inside $C_q^{\gamma_e}$.*

Proof Since $\mathcal{C}(q, S_e) \le \gamma_e$, by Lemma 1 we have that S_e is inside $C_q^{\gamma_e}$. For any feasible set $S \neq S_e$, since $\mathcal{C}(q, S) > \mathcal{C}(q, S_e) = \gamma_e$, by Lemma 1, there is at least one object $o \in S$ whose corresponding circle $C_o^{\mathcal{P}(S)}$ is not inside $C_q^{\gamma_e}$. Therefore, S_e is the only feasible set inside $C_q^{\gamma_e}$. □

Suppose that the query q moves to q' within the circle $C_q^{\gamma_e}$ where $\gamma_e = \mathcal{C}(q, S_e)$. Let $C_{q'}^{\gamma_e - d(q,q')}$ be the circle centered at q' with a radius of $\gamma_e - d(q, q')$. When the current S_e is inside $C_{q'}^{\gamma_e - d(q,q')}$, the following Lemma 3 shows that the current S_e is still the answer to the query q'.

Lemma 3 *When S_e is inside $C_{q'}^{\gamma_e - d(q,q')}$, S_e is still the answer to q'.*

Proof (1) When S_e is inside $C_{q'}^{\gamma_e - d(q,q')}$, by Lemma 1 we have $\mathcal{C}(q', S_e) \le \gamma_e - d(q, q')$;

(2) Since the distance $d(q, q')$ is not larger than the radius difference between $C_q^{\gamma_e}$ and $C_{q'}^{\gamma_e - d(q,q')}$, i.e., $d(q, q') \le \gamma_e - (\gamma_e - d(q, q'))$, we have $C_{q'}^{\gamma_e - d(q,q')}$ is inside $C_q^{\gamma_e}$. By Lemma 2, the exact result set S_e is the only feasible set inside $C_q^{\gamma_e}$. Thus, when S_e is also inside $C_{q'}^{\gamma_e - d(q,q')}$, S_e is the only feasible set inside $C_{q'}^{\gamma_e - d(q,q')}$. For any feasible set $S \neq S_e$, since S is not inside $C_{q'}^{\gamma_e - d(q,q')}$, by Lemma 1 we have

$$\mathcal{C}(q', S) > \gamma_e - d(q, q');$$

Combining (1) and (2), we have $\mathcal{C}(q', S) > \mathcal{C}(q', S_e)$ for any feasible set $S \neq S_e$, which completes the pf. □

For example, in Fig. 4, the exact result set $S_e = \{o_1, o_2\}$ which is the answer to the query q. When q moves to q', since $C_{o_1}^{\mathcal{P}(S_e)}$ and $C_{o_2}^{\mathcal{P}(S_e)}$ are both inside $C_{q'}^{\gamma_e - d(q,q')}$, $S_e = \{o_1, o_2\}$ is still the answer to q'.

One important usage of Lemma 3 is that we can use it to construct a safe region of S_e. For any object $o \in S_e$, the corresponding circle $C_o^{\mathcal{P}(S_e)}$ is inside $C_{q'}^{\gamma_e - d(q,q')}$ if and only if:

$$d(q', o) \le \gamma_e - d(q, q') - \mathcal{P}(S_e),$$

which can be written as:

$$d(q', o) + d(q, q') \le \gamma_e - \mathcal{P}(S_e).$$

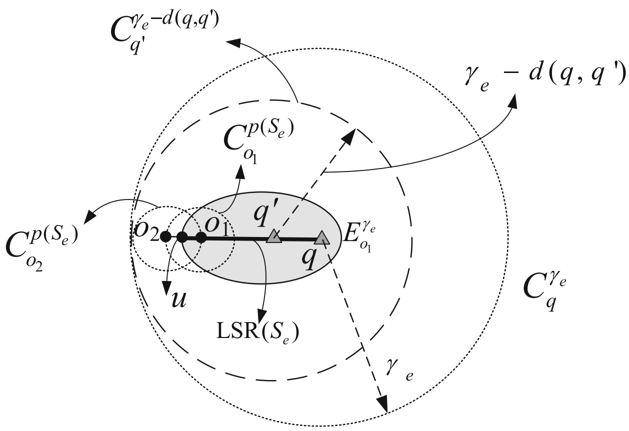


Fig. 4 A local safe region of S_e

Therefore, the possible locations of q' form an ellipse with q and o being its two foci and $\gamma_e - \mathcal{P}(S_e)$ its major axis length. We denote this ellipse by

$$E_o^{\gamma_e} = \{q' | d(q', o) + d(q, q') \leq \gamma_e - \mathcal{P}(S_e)\}. \quad (4)$$

The intersection of all the ellipses $E_o^{\gamma_e}$ for $o \in S_e$ forms a safe region of S_e, within which when q moves S_e remains the same. We call this a *local safe region* (LSR), i.e.,

$$LSR(S_e) = \bigcap_{o \in S_e} E_o^{\gamma_e}. \quad (5)$$

Suppose that there is only one object $o \in S_e$ having $d(q, o) = \max_{o' \in S_e} d(q, o')$. We show with Theorem 1 that LSR(S_e) is a line segment, which means that the exact result set S_e is valid only when the query object moves in a particular direction.

Theorem 1 *LSR(S_e) is a line segment*

Proof Recall that $\gamma_e = \mathcal{C}(q, S_e)$. By Eq. (2), we have $\gamma_e = \max_{o \in S_e}\{d(q, o) + \mathcal{P}(S_e)\}$. Let o_m be an object in S_e having $\gamma_e = d(q, o_m) + \mathcal{P}(S_e)$. By Eq. (4), the corresponding ellipse of o_m is

$$E_{o_m}^{\gamma_e} = \{q' | d(q', o_m) + d(q, q') \leq \gamma_e - \mathcal{P}(S_e)\}.$$

Using $d(q, o_m)$ to replace $\gamma_e - \mathcal{P}(S_e)$, we have

$$E_{o_m}^{\gamma_e} = \{q' | d(q', o_m) + d(q, q') \leq d(q, o_m)\}.$$

By the triangle inequality, we have

$$d(q', o_m) + d(q, q') \geq d(q, o_m).$$

Thus, we have $E_{o_m}^{\gamma_e} = \{q' | d(q', o_m) + d(q, q') = d(q, o_m)\}$, which is the line segment $\overline{qo_m}$. For each object $o \in S_e - \{o_m\}$, since $\gamma_e > d(q, o) + \mathcal{P}(S_e)$, $E_o^{\gamma_e}$ is an ellipse containing the line segment $\overline{qo}$. Because LSR(S_e) is the intersection of $E_o^{\gamma_e}$ for each object $o \in S_e$, LSR(S_e) is a line segment. □

For example, in Fig. 4 (recall that the exact result set $S_e = \{o_1, o_2\}$), $E_{o_1}^{\gamma_e}$ is the gray ellipse with q and o_1 being its two foci and $\gamma_e - \mathcal{P}(S_e)$ the major axis length, and $E_{o_2}^{\gamma_e} = \{q' | d(q', o_2) + d(q, q') = d(q, o_2)\}$, which is the line segment $\overline{qo_2}$. Point u is the boundary point of the intersection of $E_{o_1}^{\gamma_e}$ and $\overline{qo_2}$, and LSR(S_e) is the thick line segment $\overline{qu}$, because LSR(S_e)=$E_{o_1}^{\gamma_e} \bigcap \overline{qo_2} = \overline{qu}$. Since the query q can move in any direction, LSR(S_e) is quite restrictive and q can easily move out of this safe region. This motivates us to find a larger safe region to more effectively reduce the recomputation frequency.

Relaxed safe region By Eq. (4), if γ_e increases, the ellipses $E_o^{\gamma_e}$ for each $o \in S_e$ will become larger and so will their intersection. Thus, we increase γ_e to expend the safe region.

Let $S_1, S_2, \ldots, S_k$ be the top-k feasible sets of a query q. Since S_e is the exact result set, we have $S_1 = S_e$. Hereafter, we will use S_1 and S_e interchangeably. Let $C_q^{\gamma_k}$ be a circle centered at q with a radius of $\gamma_k = \mathcal{C}(q, S_k)$. Since the costs of the other (non-top-k) feasible sets are larger than γ_k, by Lemma 1, only the top-k feasible sets $S_1, S_2, \ldots, S_k$ are inside $C_q^{\gamma_k}$.

For an object $o \in S_e$, we use γ_k to replace γ_e in Eq. (4) and have the ellipse $E_o^{\gamma_k}$, i.e.,

$$E_o^{\gamma_k} = \{q' | d(q', o) + d(q, q') \leq \gamma_k - \mathcal{P}(S_e)\}. \quad (6)$$

We call the intersection of all the ellipses $E_o^{\gamma_k}$ for each $o \in S_e$ the *relaxed safe region* (RSR) of S_e, i.e.,

$$RSR(S_e) = \bigcap_{o \in S_e} E_o^{\gamma_k}. \quad (7)$$

For example, in Table 3, $S_1 = \{o_1, o_2\}$, $S_2 = \{o_5, o_8\}$, and $S_3 = \{o_3, o_4\}$ are the top-3 feasible sets of query q. For the objects o_1, o_2, o_3, o_4, o_5, and o_8, their corresponding circles are shown in Fig. 5, which are inside the circle $C_q^{\gamma_3}$ centered at q with a radius of $\gamma_3 = \mathcal{C}(q, S_3)$. The gray region is RSR(S_e), which is the intersection of the two ellipses $E_{o_1}^{\gamma_3}$ and $E_{o_2}^{\gamma_3}$.

Using the following Theorem 2, we show that when q moves to q', as long as q' is in RSR(S_e), the current S_e is still better than any non-top-k feasible sets and thus the answer to q' is the one having the minimum cost among the top-k feasible sets of q.

Theorem 2 *Given the top-k feasible sets $S_1, S_2, \ldots, S_k$ of q, when q moves to q' within $RSR(S_e)$, for any non-top-k feasible set S_j for $j > k$: $\mathcal{C}(q', S_j) > \mathcal{C}(q', S_e)$.*

Proof (1) When q' is in RSR(S_e), by Eqs. (6) and (7), for each object $o \in S_e$, we have $d(q', o) + d(q, q') \leq \gamma_k - \mathcal{P}(S_e)$,

Table 3 The top-3 feasible sets of q

Feasible set	Cost value	Maximum distance
$S_1 = \{o_1, o_2\}$	$\mathcal{C}(q, S_1) = 3$	$\mathcal{P}(S_1) = 0.5$
$S_2 = \{o_5, o_8\}$	$\mathcal{C}(q, S_2) = 4.57$	$\mathcal{P}(S_2) = \sqrt{2}$
$S_3 = \{o_3, o_4\}$	$\mathcal{C}(q, S_3) = 4.64$	$\mathcal{P}(S_3) = 1$

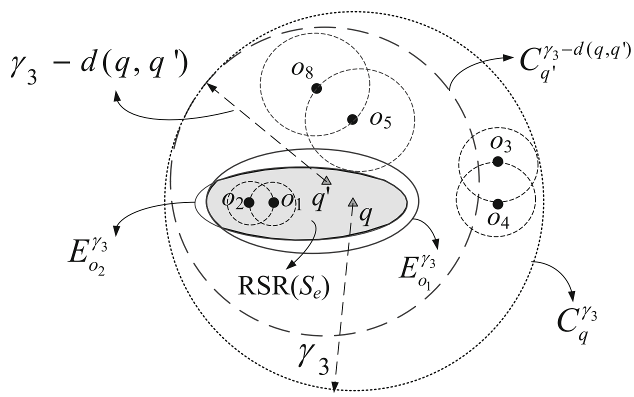


Fig. 5 Relaxed safe region of S_e

which can be written as

$$d(q', o) + \mathcal{P}(S_e) \le \gamma_k - d(q, q').$$

By definition, we have

$$\mathcal{C}(q', S_e) \le \gamma_k - d(q, q').$$

(2) It is easy to see that the circle $C_{q'}^{\gamma_k - d(q,q')}$ is inside $C_q^{\gamma_k}$ as $d(q, q') \le \gamma_k - (\gamma_k - d(q, q'))$. Since only the top-$k$ feasible sets are inside the circle $C_q^{\gamma_k}$, and $C_{q'}^{\gamma_k - d(q,q')}$ also locates inside $C_q^{\gamma_k}$, any non-top-k feasible set S_j for $j > k$, S_j cannot be inside the circle $C_{q'}^{\gamma_k - d(q,q')}$, i.e.,

$$\mathcal{C}(q', S_j) > \gamma_k - d(q, q').$$

Combining (1) and (2), we have $\mathcal{C}(q', S_j) > \mathcal{C}(q', S_e)$ for any non-top-k feasible set S_j for $j > k$. □

4.2 MCSKQ algorithms based on relaxed safe region

In this section, we present two algorithms for MCSKQ, which use relaxed safe region to continuously return the exact result set.

4.2.1 Single relaxed safe region algorithm

First, we discuss the algorithm that continuously returns the exact result set S_e using a single relaxed safe region RSR(S_e).

Query updates Suppose that we have computed the top-k feasible sets of q, and use a prioritized queue Q to store them by ascending order of their costs. When q moves to q', we check whether q' is in RSR(S_e). If not, then we recompute the top-k feasible sets and return the new S_e. If so, we compute the new S_e and update Q as follows: For any feasible set S_i in Q, if S_i is inside the circle $C_{q'}^{\gamma_k - d(q,q')}$ (recall that $\gamma_k = \mathcal{C}(q, S_k)$), i.e., $\mathcal{C}(q', S_i) \le \gamma_k - d(q, q')$ (cf. Lemma 1), then we put S_i into a new queue Q' prioritized by the updated cost value. The queue Q' becomes the new Q and the first entry S'_1 is the new S_e. The correctness of this algorithm is discussed using the following theorem.

