World Wide Web (2018) 21:687–712
DOI 10.1007/s11280-017-0488-3

Continuous monitoring of range spatial keyword query over moving objects

Chaluka Salgado[1] · Muhammad Aamir Cheema[1] · Mohammed Eunus Ali[2]

Received: 12 February 2017 / Revised: 2 June 2017 / Accepted: 3 August 2017 /
Published online: 11 August 2017
© Springer Science+Business Media, LLC 2017

Abstract In this paper, we propose an efficient solution for processing continuous range spatial keyword queries over moving spatio-textual objects (namely, *CRSK-mo* queries). Major challenges in efficient processing of CRSK-mo queries are as follows: (i) the query range is determined based on both spatial proximity and textual similarity; thus a straightforward spatial proximity based pruning of the search space is not applicable as any object far from a query location with a high textual similarity score can still be the answer (and vice versa), (ii) frequent location updates may invalidate a query result, and thus require frequent re-computing of the result set for any object updates. To address these challenges, the key idea of our approach is to exploit 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), and then use these bounds to quickly prune objects and queries through smart in-memory data structures. We conduct extensive experiments with a synthetic dataset that verify the effectiveness and efficiency of our proposed algorithm.

Keywords Spatial keyword queries · Continuous range queries · Moving objects · Safe zone · Location based services

✉ Chaluka Salgado
chaluka.salgado@monash.edu

Muhammad Aamir Cheema
aamir.cheema@monash.edu

Mohammed Eunus Ali
eunus@cse.buet.ac.bd

1 Monash University, Melbourne, Australia

2 Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh

1 Introduction

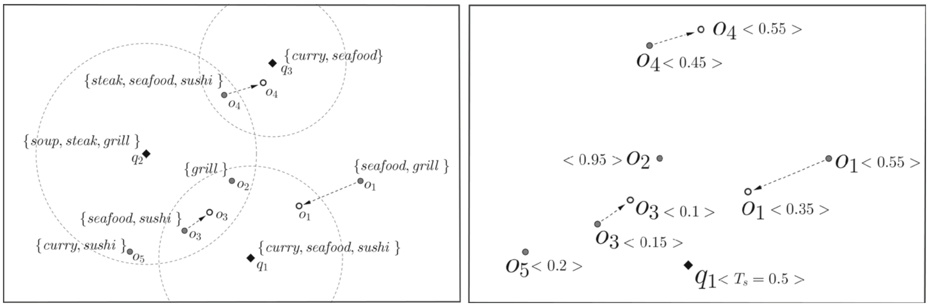
The proliferation of GPS-enabled mobile devices, and the huge popularity of location based social networking sites (LBSN, e.g., Foursquare, Yelp) have facilitated the generation of a large volume of geo-textual (or spatio-textual) datasets which form the basis of many emerging location based services (LBS). One of the important and popular forms of queries in LBS is the spatial keyword query: given a set O of spatio-textual objects where each $o \in O$ is described by its location and a set of keywords, a spatial keyword query q finds objects that meet the requirements of the query in terms of both spatial proximity and textual similarity. The spatial keyword queries have been extensively studied in different contexts that include range query, k nearest neighbour query, top-k query, and publish-subscribe query. A comprehensive study of these queries can be found in [10]. However, to the best of our knowledge, we are the first to study continuous monitoring of *moving* spatio-textual objects for thousands of continuous (long running) queries in real time. Next, we present our motivation for studying this problem.

1.1 Motivation

Consider the example of a fast food chain that wants to continuously monitor the potential customers to send them targeted advertisements and deals. Potential customers of a fast food outlet are the users that are close to it and whose preferences (keywords) match the menu of the restaurant. The fast food chain may want to continuously monitor all such potential customers to increase their sale. Similarly, a supermarket may want to monitor the people who are close to it and are looking for products sold at the supermarket. These people may be attracted by sending e-coupons or personalized deals. Spatial keyword queries are also important in other domains. For example, in an emergency scenario, hospitals could monitor the locations of health workers or volunteers, and send requests to those who are nearby and whose expertise match with the required expertise. In all of the above scenarios, we need to continuously monitor moving spatio-textual objects (e.g. customers or users) for multiple long running range queries with respect to query objects (e.g., facilities such as restaurants or hospitals). We call these queries continuous range spatial keyword queries over moving spatio-textual objects (or $CRSK$-*mo* queries). Next, we provide an example of such queries.

Example 1 Figure 1a shows three range queries q_1, q_2, and q_3 and five spatio-textual moving objects (users) o_1, o_2, o_3, o_4, and o_5. Assume that a query wants to track every user whose current location is inside the query range, and who has at least one common keyword with the query keywords. In this case, at time t_1, the $CRSK$-*mo* query returns $RS^{t_1}_{q_1} = \{o_3\}$, $RS^{t_1}_{q_2} = \{o_2, o_4\}$ and $RS^{t_1}_{q_3} = \{o_4\}$ as the result sets of q_1, q_2, and q_3, respectively. Now, at time t_2, objects o_1, o_3, o_4 move to new locations as shown using small blank circles. Thus, the query results are updated as $RS^{t_2}_{q_1} = \{o_1, o_3\}$, $RS^{t_2}_{q_2} = \{o_2\}$ and $RS^{t_2}_{q_3} = \{o_4\}$.

In the above example, we explain the concept of $CRSK$-*mo* queries by using *boolean range queries* [13, 30, 32] where an object is a result of a query if it is within a specific range and matches a keyword criteria (e.g., at least one keyword matches). A general and often preferred approach is to define the relevance of an object to a query using a relevance score that combines both spatial proximity and textual similarity between