Theorem 3 *Let $S_{|Q'|}$ be the last entry in queue Q', i.e., $\mathcal{C}(q', S_{|Q'|})$ is the maximal cost among all the entries in Q'. Then $\forall S \notin Q'$ that is a feasible set: $\mathcal{C}(q', S) > \mathcal{C}(q', S_{|Q'|})$.*

Proof (1) From the update process, we know that the cost of any feasible set in Q' is not larger than $\gamma_k - d(q, q')$, i.e., $\mathcal{C}(q', S_{|Q'|}) \le \gamma_k - d(q, q')$;

(2) For any feasible set $S_i \in Q \backslash Q'$, since its cost is larger than $\gamma_k - d(q, q')$, we have $\mathcal{C}(q', S_i) > \gamma_k - d(q, q')$;

(3) For any feasible set $S_j \notin Q$, it holds that $\mathcal{C}(q', S_j) > \gamma_k - d(q, q')$ (cf. step 2 of the pf of Theorem 2);

Combining (1), (2), and (3), we have $\forall S \notin Q'$ that is a feasible set: $\mathcal{C}(q', S) > \mathcal{C}(q', S_{|Q'|})$. □

For example, we store the top-3 feasible sets of q, $S_1 = \{o_1, o_2\}$, $S_2 = \{o_5, o_8\}$, and $S_3 = \{o_3, o_4\}$ in queue Q. In Fig. 5, q moves to q' which is still in RSR(S_e). By Theorem 2, the answer to q' can be found in Q. In the update process, since S_1 and S_2 are inside the circle $C_{q'}^{\gamma_3 - d(q,q')}$, we insert S_1 and S_2 into Q' and return the first entry in Q' as the answer to q'.

The SRSR Algorithm Algorithm 1 summarizes the query processing procedure. This algorithm uses a single relaxed safe region RSR(S_e) to maintain S_e, so we call it the *single relaxed safe region* (SRSR) algorithm. When a query comes, we first compute the top-k feasible sets $S_1, S_2, \ldots, S_k$ and keep them in Q. The current S_e is the first entry of Q. We compute RSR(S_e) and return S_e as the answer (Lines 1–3). Then, we start query maintenance (Lines 4–15). When q moves to q', we check whether q' is in RSR(S_e). If not then Q becomes invalid and we recompute the top-k feasible sets (Lines 6–7). If q' is still in RSR(S_e), then Q is valid, i.e., the new S_e is still in Q. We update Q with queue Q' to obtain the new S_e (Lines 9–12). When this is done, we replace Q and q by Q' and q', respectively. We recompute the relaxed safe region of the new S_e and return the new S_e as the answer (Lines 14–15).

Algorithm 1: SRSR

Input : an MCSKQ q
Output: the new S_e

$Q \leftarrow topKSetsSearch()$
$ComputeRSR()$
$reportResult(S_e)$
while *query continues* **do**
  q *moves to* q'
  if $q'.\lambda$ *is not in* $RSR(S_e)$ **then**
    $Q \leftarrow topKSetsSearch()$
  else
    while *not* $Q.empty()$ **do**
      $S_i \leftarrow Q.dequeue()$
      if $\mathcal{C}(q', S_i) \leq \gamma_k - d(q, q')$ **then**
        $Q'.enqueue(S_i, \mathcal{C}(q', S_i))$
    $q \leftarrow q'$, $Q \leftarrow Q'$, $Q'.clear()$
  $ComputeRSR()$
  $reportResult(S_e)$

procedure $ComputeRSR()$
$S_e \leftarrow$ *first entry of* Q
$k \leftarrow |Q|, \gamma_k \leftarrow \mathcal{C}(q, S_{|Q|})$
for *each object* $o \in S_e$ **do**
  $RSR(S_e) \leftarrow RSR(S_e) \bigcap E_o^{\gamma_k}$

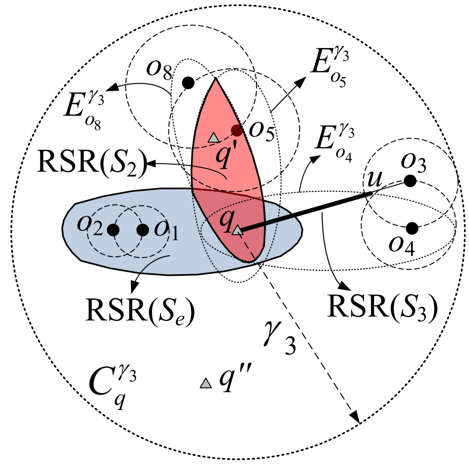


Fig. 6 Multiple relaxed safe regions

4.2.2 Multiple relaxed safe region algorithm

In order to further reduce the recomputation frequency, in this section, we introduce another algorithm that computes k RSRs from the top-k feasible sets and uses these k RSRs to collectively maintain the exact result set S_e. As long as the query moves in one of these RSRs, the new S_e remains to be one of the top-k feasible sets.

Since there are multiple RSRs need to be maintained, we propose two optimizations. One is to change the access order of the k RSRs to early terminate the checking process, and the other is to delete some of the top-k feasible sets that cannot be the new S_e in the subsequent query maintenance.

Compute more RSRs Similar to obtain the relaxed safe region of the exact result set RSR(S_e), we can compute the relaxed safe regions of the other top-k feasible sets RSR(S_i) for $i \in [2, k]$ with the function

$$RSR(S_i) = \bigcap_{o \in S_i} E_o^{\gamma_k}, \tag{8}$$

where $E_o^{\gamma_k} = \{q' | d(q', o) + d(q, q') \leq \gamma_k - \mathcal{P}(S_i)\}$ is an ellipse and $\gamma_k = \mathcal{C}(q, S_k)$.

For example, in Fig. 6, RSR(S_2) is the red region which is the intersection of the two ellipses $E_{o_5}^{\gamma_3}$ and $E_{o_8}^{\gamma_3}$, and RSR(S_3) is the thick line segment $\overline{qu}$ which is the intersection of $E_{o_4}^{\gamma_3}$ and the line segment $\overline{qo_3}$ (cf. Theorem 1).

Using the k RSRs, we can further reduce the recomputation frequency, which is based on the following Theorem 4.

Theorem 4 *Given the top-k feasible sets $S_1, S_2, \ldots, S_k$ of q, when q moves to q' within any $RSR(S_i)$ for $i \in [1, k]$, for any non-top-k feasible set S_j for $j > k$: $\mathcal{C}(q', S_j) >$* $\min\{\mathcal{C}(q', S_i), i \in [1, k]\}$.

Proof Suppose q' is in RSR(S_m) ($1 \leq m \leq k$), we can use similar steps as the pf of Theorem 2 to show that for any non-top-k feasible set S_j for $j > k$, we have that $\mathcal{C}(q', S_j) > \mathcal{C}(q', S_m)$. Thus, it holds that $\mathcal{C}(q', S_j) > \min\{\mathcal{C}(q', S_i), i \in [1, k]\}$. □

By Theorem 4, when q moves to q', as long as q' is in any RSR(S_i) for $i \in [1, k]$, the answer to q' is the one having the minimum cost among the top-k feasible sets of q. For example, in Fig. 6, when q moves to q' within RSR(S_2), by Theorem 4, the query answer of q' is the best one among S_1, S_2, and S_3.

For the query update process, we also use a queue Q to store the top-k feasible sets of q. When q moves to q', for each feasible set S_i in Q, we check whether q' is in the corresponding relaxed safe region RSR(S_i). If so, S_i is still the top-k feasible set of q' and we insert it into a new queue Q'. After that, if Q' is not empty, then the top entry S_1' in Q' is the new S_e and Q' is used in the subsequent query maintenance. In order to improve the update efficiency, next, we show two optimizations.

Access order optimization

Pruning rule Let the top-k feasible sets S_i for $i \in [1, k]$ be stored in a queue Q prioritized by $\gamma_k - \mathcal{P}(S_i)$, and we retrieve each feasible set in descending order of $\gamma_k - \mathcal{P}(S_i)$. Once we encounter an S_j having $d(q, q') > \gamma_k - \mathcal{P}(S_j)$, we can stop and ignore the remaining feasible sets.

The correctness of this pruning rule is straightforward. If $d(q, q') > \gamma_k - \mathcal{P}(S_j)$, for each object $o \in S_j$, we have

$$d(q', o) + d(q, q') > \gamma_k - \mathcal{P}(S_j).$$

Since RSR(S_j) is the intersection of all the ellipses $E_o^{\gamma_k}$ for each $o \in S_j$ where $E_o^{\gamma_k} = \{q' | d(q', o) + d(q, q') \leq \gamma_k - \mathcal{P}(S_j)\}$, q' is not in RSR(S_j).

For ease of presentation, we use $\nu_{k,i}$ to denote $\gamma_k - \mathcal{P}(S_i)$ in the rest of paper. Continue with the above example in Fig. 6 (recall that $\gamma_3 = \mathcal{C}(q, S_3)$), if we store the top-3 feasible sets in a max-heap Q prioritized by $\nu_{3,i}$ (i.e., $\gamma_3 - \mathcal{P}(S_i)$), the access order of the feasible sets will be S_1, S_3, and S_2. Suppose q moves to q'', in the query update process, when we check S_3, since $d(q, q'') = 4 > \nu_{3,3} = 3.64$, based on the pruning rule, there is no need to check whether q'' is in RSR(S_3) and RSR(S_2) any more.

Dominance-based optimization For any two feasible sets S_i and S_j, if $\mathcal{C}(q, S_j) \geq \mathcal{C}(q, S_i)$ always holds when q moves in the whole space, we say S_i *dominates* S_j, denoted by $S_i \prec S_j$. Thus, if a client–server-based system is used, before sending the top-k feasible sets to the client, we can delete the feasible sets dominated by others, which will reduce communication cost as well as improve the efficiency of the query update process. In the following, we introduce a sufficient condition for $S_i \prec S_j$: If there exists an object $o_m \in S_j$ with $d(o_m, o) \leq \mathcal{P}(S_j) - \mathcal{P}(S_i)$ for $\forall o \in S_i$, then $S_i \prec S_j$.

The sufficient condition holds because by the triangle inequality, for $o \in S_i$, we have

$$d(q, o) - d(q, o_m) \leq d(o_m, o).$$

Since $d(o_m, o) \leq \mathcal{P}(S_j) - \mathcal{P}(S_i)$, it holds that

$$d(q, o) + \mathcal{P}(S_i) \leq d(q, o_m) + \mathcal{P}(S_j).$$

By Eq. (3), i.e., $\mathcal{C}(q, S) = \max_{o \in S}\{d(q, o) + \mathcal{P}(S)\}$, we have

$$\mathcal{C}(q, S_i) \leq d(q, o_m) + \mathcal{P}(S_j) \leq \mathcal{C}(q, S_j).$$

For example, given the query q in Fig. 2, there are two feasible sets $S_{\mathrm{I}} = \{o_1, o_2\}$ ($\mathcal{P}(S_{\mathrm{I}}) = 0.5$) and $S_{\mathrm{II}} = \{o_0, o_2\}$ ($\mathcal{P}(S_{\mathrm{II}}) = 2.69$). We have $S_{\mathrm{I}} \prec S_{\mathrm{II}}$ because there exists an object $o_2 \in S_{\mathrm{II}}$ that for all objects o_1, o_2 in S_{I} we have $d(o_2, o_1) = 0.5 < \mathcal{P}(S_{\mathrm{II}}) - \mathcal{P}(S_{\mathrm{I}}) = 2.69 - 0.5 = 2.19$ and $d(o_2, o_2) = 0 < \mathcal{P}(S_{\mathrm{II}}) - \mathcal{P}(S_{\mathrm{I}}) = 2.19$.

The MRSR$_{ad}$ algorithm Algorithm 2 summarizes the above query processing procedure. This algorithm uses multiple relaxed safe regions to maintain S_e and uses the proposed two optimizations to accelerate the query update process. We call it $MRSR_{ad}$. The function $dominance(\cdot)$ implements the dominance-based optimization. The access order optimization is applied in Lines 9–10. Compared with SRSR, this algorithm spends more time maintaining S_e.

Complexity Let $\mathcal{O}_\psi \subseteq \mathcal{O}$ be a set of objects. For each object $o \in \mathcal{O}_\psi$, $o.\psi \bigcap q.\psi \neq \emptyset$. The main costs of SRSR and MRSR$_{ad}$ lie in $topKSetsSearch(\cdot)$, which is the function that computes the top-k feasible sets and it takes $O(k \log k \cdot |\mathcal{O}_\psi|^{|q.\psi|})$ time. Assuming that the query object q moves at a constant speed v, for SRSR, if q exits RSR(S_e), the function $topKSetsSearch(\cdot)$ will be invoked. We denote the point of exit by q_e. The minimum distance between q and q_e is

Algorithm 2: MRSR$_{ad}$

Input : an MCSKQ q
Output: the new S_e

1 $Q \leftarrow topKSetsSearch()$
2 $dominance(Q)$
3 $ComputeRSRs()$
4 $reportResult(S_e)$
5 **while** *query continues* **do**
6 $\quad$ q *moves to* q'
7 $\quad$ **while** *not* $Q.empty()$ **do**
8 $\quad\quad$ $S_i \leftarrow Q.dequeue()$
9 $\quad\quad$ **if** $d(q, q') > \nu_{k,i}$ **then**
10 $\quad\quad\quad$ $break$
11 $\quad\quad$ **else**
12 $\quad\quad\quad$ **if** $q'.\lambda$ *is in* $RSR(S_i)$ **then**
13 $\quad\quad\quad\quad$ $Q'.enqueue(S_i, \mathcal{C}(q', S_i))$
14 $\quad$ $q \leftarrow q'$, $Q \leftarrow Q'$, $Q'.clear()$
15 $\quad$ **if** $Q.empty()$ **then**
16 $\quad\quad$ $Q \leftarrow topKSetsSearch()$
17 $\quad\quad$ $dominance(Q)$
18 $\quad$ $ComputeRSRs()$
19 $\quad$ $reportResult(S_e)$
20
21 **procedure** $ComputeRSRs()$
22 $S_e \leftarrow first\ entry\ of\ Q$
23 $k \leftarrow |Q|$, $\gamma_k \leftarrow \mathcal{C}(q, S_{|Q|})$
24 **while** *not* $Q.empty()$ **do**
25 $\quad$ $S_j \leftarrow Q.dequeue()$
26 $\quad$ **for** *each object* $o \in S_j$ **do**
27 $\quad\quad$ $RSR(S_j) \leftarrow RSR(S_j) \bigcap E_o^{\gamma_k}$
28 $\quad$ $\nu_{k,j} = \gamma_k - \mathcal{P}(S_j)$
29 $\quad$ $Q'.enqueue(S_j, \nu_{k,j})$
30 $Q \leftarrow Q'$, $Q'.clear()$

$$d_e = \frac{\mathcal{C}(q, S_k) - \mathcal{C}(q, S_e)}{2}.$$

In the worst-case, q moves in a straight line. Then, the frequency f of $topKSetsSearch(\cdot)$ invoked is

$$f = \frac{v}{d_e} = \frac{2v}{\mathcal{C}(q, S_k) - \mathcal{C}(q, S_e)}.$$

Thus, the time complexity of SRSR is $O(f \cdot k \log k \cdot |\mathcal{O}_\psi|^{|q.\psi|})$. If a client–server-based system is used, for each time $topKSetsSearch(\cdot)$ is invoked, k feasible sets are sent from the server to the client. As the size of a feasible set is at most $k \cdot |q.\psi|$, the communication cost for SRSR is $O(k \cdot |q.\psi| \cdot f)$. For MRSR$_{ad}$, only when q exits all RSR(S_i) for $i \in [1, k]$ is the function $topKSetsSearch(\cdot)$ invoked. Thus, we have

$$d_e = \max_{i \in [1,k]} \frac{\mathcal{C}(q, S_k) - \mathcal{C}(q, S_i)}{2},$$

and the frequency is

$$f = \max_{i \in [1,k]} \frac{2v}{\mathcal{C}(q, S_k) - \mathcal{C}(q, S_i)}.$$

5 MCSKQ algorithms for approximate result maintenance

As the cost analysis above shows, even with multiple RSRs and the two optimizations, the overall cost of MRSR$_{ad}$ is still high due to the recomputation of top-k feasible sets. In order to avoid the high cost in recomputation, in this section, we propose another two algorithms AIM and AAM, which both adopt the approximate algorithms described in Sect. 2.1 in recomputation. AIM uses the L-Appro algorithm to compute an approximate result set S_a and maintains S_a under a 3-approximation ratio. AAM is based on AIM and MRSR$_{ad}$, and maintains S_a under a 1.8- or 1.375-approximation ratio (depending on the underlying H-Appro algorithms in recomputation).

5.1 Approximate incremental maintenance

Given a query q, if we adopt the L-Appro algorithm in recomputation, we will have a 3-approximation result set S_a which consists of the nearest neighbors (NNs) of q for each keyword $t \in q.\psi$. When the query moves to q', if each object in S_a is still the NN of q', then the current S_a is the answer to q'. In order to check the validity of S_a efficiently, we propose the concept of *keyword-based Voronoi neighbor set* (KVNS). Based on KVNS, we propose the *approximate incremental maintenance* (AIM) algorithm, which adopts an incremental strategy to maintain S_a as a 3-approximation result.

Keyword-based Voronoi neighbor set Before formally defining KVNS, we first briefly discuss the Voronoi diagram. The *Voronoi diagram* [33] of a set $\mathcal{O}$ of n objects is a subdivision of the plane into n Voronoi cells, where each Voronoi cell $V(o_i)$ corresponds to an object o_i in $\mathcal{O}$ and for any point p within $V(o_i)$, o_i is the NN of p, i.e., $\forall o_j \in \mathcal{O} \backslash \{o_i\}$, $d(p, o_i) \le d(p, o_j)$. The Voronoi cells are also called order-1 Voronoi cells. For kNN ($k > 1$) queries, we have the corresponding higher-order Voronoi diagrams and Voronoi cells. For ease of presentation, in this section, Voronoi cells refer to order-1 Voronoi cells unless specified otherwise. The Voronoi diagram is computed using the bisectors between the objects in $\mathcal{O}$. Only those between adjacent data objects will be included in the Voronoi diagram.

Let $\mathcal{O}_t \subseteq \mathcal{O}$ contain the objects $o \in \mathcal{O}_t$ having keyword t. Given a Voronoi diagram of $\mathcal{O}_t$, for any two objects $o_i, o_j \in \mathcal{O}_t$, we call o_j a $t - Voronoi neighbor$ of o_i if their bisector is included in the Voronoi diagram, i.e., o_i and o_j share an edge of their Voronoi cells, denoted by $V(o_i)||V(o_j)$.

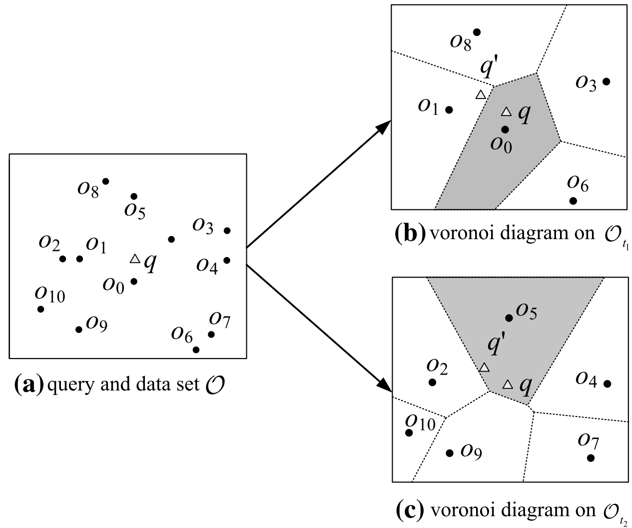


Fig. 7 The Voronoi diagram of $\mathcal{O}$

Definition 2 *[Keyword-based Voronoi Neighbor Set (KVNS)]* Given an object $o \in \mathcal{O}_t$, the set $\mathcal{O}'_t \subset \mathcal{O}_t$, denoted by $N_t(o)$, that contains all the t-Voronoi neighbors of o is the t-Voronoi neighbor set of o.

Conceptually, for an object o covering the keyword t, the t-Voronoi neighbor set of o defines a safe region, which is equivalent to a Voronoi cell for t.

We illustrate the above concepts using an example in Fig. 7. For the data set $\mathcal{O} = \{o_0, o_1, \ldots, o_{10}\}$ in Fig. 7a, let $\mathcal{O}_{t_1} = \{o_0, o_1, o_3, o_6, o_8\}$ be a subset of $\mathcal{O}$ where each object covers keyword t_1 and $\mathcal{O}_{t_2} = \{o_2, o_4, o_5, o_7, o_9, o_{10}\}$ where each object covers keyword t_2. In Fig. 7b, c, the dash lines indicate the boundaries of Voronoi cells of $\mathcal{O}_{t_1}$ and $\mathcal{O}_{t_2}$. The Voronoi cell $V(o_0)$ shares an edge with $V(o_1)$, $V(o_3)$, $V(o_6)$, and $V(o_8)$. Thus, the t_1-Voronoi neighbor set of o_0 is $N_{t_1}(o_0) = \{o_1, o_3, o_6, o_8\}$. Similarly, the t_2-Voronoi neighbor set of o_5 is $N_{t_2}(o_5) = \{o_2, o_4, o_9\}$.

For each object $o \in \mathcal{O}$, we precompute and store all the t-Voronoi neighbor sets for each keyword $t \in o.\psi$. Such precomputation is only needed once and can be computed efficiently [34]. As described later in the complexity analysis, the storage of KVNS only takes reasonable space.

Using KVNS, we can use only a few objects to check the validity of an existing NN of the moving query, which is formally stated in the following Lemma 4.

Lemma 4 *Given a query q, let o_t be the NN of q for keyword t. When q moves to q', if o_t is closer to q' than any object in $N_t(o_t)$, o_t is still the NN of q' for t.*

Proof By definition, $N_t(o_t)$ contains all the t-Voronoi neighbors of o_t. Since the segments of the bisectors of o_t and each object in $N_t(o_t)$ construct the (order-1) Voronoi cell $V(o_t)$ of o_t on $\mathcal{O}_t$, if o_t is closer to q' than any object in $N_t(o_t)$, q' is in $V(o_t)$. Thus, o_t is still the NN of q' for t. □

Query updates By Lemma 4, when the query q moves to q', a straightforward way to check the validity of the current approximate result set S_a is to check whether each object o in S_a is closer to q' than o's KVNS. If so, the current S_a is still valid. Otherwise, we invoke the L-Appro algorithm at q' to recompute a new S_a. In order to further reduce the recomputation frequency, we propose the following incremental maintenance strategy.

When q moves to q', let $o_f \in S_a$ be the furthest object from q' in the current S_a. We use the following lemma to show that we can simply use o_f to check the validity of S_a instead of all objects in S_a.

Lemma 5 *Let t be the keyword o_f covers. When o_f is still the NN of q' for t, the current S_a gives a 3-approximation result to q'.*

Proof Let S_e be the exact result for the query q' and o_t be the object covering keyword t in S_e (recall that o_t is not necessarily the NN of q' for keyword t). Since o_f is the NN of q' for t, we have $d(q', o_f) \leq d(q', o_t)$. We also have $d(q', o_f) \leq \mathcal{C}(q', S_e)$, since $d(q', o_t) \leq \mathcal{C}(q', S_e)$. Because o_f is the furthest object from q' in the current S_a, the largest possible distance between two objects in S_a is $2 \cdot d(q', o_f)$ by the triangle inequality. Therefore, S_a satisfies the following inequalities:

$$\begin{aligned}\mathcal{C}(q', S_a) &= \max_{o \in S_a} d(q', o) + \max_{o_1, o_2 \in S_a} d(o_1, o_2)\\ &\leq d(q', o_f) + 2 \cdot d(q', o_f)\\ &\leq 3 \cdot d(q', o_f)\\ &\leq 3 \cdot \mathcal{C}(q', S_e),\end{aligned}$$

i.e., the current S_a still gives a 3-approximation result to q'. □

For example, in Fig. 7a, the NNs of q for t_1 and t_2 are o_0 and o_5, respectively. Thus, an approximate result set is $S_a = \{o_0, o_5\}$. When q moves to q', as $d(q', o_5) > d(q', o_0)$, o_5 is the furthest object from q' in S_a. From Fig. 7c, we find that q' is in the Voronoi cell $V(o_5)$, which means o_5 is still the NN for t_2. By Lemma 5, the current $S_a = \{o_0, o_5\}$ still gives a 3-approximation result to q', although the NN of q' for t_1 changes from o_0 to o_1 (as illustrated in Fig. 7b).

When o_f is no longer the NN of q' (for its corresponding keyword), we can actually use the t-Voronoi neighbor set $N_t(o_f)$ to efficiently find the new NN, and the method is stated in the following Lemma 6.

Lemma 6 *Let $o_t \in S_a$ be the NN of q for keyword t, o_m be the object having the minimum distance to q' in $N_t(o_t)$. If o_m is closer to q' than any object in $N_t(o_m)$, then o_m is the NN of q' for t.*

Proof Since o_m is a t-Voronoi neighbor of o_t, by definition, o_t is also a t-Voronoi neighbor of o_m, i.e., $o_t \in N_t(o_m)$. Thus, if o_m is closer to q' than any object in $N_t(o_m)$, by Lemma 4, o_m is the NN of q' for t. □

Based on above analysis, our query update process is described as follows: When q moves to q', we first check whether the furthest object o_f from q' in the current S_a is the NN of q' for its corresponding keyword, e.g., keyword t. If so, the current S_a is valid. Otherwise, we use the t-Voronoi neighbor set $N_t(o_f)$ of o_f to find the new NN o_m. If o_m can be found (by Lemma 6), we replace o_f by o_m and get a new $S_a = current\ S_a \backslash \{o_f\} \bigcup \{o_m\}$. We then repeat the above steps to check the validity of this new S_a. If o_m cannot be found, we invoke the L-Appro algorithm to recompute a new S_a. Algorithm 3 summarizes the query processing procedure.

Algorithm 3: AIM

Input : an MCSKQ q
Output: the new S_a

- $S_a \leftarrow L - Appro(q)$
- $reportResult(S_a)$
- **while** *query continues* **do**
  - *q moves to q'*
  - **while** *true* **do**
    - $o_f = \arg\max_{o \in S_a} d(q', o)$
    - $o_m = \arg\min_{o \in N_t(o_f)} d(q', o)$
    - **if** $d(q', o_f) < d(q', o_m)$ **then**
      - $q \leftarrow q'$
      - $reportResult(S_a)$
      - $break$
    - **else**
      - **if** *o_m is closer to q' than any object in $N_t(o_m)$* **then**
        - $S_a \leftarrow S_a \backslash \{o_f\} \bigcup \{o_m\}$
      - **else**
        - $S_a \leftarrow L - Appro(q')$
        - $q \leftarrow q'$
        - $reportResult(S_a)$
        - $break$

Complexity By Algorithm 3, the current approximate result set S_a is valid as long as the furthest object $o_f \in S_a$ is still the NN of the new query for its corresponding keyword t. Let the size of the data space be 1, then the area of the safe region for S_a is $R = \min\{\frac{1}{|\mathcal{O}_t|} \mid t \in q.\psi\}$ where $\mathcal{O}_t \subseteq \mathcal{O}$ and each object $o \in \mathcal{O}_t$ covers keyword t. According to [9], the recomputation frequency f is inversely proportional to the square root of the area of the safe region. Therefore, we have $f = O(\sqrt{\frac{1}{R}})$. Let $\mathcal{O}_\psi \subseteq \mathcal{O}$ be a set of objects and $o.\psi \bigcap q.\psi \neq \emptyset$ for each object $o \in \mathcal{O}_\psi$. Since the cost of the L-Appro algorithm is $O(|\mathcal{O}_\psi| \log |\mathcal{O}_\psi|)$ [12], the time complexity of AIM is $O(\sqrt{\frac{1}{R}} \cdot |\mathcal{O}_\psi| \log |\mathcal{O}_\psi|)$. Okabe et al. [33] have proved that the average number of edges per Voronoi cell does not exceed 6, i.e., there are at most $6|o.\psi|$ Voronoi neighbors for each object o. If a client–server-based system is used, since the size of S_a is at most $|q.\psi|$, the communication cost of AIM is $O(|q.\psi| \cdot f \cdot \max_{o \in \mathcal{O}_\psi} |o.\psi|)$ in the worst case. In our experiments, the largest size of $o.\psi$ is 36.

5.2 Advanced approximate maintenance

To provide users with better approximate results, in this section, we introduce the *advanced approximate maintenance* (AAM) algorithm. AAM adopts the H-Appro algorithms (cf. Sect. 2.1) in recomputation and can maintain a ρ-approximation result ($\rho \in \{1.375, 1.8\}$). Since AAM is based on MRSR$_{ad}$ (cf. Sect. 4.2) and AIM, it indicates that the proposed algorithms MRSR$_{ad}$ and AIM are general and flexible and can be integrated with any static CSKQ computation methods.

Given an MCSKQ q, AAM first computes an approximate result set S_a from the better one of the two candidates: One is the set S_n computed by the L-Appro algorithm, and the other is the set S_p having the minimum cost among all special feasible sets. When the query moves, AAM aims to maintain S_n and S_p. If so, there is no need to issue a new query. For the maintenance of S_n, we use the query update process of AIM (cf. Algorithm 3 Lines 3–19). For S_p maintenance, we compute the relaxed safe region RSR(S_p) using similar computation steps as the relaxed safe region RSR(S_e) of the exact result set S_e (cf. Sect. 4.1), i.e.,

$$RSR(S_p) = \bigcap_{o \in S_p} E_o^{\gamma_k},$$
$$E_o^{\gamma_k} = \{q' | d(q', o) + d(q, q') \leq \gamma_k - \mathcal{P}(S_p)\},$$

where $\mathcal{P}(S_p)$ is the maximum distance between any two objects in S_p and $\gamma_k = \mathcal{C}(q, S_k)$ with S_k being the k-th special feasible set of q. When the query moves within RSR(S_p), the new S_p remains to be one of the top-k special feasible sets of q. Thus, we adopt the query update process of MRSR$_{ad}$ (cf. Algorithm 2 Lines 5–19) to maintain S_p by replacing the top-k feasible sets with the top-k special feasible sets. Algorithm 4 summarizes the query processing procedure.

Algorithm 4: AAM

```
Input  : an MCSKQ q
Output: the new S_a
S_n ← AIM (Lines 1 to 2)
S_p ← MRSR_ad (Lines 1 to 4)
S_a ← min{S_n, S_p}
reportResult(S_a)
while query continues do
    q moves to q'
    maintain S_n
    maintain S_p
    S_a ← min{S_n, S_p}
    q ← q'
    reportResult(S_a)
```

6 The variants of MCSKQ

In this section, we show that the MRSR$_{ad}$ and AIM algorithms can also effectively solve MCSKQ with weighted objects, MCSKQ on road networks, and other variants of MCSKQ (e.g., MCSKQ with a group of query users, MCSKQ with the MinMax cost function).

6.1 MCSKQ with weighted objects

In real life, users may prefer to a far-away object with a much higher popularity than one close to their locations. Therefore, in this section, we consider the popularity of objects in MCSKQ and study the MCSKQ problem with weighted objects, i.e., weighted MCSKQ. Different from the original MCSKQ where objects are treated equally, weighted MCSKQ takes into account object weights when evaluating the cost of an object set. The object weights can capture aspects of objects such as user ratings, popularity, and text relevance to the query.

Following existing work [12], we use $w(q, o) = e^{-score}$ as the weight of an object o where $score \in [0, 1]$ can be the normalized user rating or relevance score to the query. We use the following *weighted MaxMax* cost function to evaluate the cost of a feasible set S, i.e.,

$$\mathcal{C}_w(q, S) = \alpha \cdot \max_{o \in S} d_w(q, o) + (1 - \alpha) \cdot \max_{o \in S} w(q, o) \cdot \max_{o_1, o_2 \in S} d(o_1, o_2),$$

where $d_w(o, q)$ is the *weighted distance* between object o and query q and $d_w(q, o) = w(q, o) \cdot d(q, o)$. Omitting the parameter α, we have

$$\mathcal{C}_w(q, S) = \max_{o \in S} d_w(q, o) + \max_{o \in S} w(q, o) \cdot \max_{o_1, o_2 \in S} d(o_1, o_2). \tag{9}$$

Weighted MCSKQ is to continuously find a feasible set S having the minimum cost measured by the cost function $\mathcal{C}_w(\cdot)$ when the query moves. Next, we discuss how to reuse the proposed algorithms MRSR$_{ad}$ and AIM to solve weighted MCSKQ efficiently.

Adaption of MRSR$_{ad}$ Let Eq. (9) be rewritten as

$$\mathcal{C}_w(q, S) = \max_{o \in S}\{w(q, o) \cdot (d(q, o) + \mathcal{P}_w(S))\}, \quad (10)$$

$$\mathcal{P}_w(S) = \frac{\max_{o_m \in S} w(q, o_m) \cdot \max_{o_1, o_2 \in S} d(o_1, o_2)}{w(q, o)}.$$

From Eq. (10), if we model each object $o \in S$ as a circle $C_o^{\mathcal{P}_w(S)}$ centered at o with a radius of $\mathcal{P}_w(S)$, then $w(q, o) \cdot (d(q, o) + \mathcal{P}_w(S))$ is the *furthest weighted distance* from q to circle $C_o^{\mathcal{P}_w(S)}$. Thus, the cost of S is the *maximum* furthest weighted distance from q to circles $C_o^{\mathcal{P}_w(S)}$ for $o \in S$. It is similar to the original MCSKQ where the cost of S is the maximum furthest distance from q to circles $C_o^{\mathcal{P}(S)}$ for $o \in S$ (cf. Eq. (3)). We are aware that, for the exact result set S_e computed by a static weighted CSKQ algorithm (e.g., wMaxMax-Exact [12]), we can also derive the relaxed safe region RSR(S_e) using the similar steps described in Section 4.1, which is the key component of MRSR$_{ad}$. The detailed computation process is as follows.

First, we introduce the concept of *weighted inside*. Given a circle C_q^{γ}, for an object $o \in S$, we say the corresponding circle $C_o^{\mathcal{P}_w(S)}$ is weighted inside C_q^{γ}, if and only if it satisfies

$$w(q, o) \cdot d(q, o) \leq \gamma - w(q, o) \cdot \mathcal{P}_w(S).$$

Then, by Eq. (10), we have $\mathcal{C}_w(q, S) \leq \gamma$ if and only if all the corresponding circles for the objects in S are weighted inside C_q^{γ}, where we say S is weighted inside C_q^{γ}.

Given the top-k feasible sets $S_1, S_2, \ldots, S_k$ ($S_e = S_1$) and a circle $C_q^{\gamma_k}$ centered at q with a radius of $\gamma_k = \mathcal{C}_w(q, S_k)$. Since the costs of the other (non-top-k) feasible sets are larger than γ_k, only $S_1, S_2, \ldots, S_k$ are weighted inside $C_q^{\gamma_k}$. When q moves to q' and the current S_e is weighted inside the circle $C_{q'}^{\gamma'}$ where $\gamma' = \gamma_k - d(q, q')$, as stated in the following Lemma 7, any non-top-k feasible set S_j ($j > k$) is not better than S_e, which means the result set of q' is the one having the minimum cost among the top-k feasible sets.

Lemma 7 *When the query q moves to q' and the current S_e is weighted inside the circle $C_{q'}^{\gamma'}$, for any non-top-k feasible set S_j ($j > k$): $\mathcal{C}_w(q', S_j) > \mathcal{C}_w(q', S_e)$.*

Proof When the current S_e is weighted inside $C_{q'}^{\gamma'}$, we have $\mathcal{C}_w(q', S_e) \leq \gamma'$. Since only the top-$k$ feasible sets are weighted inside $C_q^{\gamma_k}$, for any non-top-k feasible set S_j for $j > k$, S_j is not weighted inside $C_q^{\gamma_k}$. Since $C_{q'}^{\gamma'}$ is an inscribed circle of $C_q^{\gamma_k}$, S_j is not weighted inside $C_{q'}^{\gamma'}$, i.e., $\mathcal{C}_w(q', S_j) > \gamma'$. Combining the above, we have $\mathcal{C}_w(q', S_j) > \mathcal{C}_w(q', S_e)$. □

Based on Lemma 7, we derive the RSR of S_e which is a region within which when q moves, the circle $C_o^{\mathcal{P}_w(S_e)}$ for each $o \in S_e$ is still weighted inside $C_{q'}^{\gamma'}$, i.e.,

$$RSR(S_e) = \bigcap_{o \in S_e} R_o^{\gamma_k}, \quad (11)$$

$$R_o^{\gamma_k} = \{q' | d(q', o) + \frac{d(q, q')}{w(q, o)} \leq \frac{\gamma_k}{w(q, o)} - \mathcal{P}_w(S_e)\}.$$

Similarly, for the other feasible sets $S_2, \ldots, S_k$, we can also obtain their corresponding RSRs with Eq. (11). Using these k RSRs, MRSR$_{ad}$ can be directly applied to weighted MCSKQ, by replacing $E_o^{\gamma_k}$ with $R_o^{\gamma_k}$ in Algorithm 2, and we call this algorithm $wMRSR_{ad}$.

Adaption of AIM For a static weighted CSKQ q, wMaxMax-Appro [12] computes an approximate result set S_a consists of the *weighted NNs* which have the smallest weighted distances for each query keyword, i.e.,

$$S_a = \bigcup_{t \in q.\psi} \{o_t\},$$

$$o_t = \arg\min_{t \in o.\psi} d_w(q, o).$$

It can be proved that S_a gives a $(1+2e)$-approximation result ($e \approx 2.718$).

For weighted MCSKQ with wMaxMax-Appro in recomputation, in order to reduce the recomputation frequency, we introduce the concept of the *keyword-based weighted Voronoi neighbor set* (wKVNS), which is used to check whether the objects in S_a are still the weighted NNs of the new query.

Different from KVNS (cf. Sect. 5.1) which is constructed by 1-order Voronoi diagram, we use *multiplicatively weighted Voronoi* (MW-Voronoi) diagram [35] to construct wKVNS. Given a set $\mathcal{O}$ of data objects, the MW-Voronoi diagram of $\mathcal{O}$ is the collection of MW-Voronoi regions of all objects in $\mathcal{O}$. These regions form a disjoint and complete partitioning of the spatial domain. For an object $o \in \mathcal{O}$, when the query q moves in o's MW-Voronoi region $V_w(o)$, o has a smaller weighted distance to q than any other object.

Based on MW-Voronoi, we formalize wKVNS. Let a set $\mathcal{O}_t \subseteq \mathcal{O}$ contain the objects $o \in \mathcal{O}_t$ having keyword t. Given the MW-Voronoi of $\mathcal{O}_t$, for any two objects $o_i, o_j \in \mathcal{O}_t$, o_j is a t-weighted Voronoi neighbor of o_i if their MW-Voronoi regions share an edge, i.e., $V_w(o_i)||V_w(o_j)$. We call the set $\mathcal{O}'_t \subset \mathcal{O}_t$, denoted by $N_t^w(o_i)$, that contains all the t-weighted Voronoi neighbors of o_i the $t - weightedVoronoineighborset$ of o_i.

Using wKVNS, we reduce the recomputation frequency of wMaxMax-Appro. When q moves to q', a straightforward way to check the validity of the current S_a, i.e., whether S_a gives a $(1 + 2e)$-approximation result to q', is to check whether each object o in S_a has a smaller weighted distance than o's wKVNS. If so, S_a is still valid. Otherwise, we invoke wMaxMax-Appro at q' to recompute a new S_a.

In addition, the incremental maintenance strategy used in AIM (cf. Lemma 5) still works, as stated in the following

Lemma 8. Therefore, we can simply use the object having the maximum weighted distance to check the validity of S_a, and AIM can also be directly applied to weighted MCSKQ by replacing KVNS with wKVNS in Algorithm 3, which we call *wAIM*.

Lemma 8 *When the query q moves to q', let o_{mw} be the object covering keyword $t \in q.\psi$ and having the maximum weighted distance to q' in the current S_a. When o_{mw} is also the weighted NN of q' for keyword t, we have*

$$\mathcal{C}_w(q', S_a) \leq (1 + 2e) \cdot \mathcal{C}_w(q', S_e),$$

where S_e is the exact result set of q'.

Proof Since there must exist an object o_m in S_e containing keyword t, by Eq. (9) we have

$$d_w(q', o_{mw}) < d_w(q', o_m) < \mathcal{C}_w(q', S_e).$$

Let o_f be the furthest object from q' in S_a. The maximum distance between any two objects in S_a is $2 \cdot d(q', o_f)$ by the triangle inequality. Since o_{mw} has the maximum weighted distance to q', we have $d_w(q', o_f) \leq d_w(q', o_{mw})$. Thus, it holds that

$$d(q', o_f) \leq \frac{d_w(q', o_{mw})}{w(q', o_f)}.$$

Then, we have

$$\begin{aligned}
\mathcal{C}_w(q', S_a) &= \max_{o \in S_a} d_w(q', o) \\
&\quad + \max_{o \in S_a} w(q', o) \cdot \max_{o_1, o_2 \in S_a} d(o_1, o_2) \\
&\leq d_w(q', o_{mw}) + \max_{o \in S_a} w(q', o) \cdot 2d(q', o_f) \\
&\leq d_w(q', o_{mw}) + \frac{\max_{o \in S_a} w(q', o)}{w(q', o_f)} \cdot 2d_w(q', o_{mw}).
\end{aligned}$$

Since $\max_{o \in S_a} w(q', o) \leq 1$ and $w(q', o_f) \geq e^{-1}$, we have

$$\begin{aligned}
\mathcal{C}_w(q', S_a) &\leq d_w(q', o_{mw}) + \frac{\max_{o \in S_a} w(q', o)}{w(q', o_f)} \cdot 2d_w(q', o_{mw}) \\
&\leq d_w(q', o_{mw}) + 2e \cdot d_w(q', o_{mw}) \\
&= (1 + 2e) \cdot d_w(q', o_{mw}) \\
&\leq (1 + 2e) \cdot \mathcal{C}_w(q', S_e)
\end{aligned}$$

□

6.2 MCSKQ on road networks

In the real world, the movement of users is constrained by road networks, on which the spatial proximity between objects is determined by the network distance instead of the Euclidean distance. Thus, in this section, we study MCSKQ on road networks and we find that MRSR_{ad} and AIM also work for this variant.

A road network is defined as an undirected weighted graph $G = (N, E, W)$, where N is a set of vertices/nodes, E a set of edges, and W a set of weights representing the length of edges. Let $\mathcal{O}$ be a set of two-dimensional static objects on the edges, and each object $o \in \mathcal{O}$ has a location $o.\lambda$ and a set of keywords $o.\psi$. Here, the location $o.\lambda$ of an object is denoted by a triple $(n_i, n_j, dist)$ where n_i and n_j are the vertices of the edge $e_{i,j}$ on which the object is located, and $dist$ is the distance from o to n_i along the edge $e_{i,j}$ $(i < j)$. Given two objects o_i and o_j, we use $d_r(o_i, o_j)$ to denote the length of $path(o_i, o_j)$, which is the shortest path between o_i and o_j. Figure 8a shows an example of a road network, where squares denote the vertices and solid dots denote the objects. The locations and the keywords of each object are shown in Fig. 8b. Figure 8c shows the lengths of the edges. Take $o_2(\{t_1\}, (n_3, n_4, 1))$ as an example, it contains the keyword t_1, locates on the edge $e_{3,4}$, and is one unit away from vertex n_3. The distance $d_r(o_1, o_2)$ between o_1 and o_2 is computed as $d_r(o_1, o_2) = d_r(o_1, n_4) + d_r(n_4, o_2) = 2 + 3 = 5$.

The static CSKQ on road networks is studied in [14] with the cost function

$$\mathcal{C}_r(q, S) = \max_{o \in S} d_r(q, o) + \max_{o_1, o_2 \in S} d_r(o_1, o_2).$$

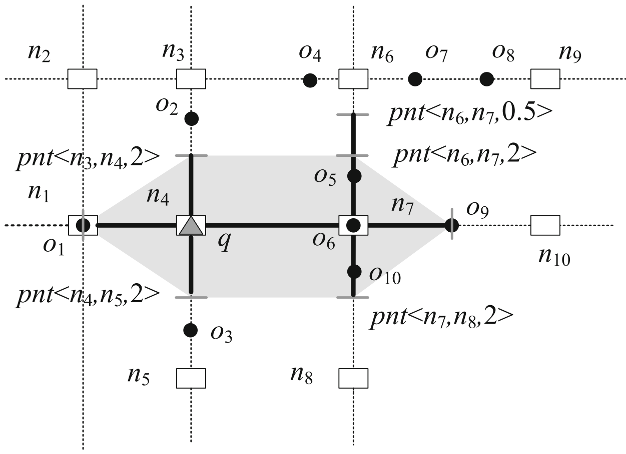
For MCSKQ on road networks, we aim to continuously find a feasible set S that has the minimum cost measured by this cost function $\mathcal{C}_r(\cdot)$ when the query moves.

Two static algorithms Sliding Window (SW) and NEB [14] are proposed for static CSKQ on road networks. SW computes the exact result set S_e (briefly reviewed in Sect. 7). NEB computes an approximate result set S_a under a 3-approximation ratio, which consists of the NNs for each query keyword. We next discuss how the proposed algorithms MRSR_{ad} and AIM can solve MCSKQ on road networks using SW and NEB in recomputation, respectively.

Recall the cost function we used in the original MCSKQ

$$\mathcal{C}(q, S) = \max_{o \in S} d(q, o) + \max_{o_1, o_2 \in S} d(o_1, o_2).$$

We can see that the only difference between $\mathcal{C}_r(\cdot)$ and $\mathcal{C}(\cdot)$ is the distance measurement, i.e., replacing the Euclidean distance $d(\cdot)$ with the network distance $d_r(\cdot)$. Given any three objects o_1, o_2, and o_3, for the network distance $d_r(\cdot)$ the following conditions hold.



(a)

Object
o_1 ($\{t_2\}$,(n_1,n_2,0))
o_2 ($\{t_1\}$,(n_3,n_4,1))
o_3 ($\{t_3\}$,(n_4,n_5,3.5))
o_4 ($\{t_3\}$,(n_3,n_6,3))
o_5 ($\{t_1\}$,(n_6,n_7,2.5))
o_6 ($\{t_3\}$,(n_4,n_7,0))
o_7 ($\{t_2\}$,(n_6,n_9,1))
o_8 ($\{t_3\}$,(n_6,n_9,3))
o_9 ($\{t_1$,$t_2\}$,(n_7,n_{10},2))
o_{10} ($\{t_2\}$,(n_7,n_8,1.5))

(b)

Edge	Length
$e_{1,2}$	4
$e_{1,4}$	2
$e_{2,3}$	2
$e_{3,4}$	4
$e_{3,6}$	4
$e_{4,5}$	5
$e_{4,7}$	4
$e_{6,7}$	4
$e_{6,9}$	4
$e_{7,8}$	5
$e_{7,10}$	4

(c)

Fig. 8 An example for MCSKQ on a road network

1. $d_r(o_1, o_2) \geq 0$;
2. $d_r(o_1, o_2) = d_r(o_2, o_1)$;
3. $d_r(o_1, o_2) = 0, iff\ o_1 = o_2$;
4. $d_r(o_1, o_2) \leq d_r(o_1, o_3) + d_r(o_3, o_2)$.

The established lemmas for AIM using Euclidean distance only require these properties. Thus, they can also apply to the road network distance. Therefore, AIM (cf. Algorithm 3) still works for MCSKQ on road networks, and we only need to make the following adaptations: (1) using $d_r(\cdot)$ to replace $d(\cdot)$; (2) using the VN^3 method [36] to precompute and store the KVNS for each object on road networks. We call this algorithm *rAIM*.

We proceed to present how to derive the relaxed safe region RSR(S_e) on road networks. The following Lemma 9 shows that as long as the inequality $d_r(q, q') < \mathcal{C}_r(q, S_k) - \mathcal{C}_r(q', S_e)$ holds (S_k is the k-th feasible set), S_e is still better than any other non-top-k feasible sets. Note that we do not use the above condition (2) in proving the following lemma, which means this lemma suits the case where the network distance is asymmetric, i.e., $d_r(o_1, o_2)$ may not be equal to $d_r(o_2, o_1)$.

Lemma 9 *If $d_r(q, q') < \mathcal{C}_r(q, S_k) - \mathcal{C}_r(q', S_e)$, for any non-top-k feasible set S_j ($j > k$): $\mathcal{C}_r(q', S_j) > \mathcal{C}_r(q', S_e)$.*

Proof By the cost function $\mathcal{C}_r(\cdot)$, we have $\mathcal{C}_r(q', S_j) = \max_{o \in S_j} d_r(q', o) + \max_{o_1, o_2 \in S_j} d_r(o_1, o_2)$. By the above Inequality (4), we have

$$d_r(q, o) \leq d_r(q, q') + d_r(q', o).$$

Then, it holds that

$$\max_{o \in S_j} d_r(q', o) \geq \max_{o' \in S_j} (d_r(q, o') - d_r(q, q')).$$

Thus, we have

$$\begin{aligned}
\mathcal{C}_r(q', S_j) &\geq \max_{o' \in S_j} \{d_r(q, o') - d_r(q, q')\} \\
&\quad + \max_{o_1, o_2 \in S_j} d_r(o_1, o_2) \\
&= \max_{o' \in S_j} d_r(q, o') + \max_{o_1, o_2 \in S_j} d_r(o_1, o_2) - d_r(q, q') \\
&= \mathcal{C}_r(q, S_j) - d_r(q, q') \\
&\geq \mathcal{C}_r(q, S_k) - d_r(q, q').
\end{aligned}$$

Since $d_r(q, q') < \mathcal{C}_r(q, S_k) - \mathcal{C}_r(q', S_e)$, we have $\mathcal{C}_r(q', S_e) < \mathcal{C}_r(q, S_k) - d_r(q, q')$. As a result, we get $\mathcal{C}_r(q', S_j) > \mathcal{C}_r(q', S_e)$. □

By Lemma 9, we can use the possible locations of q', having $d_r(q, q') < \mathcal{C}_r(q, S_k) - \mathcal{C}_r(q', S_e)$, to compute RSR($S_e$), i.e.,

$$\begin{aligned}
&RSR(S_e) = \bigcap_{o \in S_e} E_o^{\gamma_k}, \\
&E_o^{\gamma_k} = \{q' | d_r(q', o) + d_r(q, q') \leq \gamma_k - \mathcal{P}(S_e)\},
\end{aligned} \tag{12}$$

where $\gamma_k = \mathcal{C}_r(q, S_k)$ and $\mathcal{P}(S_e) = \max_{o_1, o_2 \in S_e} d_r(o_1, o_2)$. Similarly, for the other feasible sets $S_2, S_3, \ldots, S_k$, we can also obtain their corresponding RSRs with the above equation. Using these k RSRs, MRSR$_{ad}$ can be applied to MCSKQ on road networks. Moreover, we add a procedure in MRSR$_{ad}$ to identify the edges and line segments (e&s) included in the RSRs. We call this algorithm $rMRSR_{ad}$. Next, we explain the computation of e&s included in RSR(S_i) ($i \in [1, k]$).

To find the e&s included in RSR(S_i), based on Eq. (12), we can first find the e&s included in $E_o^{\gamma_k}$ for each $o \in S_i$. Then, the intersection of all the e&s is the result. For the first (and most important) step, we only need to identify the boundary points of $E_o^{\gamma_k}$, using which we can easily find the e&s included in $E_o^{\gamma_k}$. Specifically, we adapt the *Incremental Network Expansion* (INE) algorithm [37] around q and o to first identify the edges that cross the boundary of $E_o^{\gamma_k}$, and then the boundary points of $E_o^{\gamma_k}$. We use a priority queue Q_r to keep the vertices to be examined in ascending order of the sum aggregate distance to o and q. Initially, Q_r contains the vertices on $path(q, o)$ and their immediate neighbors. We then check the vertices v in Q_r one by one. When v is inside $E_o^{\gamma_k}$, i.e., $d_r(v, o) + d_r(q, v) \leq \gamma_k - \mathcal{P}(S_i)$, we expand at v and put all the immediate neighbors of v into Q_r. When encountering a vertex v locates outside $E_o^{\gamma_k}$, we know there must be edges on $path(q, v)$ and $path(v, o)$ crossing the boundary of $E_o^{\gamma_k}$. Therefore, we add all the intersection points of $path(q, v)$ and $E_o^{\gamma_k}$, $path(v, o)$ and $E_o^{\gamma_k}$, respectively, into the result list as boundary points of $E_o^{\gamma_k}$. We repeat the previous steps until Q_r is empty. Algorithm 5 summarizes these steps.

Algorithm 5: Compute Boundary Points

Input : $E_o^{\gamma_k}$
Output: the boundary points of $E_o^{\gamma_k}$

1 create a queue Q_r and an empty list L of boundaries
2 insert vertices adjacent to $path(q, o)$ into Q_r
3 **while** *not* $Q_r.empty()$ **do**
4     $v \leftarrow Q_r.dequeue()$
5   **if** $d_r(v, o) + d_r(q, v) \leq \gamma_k - \mathcal{P}(S_i)$ **then**
6   insert all immediate neighbors of v into Q_r
7   **else**
8   identify the boundary points on $path(q, v)$ and $path(v, o)$, then insert them into L
9 return L

We explain Algorithm 5 with the following example. In Fig. 8, given the query q located at vertex n_4 with query keywords $q.\psi = \{t_1, t_2, t_3\}$, we find the top-3 feasible sets $S_1 = \{o_5, o_6, o_{10}\}$, $S_2 = \{o_1, o_2, o_3\}$, and $S_3 = \{o_4, o_5, o_7\}$. To compute the e&s included in RSR(S_1), we need to compute the boundary points of $E_{o_5}^{\gamma_3}$, $E_{o_6}^{\gamma_3}$, and $E_{o_{10}}^{\gamma_3}$ ($\gamma_3 = \mathcal{C}_r(q, S_3)$). A summary of execution steps for computing the boundary points of $E_{o_5}^{\gamma_3}$ is given in Table 4 and detailed explanations are given as follows. For convenience, we use $pnt(n_i, n_j, l)$ to denote a boundary point on edge $e_{i,j}$ with distance l to vertice n_i, and $seg(n_i, n_j, l)$ to denote a line segment on edge $e_{i,j}$ whose two end points are n_i and $pnt(n_i, n_j, l)$.

Step 1 Vertices n_1, n_3, n_6, n_{10}, n_5, and n_8, which are adjacent to $path(q, o_5)$ ($n_4 \rightarrow n_7 \rightarrow o_5$), are inserted into Q_r (Line 2).
Step 2 Vertex n_1 is retrieved from Q_r. Since $d_r(n_1, o_5) + d_r(q, n_1) \leq \gamma_3 - \mathcal{P}(S_1)$, we insert n_2, immediate neighbor of n_1, into Q_r (Lines 5–6).

Table 4 Example run of Algorithm 5

Step	Vertex	Q_r	Boundary point
1	–	$< n_1, n_3, n_6, n_{10}, n_5, n_8 >$	–
2	n_1	$< n_3, n_6, n_{10}, n_2, n_5, n_8 >$	–
3	n_3	$< n_6, n_{10}, n_2, n_5, n_8 >$	$pnt(n_3, n_4, 2)$, $pnt(n_6, n_7, 0.5)$
4	n_6	$< n_{10}, n_2, n_5, n_8 >$	–
5	n_{10}	$< n_2, n_5, n_8 >$	$pnt(n_7, n_{10}, 2)$
6	n_2	$< n_5, n_8 >$	n_1
7	n_5	$< n_8 >$	$pnt(n_4, n_5, 2)$
8	n_8	$<>$	$pnt(n_7, n_8, 2)$

Step 3 Vertex n_3 is retrieved from Q_r. Since $d_r(n_3, o_5) + d_r(q, n_3) > \gamma_3 - \mathcal{P}(S_1)$, we identify the boundary points on $path(q, n_3)$ and $path(n_3, o_5)$. Two boundary points $pnt(n_3, n_4, 2)$ and $pnt(n_6, n_7, 0.5)$ are identified and inserted into L (Line 8).
Steps 4–8 We examine n_6, n_{10}, n_2, n_5, and n_8, respectively, and identify the boundary points, which are listed in Table 4.

We continue with the above example to show the e&s included in RSR(S_1). After we obtain the boundary points of $E_{o_5}^{\gamma_3}$ (which are n_1, $pnt(n_3, n_4, 2)$, $pnt(n_4, n_5, 2)$, $pnt(n_6, n_7, 0.5)$, $pnt(n_7, n_8, 2)$, and $pnt(n_7, n_{10}, 2)$), the e&s corresponding to these boundary points can be easily obtained (which are e_{n_1,n_4}, e_{n_4,n_7}, $seg(n_4, n_3, 2)$, $seg(n_4, n_5, 2)$, $seg(n_7, n_6, 0.5)$, $seg(n_7, n_8, 2)$, and $seg(n_7, n_{10}, 2)$, respectively, shown as the thick line segments in Fig. 8a). Using the same steps we can also obtain the e&s included in $E_{o_6}^{\gamma_3}$ and $E_{o_{10}}^{\gamma_3}$. The e&s included in RSR(S_1) are the intersection of the e&s included in $E_{o_5}^{\gamma_3}$, $E_{o_6}^{\gamma_3}$, and $E_{o_{10}}^{\gamma_3}$, which are e&s in the gray region of Fig. 8a.

6.3 Other variants of MCSKQ

The proposed algorithms may be adapted to handle variants of MCSKQ, such as MCSKQ with a group of query users and MCSKQ with the MinMax cost function, etc.
MCSKQ with a group of users (or group-based MCSKQ) aims to continuously return a set of objects that cover all the query keywords. These objects are close to the group of moving users and are close to each other. We use the following cost function $\mathcal{C}_g(U, S)$ [8] to evaluate the cost of a feasible set S, where U denotes the group of query users.

$$\mathcal{C}_g(U, S) = \max_{o \in S, u \in U} d(u, o) + \max_{o_1, o_2 \in S} d(o_1, o_2).$$

Group-based MCSKQ is to continuously find a feasible set S having the minimum cost measured by $\mathcal{C}_g(\cdot)$ when the group of users moves. We can reuse the proposed algorithms MRSR$_{ad}$ and AIM to effectively maintain the result set of group-based MCSKQ. For the exact result set maintenance, let $\gamma_k = \mathcal{C}_g(U, S_k)$, and for each user $u \in U$, we use γ_k to derive RSR(u, S_e), i.e.,

$$RSR(u, S_e) = \bigcap_{o \in S_e} \{u' | d(u', o) + d(u, u') \leq \gamma_k - \max_{o_1, o_2 \in S_e} d(o_1, o_2)\}.$$

As long as all the users move within their corresponding RSR, the new answer is the one having the minimum cost among the top-k feasible sets (we omit the proof). For each user $u \in U$, the nearest neighbor set consists of the nearest neighbors of u for each query keyword. From the existing work [8], we know that the one having the minimum cost among these $|U|$ nearest neighbor sets gives a 5-approximation result. Thus, for this result set maintenance, we can use AIM to maintain each nearest neighbor set, respectively, and return the set with the minimum cost.

MCSKQ with MinMax The $MinMax$ cost function $\mathcal{C}_{min}$ (q, S) is denoted as:

$$\mathcal{C}_{min}(q, S) = \min_{o \in S} d(q, o) + \max_{o_1, o_2 \in S} d(o_1, o_2).$$

This function is preferable when users expect the nearest object in a result set to be close to the query location [12]. We can also adapt our algorithms to maintain the result set of MCSKQ with MinMax. For the exact result set maintenance, we can use the only object $o_k \in S_e$, having $d(q, o_k) + \max_{o_1, o_2 \in S_e} d(o_1, o_2) = \mathcal{C}_{min}(q, S_e)$, to derive RSR($S_e$), i.e.,

$$RSR(S_e) = \{q' | d(q', o_k) + d(q, q') \leq \gamma_k - \mathcal{P}(S_e)\},$$

where $\gamma_k = \mathcal{C}_{min}(q, S_k)$ and $\mathcal{P}(S_e) = \max_{o_1, o_2 \in S_e} d(o_1, o_2)$. As long as the query moves in RSR(S_e), the new answer remains to be one of the top-k feasible sets of q (we omit the proof). From the existing work [12], we know that the nearest neighbor set gives a 3-approximation result. For this result set maintenance, we can also use the furthest object to check the validity of it (we omit the proof). Thus, AIM can be directly used.

From the above analysis, we can see that for a variant of MCSKQ, MRSR$_{ad}$ works as long as we can find a region that only includes the top-k feasible sets and the exact result maintenance can be done by maintaining each object in the set separately. AIM works as long as the nearest neighbor set is an approximate result.

7 Experiment

7.1 Settings

Data sets Both real and synthetic data sets are used in the experiments. We use two real-world data sets, whose properties are shown in Table 5. The first data set contains 1,030,754 tweets in Los Angeles extracted based on coordinates from the data set used in [38]. Each tweet with geo-location from Los Angeles is considered a geo-textual object. The second data set contains 50,334 Foursquare check-in venues in New York City and nearby suburbs extracted based on coordinates from the data set used in [39]. We denote the two data sets by **LA** and **NY**, respectively. We also use synthetic data sets of different sizes (2–10 million objects) to conduct a scalability test. The synthetic data sets are generated from the LA data set. To generate a data set $\mathcal{O}$ of size n, we first insert all the objects from LA into $\mathcal{O}$ and then repeatedly create objects in $\mathcal{O}$ such that $\mathcal{O}$ has a similar spatial distribution as LA until $|\mathcal{O}| = n$. For each newly created object o in $\mathcal{O}$, we randomly pick a document from the text descriptions of the objects in LA. We use a real-world road network data set CA[4] for MCSKQ on road networks. The CA data set contains 21,048 nodes and 21,693 edges. We generate randomly 80,000 objects for CA, and for each object, we randomly pick a document from the text descriptions of the objects in the LA data set.

Algorithms

We empirically compare the two exact algorithms SRSR and MRSR$_{ad}$ (cf. Sect. 4), and two approximate algorithms AIM and AAM (cf. Sect. 5) with three sampling-based algorithms denoted by $BASE_e$, $BASE_g$, and $BASE_a$, which invoke MaxMax-Exact [12], MaxMax-Appro1 [12], and MaxMax-Appro2 [12] (cf. Sect. 2.1) at every timestamp, respectively. Similar to the MaxMax-Exact algorithm, we compute the top-k feasible sets, in the recomputation of SRSR and MRSR$_{ad}$, by first finding the most infrequent query keyword t_{inf}, and processing the objects containing t_{inf} (the pivot) in ascending order of their distances to q. We then perform an exhaustive search on each pivot, which aims to find the best feasible set containing the pivot, and maintain the current top-k feasible sets. Once we reach a pivot whose distance is larger than the cost of the k-th feasible set, we stop and return the current top-k feasible sets. For AIM, we use MaxMax-Appro1 in recomputation, and for AAM, we use MaxMax-Appro2 to find the top-k special feasible sets. For MCSKQ with weighted objects and MCSKQ on road networks, we evaluate wMRSR$_{ad}$, wAIM, rMRSR$_{ad}$, and rAIM with sampling-based algorithms, which will be described in detail later. Note that, in this section, we call the algorithms used in recomputation static algorithms.

[4] http://www.cs.utah.edu/~lifeifei/.

Table 5 Data set properties

Property	LA	NY
Number of objects	1,030,754	50,334
Number of unique words	1,065,061	379
Avg number of words per object	9.68	3.23

Table 6 Experiment parameters

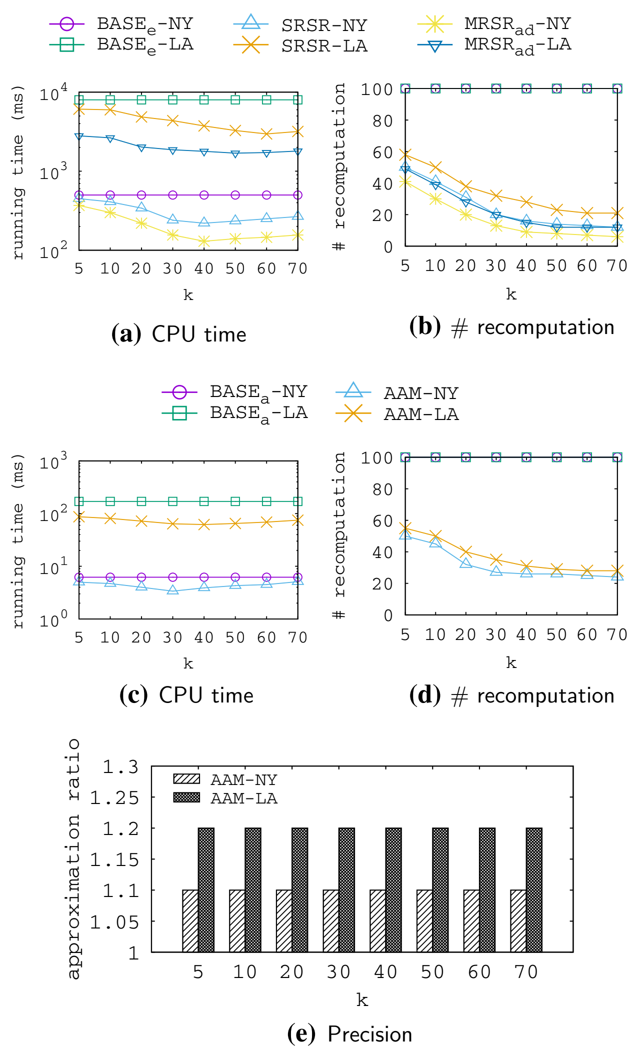
Parameter	Default	Values
Data set	LA	LA, NY
k	–	5, 10, 20, 30, 40, 50, 60, 70
$\|q.\psi\|$	6	2, 4, 6, 8, 10
Trajectory interval	200	50, 100, 200, 400, 800
Query trajectory	directional	directional, random, real
Data set size	–	2M, 4M, 6M, 8M, 10M
α	0.5	0.1, 0.3, 0.5, 0.7, 0.9

We use IR-tree [40] in the static algorithms and set the page size to 4KB (100 data objects per page). The experiments are conducted on a desktop computer with a 3.40 GHz Intel Core i7-6700 CPU, 16 GB memory, and 64-bit Windows operating system. All algorithms are implemented in C++.

Queries We generate two types of trajectories for the query object, "random" and "directional". In the random trajectories, the query object starts at a random point in the data space and moves toward a randomly chosen new direction at every timestamp. In the directional trajectories, the query object also starts at a random point and chooses a random direction, but it then keeps moving toward this direction until reaching the boundary of the space, where a new direction within the space is randomly chosen. By default, between two timestamps (i.e., two query computations) the query object moves for a distance interval that is randomly generated between 1 and 200 m (i.e., the maximum length of trajectories is 20 km). We also use a real trajectory data set *Buses*[5] for query. This data set consists of 145 trajectories, and each trajectory consists of location information for a bus within a day, collected every 30 s. We map this data set on to the space of the LA data set.

We generate 20 trajectories (100 timestamps each) for each set of experiments, and the query keywords are generated randomly within the word domain of the used data set. We report the average CPU time per timestamp and the average total number of times that the static algorithms are invoked for each query (# recomputation), which also indicates the communication cost if a client–server-based system is used. We also report the average approximation ratio of the approximate algorithms computed by $\frac{C(q,S_a)}{C(q,S_e)}$, where S_a is an approximate result set and S_e is the exact result set.

We vary the number of query keywords $|q.\psi|$, the top-k feasible sets size k, query computation distance interval, and data set cardinality in the experiments. The value ranges and default values of these parameters are summarized in Table 6.



(a) CPU time (b) # recomputation (c) CPU time (d) # recomputation (e) Precision

Fig. 9 Effect of k

7.2 Results

Effect of k Recall that SRSR, MRSR$_{ad}$, and AAM all use relaxed safe regions to reduce the number of recomputation. Specifically, SRSR and MRSR$_{ad}$ use the top-k feasible sets and AAM uses the top-k special feasible sets to compute the relaxed safe regions, respectively. In the first set of experiments, we test the effect of k and aim to find its optimal value to be used in the rest of the experiments.

As Fig. 9 shows, the CPU time (note the log scale) and the number of recomputation of the three algorithms both drop initially as k increases, which is expected because a larger number of (special) feasible sets leads to a larger relaxed safe

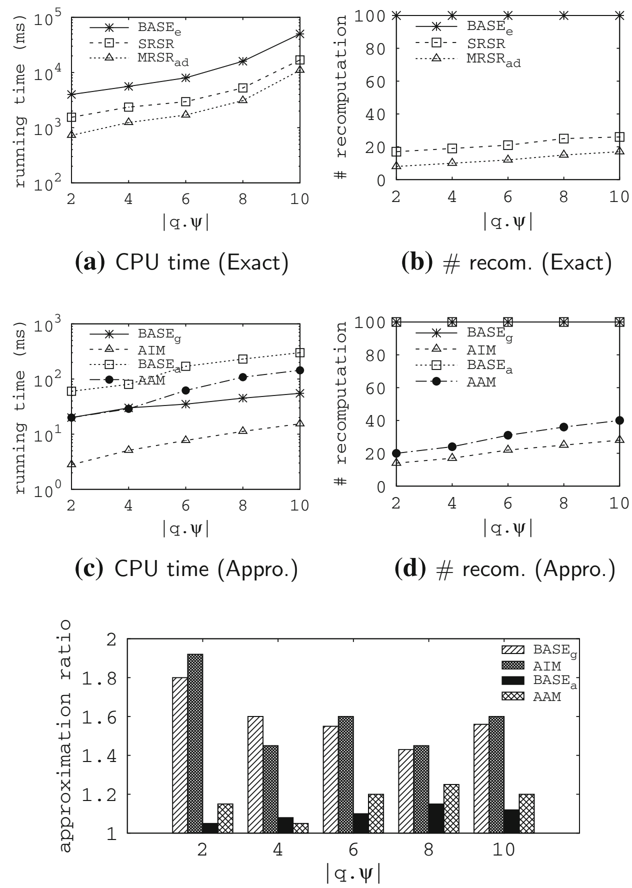
[5] http://www.chorochronos.org/.

region, reducing the frequency of recomputation. However, when k continues to increase, the benefit of larger relaxed safe regions is less significant, which results from higher computation cost of the top-k (special) feasible sets and higher query maintenance cost.

We can observe that SRSR shows the best performance at $k = 40$ on the NY data set and $k = 60$ on the LA data set; MRSR$_{ad}$ shows the best performance at $k = 40$ on the NY data set and $k = 50$ on the LA data set; AAM shows the best performance at $k = 30$ on the NY data set and $k = 40$ on the LA data set. The optimal value of k is larger on a larger data set (i.e., the LA data set) is because, on a larger data set, the query answer becomes invalid more frequently, which requires a larger relaxed safe region to reduce the recomputation frequency. In the rest of the experiments, we use these optimal values as the default values.

As SRSR and MRSR$_{ad}$ are exact algorithms, their approximation ratios are 1 no matter how k varies. Figure 9e shows the approximation ratios of AAM on NY and LA data sets. As we can see, the approximation ratio of AAM is consistently better than the worst-case ratio (i.e., 1.8) and is not affected by k. This is because the approximate result set computed by AAM is the better one of the two candidates: One is computed by the L-Appro algorithm, and the other is the best special feasible set, which are not affected by k.

Effect of $|q.\psi|$ Next we test the effect of the number of query keywords. In Figs. 10 and 11, we show the algorithm performance when the synthetic trajectories and the real trajectories are used, respectively. We can see that they have similar patterns. BASE$_e$, SRSR, and MRSR$_{ad}$ involve computing the exact result set S_e or top-k feasible sets, which are NP-hard. As the number of keywords increases, their computational costs increase exponentially. As shown in Fig. 10a, SRSR and MRSR$_{ad}$ perform much better than BASE$_e$ because they use RSRs to reduce the recomputation frequency. Since MRSR$_{ad}$ uses k RSRs to maintain the exact result set, it has the best performance among the exact algorithms. The query time is reduced by up to 80% compared with BASE$_e$. As expected, the two approximate algorithms spend much less time than the exact algorithms. As shown in Fig. 10c, AIM and AAM both perform better than their corresponding baseline algorithms, and the query times are reduced by 85% and 70%, respectively. This is because they use KVNS and RSRs to maintain the result set, which reduce the recomputation frequency and make the query processing more efficient. AIM has the lowest CPU cost, because it uses the L-Appro algorithm (MaxMax-Appro1) in recomputation with a low complexity. Because AAM needs to maintain two candidates, i.e., the nearest neighbor set and the best special feasible set, it has a higher recomputation frequency as shown in Fig. 10d. The performance gains of the proposed algorithms are comparable to existing studies on moving query processing [1,2,32]. As we can see from



(a) CPU time (Exact) **(b)** # recom. (Exact)

(c) CPU time (Appro.) **(d)** # recom. (Appro.)

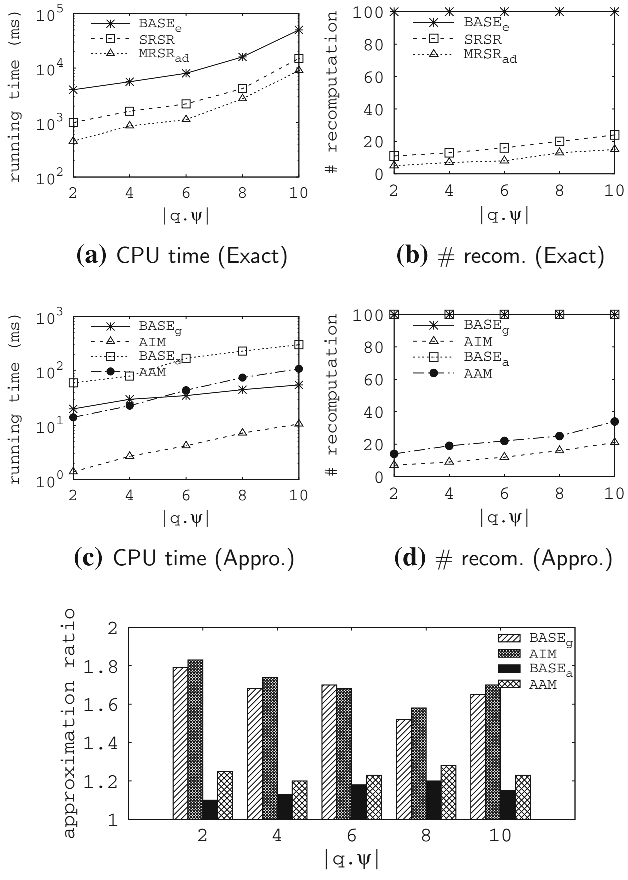
(e) Precision

Fig. 10 Effect of the number of query keywords on LA

Fig. 10e, the two approximate algorithms AIM and AAM both have good approximation ratios that are smaller than 2 and 1.3, respectively. These results confirm the effectiveness of AIM and AAM because the upper bounds of the approximation ratios are 3 and 1.8, respectively [12]. Similar patterns can be observed in the NY data set reported in Fig. 12. In the following experiments, we omit the results for NY.

Effect of query computation distance interval In Fig. 13, we compare the algorithms on the LA data set using directional query trajectories, where the query computation distance interval is varied from 50 to 800 m.

Note that, in order to better simulate the real-world scenario, for every interval value used (e.g., 200), the query object speed is *not* fixed at the interval value per timestamp (e.g., 200 meters per timestamp). Instead, the query object speed is randomly chosen between 1 and the interval value at every timestamp. We can see from Fig. 13a–d when the distance interval increases, the CPU cost and the number of recomputation both increase for our four algorithms while those of BASE$_e$, BASE$_g$, and BASE$_a$ stay stable. This is because the three baseline algorithms simply recompute the

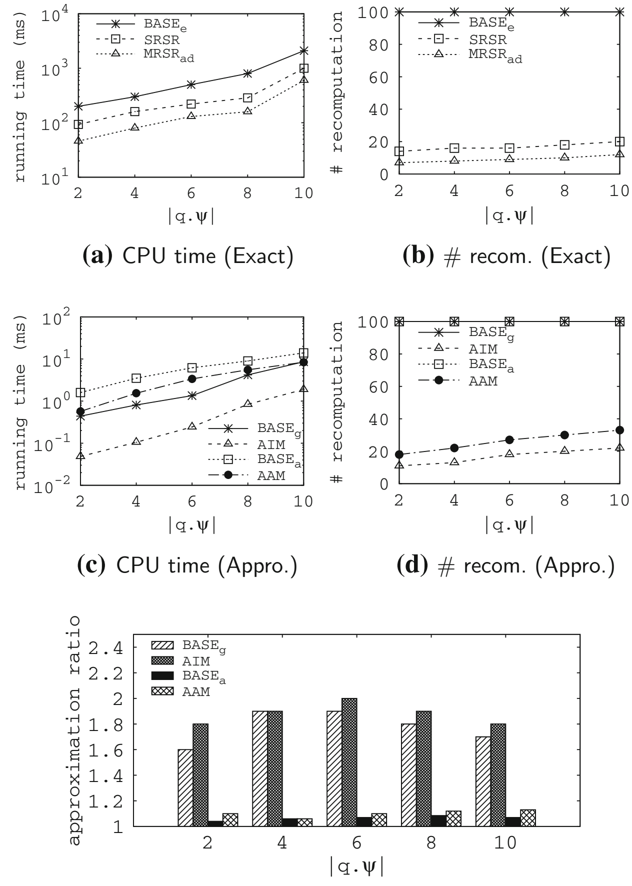


(a) CPU time (Exact) **(b)** # recom. (Exact)

(c) CPU time (Appro.) **(d)** # recom. (Appro.)

(e) Precision

Fig. 11 Effect of the number of query keywords on LA (using real trajectories)

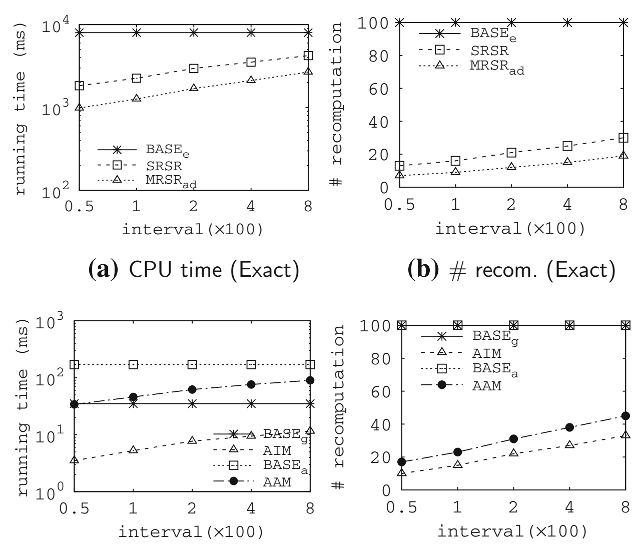


(a) CPU time (Exact) **(b)** # recom. (Exact)

(c) CPU time (Appro.) **(d)** # recom. (Appro.)

(e) Precision

Fig. 12 Effect of the number of query keywords on NY



(a) CPU time (Exact) **(b)** # recom. (Exact)

(c) CPU time (Appro.) **(d)** # recom. (Appro.)

Fig. 13 Effect of query computation distance interval (directional)

new exact result set S_e or the approximate result set S_a at every timestamp, which is not affected by the computation interval; the proposed algorithms rely on RSR and KVNS to reduce the recomputation frequency, which become invalid more frequent as the query distance interval increases. However, the four proposed algorithms still outperform the three baseline algorithms consistently, which validates the effectiveness of our proposed techniques.

We also compare the algorithms on random query trajectories. As shown in Fig. 14, the comparative performance of the algorithms is similar to that shown in Fig. 13. An observation is that the costs for our proposed four algorithms are generally lower on random query trajectories. This is because when the query object moves randomly instead of directionally, its probability of staying in the current safe region (i.e., RSR or KVNS) is higher and hence the recomputation frequency is lower.

Effect of data set size Next, we conduct a scalability test with 5 synthetic data sets whose numbers of objects vary from 2M to 10M. The results are reported in Fig. 15. As we can see, both the exact algorithms (i.e., SRSR and MRSR$_{ad}$) and the approximate algorithms (i.e., AIM and AAM) are scalable to large data sets with millions of objects. MRSR$_{ad}$

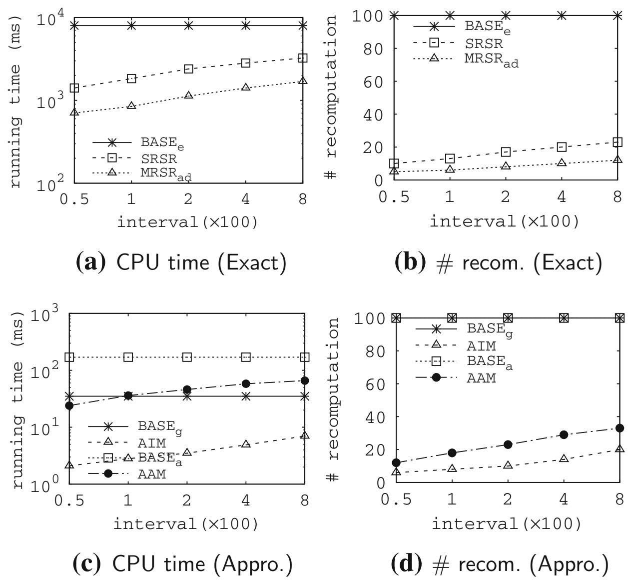


Fig. 14 Effect of query computation distance interval (random)

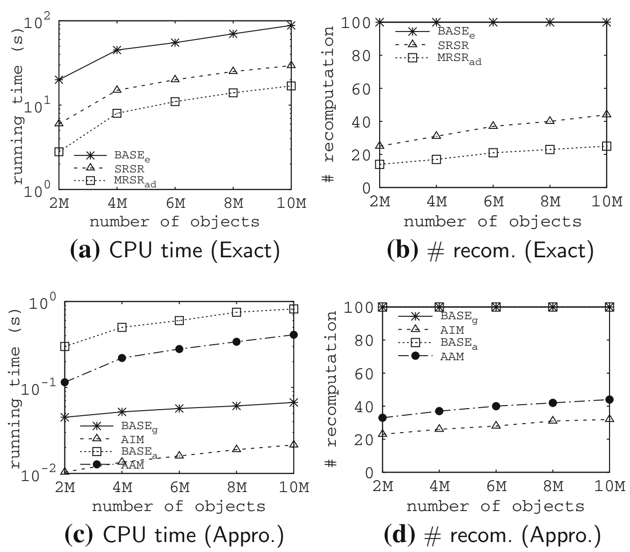


Fig. 15 Effect of data set size

runs 5 times faster than BASE$_e$, and AIM and AAM significantly outperform their corresponding baseline algorithms, and the advantage is up to 4 times. The costs of the proposed algorithms increase as the size of data sets increases, since the safe regions become smaller and recomputations become more frequent. Note that the CPU times of the two approximate algorithms (i.e., AIM and AAM) are less than 0.4 s even for 10M data objects.

Effect of α Next, we test the effect of α, which is a parameter included in the MaxMax cost function (cf. Eq. (1)). As shown in Fig. 16, the costs of MRSR$_{ad}$ and AAM increase as α increases. A possible explanation is that a larger value of α leads to smaller safe regions. The cost of AIM stays

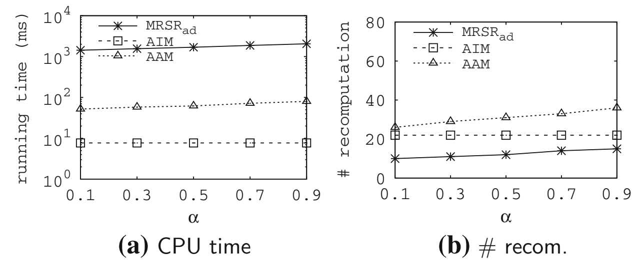


Fig. 16 Effect of α

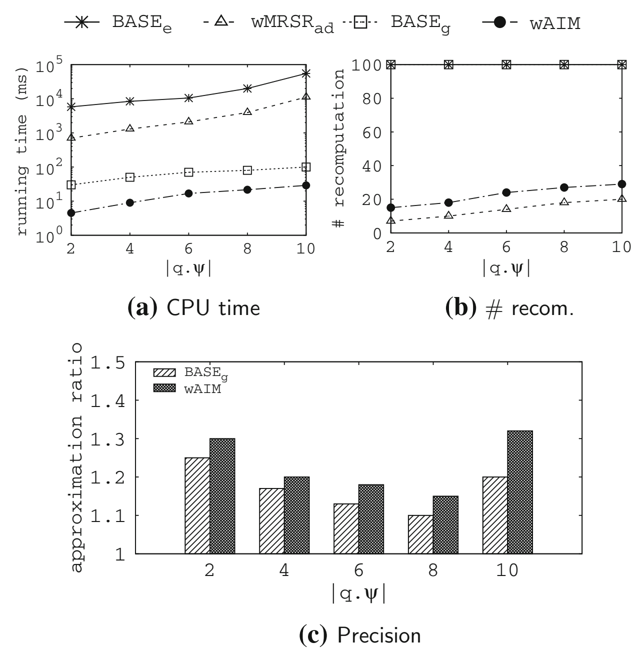
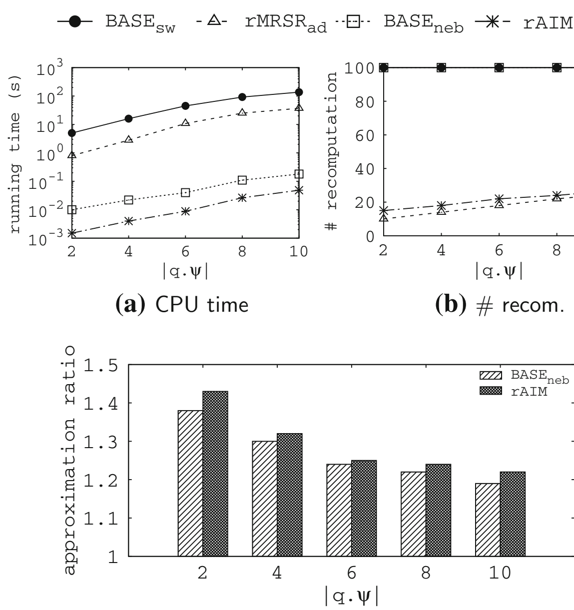


Fig. 17 Result of MCSKQ with weighted objects

unchanged. This is because AIM is used to maintain the nearest neighbor set, which is not affected by α.

MCSKQ with weighted objects We use the LA data set to study the effectiveness of our proposed algorithms on MCSKQ with weighted objects. We randomly set the weight of each object between $\frac{1}{e}$ ($e \approx 2.718$) and 1 (although the proposed algorithms can work with any weight). We still use BASE$_e$ and BASE$_g$ to denote the two baseline algorithms which invoke wMaxMax-Exact and wMaxMax-Appro [12] at every timestamp, respectively. We adapt wMaxMax-Exact to compute the top-k feasible sets in recomputation of wMRSR$_{ad}$ in a way that is similar to extending MaxMax-Exact (cf. Sect. 7.1). We report the results in Fig. 17 when varying the number of query keywords. From Fig. 17a we can see that wMRSR$_{ad}$ and wAIM perform much better than the baseline algorithms, and wAIM has the fastest runtime. Compared with the proposed algorithms used in the original MCSKQ (cf. Fig. 10a), wMRSR$_{ad}$ and wAIM have similar performances with MRSR$_{ad}$ and AIM, respectively. This confirms the flexibility of RSR and KVNS. Moreover, as



(a) CPU time **(b)** # recom.

(c) Precision

Fig. 18 Result of MCSKQ on road networks

shown in Fig. 17c, the approximation ratio of wAIM is much better than the worst-case ratio as derived in Lemma 8, that is, $1 + 2e$.

MCSKQ on road networks Next, we test the effectiveness of our proposed algorithms for MCSKQ on road networks.

We use $BASE_{sw}$ and $BASE_{neb}$ to denote the baseline exact algorithm and the baseline approximate algorithm, which invoke SW and NEB [14] at every timestamp, respectively. SW uses sliding windows technique to find a set with relatively small cost early, which can be served as a tight upper bound to prune the search space. We adapt SW to find the top-k feasible sets in recomputation of rMRSR$_{ad}$ by regarding the current k-th feasible set as the upper bound. The results are reported in Fig. 18. From Fig. 18a we can see that the CPU time costs of all the algorithms increase as the number of query keywords increases. This is because, when the number of query keywords increases, these algorithms need to retrieve more objects to cover the query keywords. Since equipped with RSR and KVNS, rMRSR$_{ad}$ and rAIM perform much better than the baseline algorithms. As we can see from Fig. 18c, rAIM has a good approximation ratio that is less than 1.5. This confirms the effectiveness of rAIM because the upper bound of the approximation ratio is 3. Compared with the proposed algorithms MRSR$_{ad}$ and AIM used in Euclidean space (cf. Fig. 12), rMRSR$_{ad}$ and rAIM spend more CPU times, because of the complicated computations of road network distance and edges and line segments (e&s) included in RSR.

8 Conclusion

We formulated the MCSKQ problem and conducted a comprehensive study. We first proposed two exact algorithms to reduce the query recomputation frequency, using the safe region technique. However, due to the high cost of query recomputation, they still lacked computation efficiency. To overcome this limitation, we proposed two approximate algorithms that compute approximate result sets in recomputation and maintain the result when the query moves, which successfully reduce the recomputation cost and hence the overall query costs. The proposed algorithms are also effective for variants of MCSKQ. We conducted a detailed cost analysis for the proposed algorithms. Empirical studies on real-world data sets and synthetic data sets demonstrate that our proposal is able to achieve a reduction of the processing time by 60–85% compared with the baseline algorithms, which confirmed our cost analysis.

In the future, we plan to study MCSKQ with moving objects and MCSKQ with user preference.

Acknowledgements This work is supported by the National Key R&D Program of China (2018YFB1003404), the National Natural Science Foundation of China (61872070, U1811261), the Fundamental Research Funds for the Central Universities (N171605001) and Liao Ning Revitalization Talents Program (XLYC1807158).

References

1. Guo, L., Shao, J., Aung, H., Tan, K.: Efficient continuous top-k spatial keyword queries on road networks. Geoinformatica **19**(1), 29–60 (2015)
2. Huang, W., Li, G., Tan, K., Feng, J.: Efficient safe-region construction for moving top-k spatial keyword queries. In: CIKM, pp. 932–941 (2012)
3. Qi, J., Zhang, R., Jensen, C., Ramamohanarao, K., He, J.: Continuous spatial query processing: a survey of safe region based techniques. ACM Comput. Surv. **51**(3), 1–39 (2018)
4. Wu, D., Yiu, M., Jensen, C., Cong, G.: Efficient continuously moving top-k spatial keyword query processing. In: ICDE, pp. 541–552 (2011)
5. Cao, X., Cong, G., Guo, T., Jensen, C., Ooi, B.: Collective spatial keyword querying. In: SIGMOD, pp. 373–384 (2011)
6. Chan, H., Long, C., Wong, R.: On generalizing collective spatial keyword queries. IEEE Trans. Knowl. Data Eng. **30**(9), 1712–1726 (2018)
7. Long, C., Wong, C., Wang, K., Fu, W.: Collective spatial keyword queries: a distance owner-driven approach. In: SIGMOD, pp. 689–700 (2013)
8. Su, S., Zhao, S., Cheng, X., Bi, R., Cao, X., Wang, J.: Group-based collective keyword querying in road networks. Inf. Process. Lett. **118**, 83–90 (2017)
9. Nutanong, S., Zhang, R., Tanin, E., Kulik, L.: Analysis and evaluation of V*-kNN: an efficient algorithm for moving kNN queries. VLDBJ **19**(3), 307–332 (2010)
10. Wang, Y., Zhang, R., Xu, C., Qi, J., Gu, Y., Yu, G.: Continuous visible k nearest neighbor query on moving objects. Inf. Syst. **44**, 1–21 (2014)

11. Ward, P., He, Z., Zhang, R., Qi, J.: Real-time continuous intersection joins over large sets of moving objects using graphic processing units. VLDBJ **23**(6), 965–985 (2014)
12. Cao, X., Cong, G., Guo, T., Jensen, C., Ooi, B.: Efficient processing of spatial group keyword queries. ACM TODS **40**(2), 1–48 (2015)
13. Chan, H., Long, C., Wong, R.: Inherent-cost aware collective spatial keyword queries. In: SSTD, pp. 357–375 (2017)
14. Gao, Y., Zhao, J., Zheng, B., Chen, G.: Efficient collective spatial keyword query processing on road networks. IEEE Trans. Intell. Transp. Syst. **17**(2), 469–480 (2016)
15. Jin, X., Shin, S., Jo, E., Lee, K.: Collective keyword query on a spatial knowledge base. IEEE Trans. Knowl. Data Eng. **31**(11), 2051–2062 (2019)
16. Zhao, S., Cheng, X., Su, S., Shuang, K.: Popularity-aware collective keyword queries in road networks. Geoinform. **21**(3), 485–518 (2017)
17. Zhang, P., Lin, H., Yao, B., Lu, D.: Level-aware collective spatial keyword queries. Inf. Sci. **378**, 194–214 (2017)
18. Shekhar, S., Liu, D.: Ccam: a connectivity-clustered access method for networks and network computations. IEEE Trans. Knowl. Data Eng. **9**(1), 102–119 (1993)
19. Gu, Y., Liu, G., Qi, J., Xu, H., Yu, G., Zhang, R.: The moving k diversified nearest neighbor query. IEEE Trans. Knowl. Data Eng. **28**(10), 2778–2792 (2016)
20. Li, C., Gu, Y., Qi, J., Yu, G., Zhang, R., Yi, W.: Processing moving $k$nn queries using influential neighbor sets. PVLDB **8**(2), 113–124 (2014)
21. Tao, Y., Papadias, D., Shen, Q.: Continuous nearest neighbor search. In: VLDB, pp. 287–298 (2002)
22. Attique, M., Cho, H., Jin, R., Chung, T.: Efficient processing of continuous reverse k nearest neighbor on moving objects in road networks. Geo-Inf **5**(12), 247 (2016)
23. Cheema, M., Zhang, W., Lin, X., Zhang, Y., Li, X.: Continuous reverse k nearest neighbors queries in Euclidean space and in spatial networks. VLDBJ **21**(1), 69–95 (2012)
24. Cheema, M., Brankovic, L., Lin, X., Zhang, W., Wang, W.: Multi-guarded safe zone: An effective technique to monitor moving circular range queries. In: ICDE, pp. 189–200 (2010)
25. Cho, H., Ryu, K., Chung, T.: An efficient algorithm for computing safe exit points of moving range queries in directed road networks. Inf. Syst. **41**, 1–19 (2014)
26. Huang, J., Huang, C.: A proxy-based approach to continuous location-based spatial queries in mobile environments. IEEE Trans. Knowl. Data Eng. **25**(2), 260–273 (2013)
27. Mahmood, A., Daghistani, A., Aly, A., Tang, M., Basalamah S., Prabhakar,S., Aref, W.: Adaptive processing of spatial-keyword data over a distributed streaming cluster. In: SIGSPATIAL, pp. 219–228 (2018)
28. Chen, B., Lv, Z., Yu, X., Liu, Y.: Sliding window top-k monitoring over distributed data streams. Data Sci. Eng. **2**(4), 289–300 (2017)
29. Wang, X., Zhang, Y., Zhang, W., Lin, X., Wang, W.: AP-tree: efficiently support location-aware publish/subscribe. VLDBJ **24**(6), 823–848 (2015)
30. Salgado, C., Cheema, M., Ali, M.: Continuous monitoring of range spatial keyword query over moving objects. World Wide Web **21**(3), 687–712 (2018)
31. Guo, L., Zhang, D., Li, G., Tan, K., Bao, Z.: Location-aware pub/sub system: when continuous moving queries meet dynamic event streams. In: SIGMOD, pp. 843–857 (2015)
32. Zheng, B., Zheng, K., Xiao, X., Su, H., Yin, H., Zhou, X., Li, G.: Keyword-aware continuous kNN query on road networks. In: ICDE, pp. 871–882 (2016)
33. Okabe, A., Boots, B., Sugihara, K., Chiu, S.: Spatial Tessellations: Concepts and Applications of Voronoi Diagrams. Wiley, London (2001)
34. Liu, C., Papadopoulou, E., Lee, D.: An output-sensitive approach for the L1/L∞ k-nearest-neighbor voronoi diagram. Algorithms ESA **1**, 70–81 (2011)
35. Mu, L.: Polygon characterization with the multiplicatively weighted voronoi diagram. Prof. Geogr. **56**(2), 223–239 (2004)
36. Kolahdouzan, M., Shahabi, C.: Voronoi-based k nearest neighbor search for spatial network databases. In: VLDB, pp. 840–851 (2004)
37. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: VLDB, pp. 802–813 (2003)
38. Chen, L., Cong, G., Cao, X., Tan, K.: Temporal spatial-keyword top-k publish/subscribe.In: ICDE, pp. 255–266 (2015)
39. Bao, J., Zheng, Y., Mokbel, M.: Location-based and preference-aware recommendation using sparse geo-social networking data. In: SIGSPATIAL, pp. 199–208 (2012)
40. Cong, G., Jensen, C., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. VLDB Endow. **2**(1), 337–348 (2009)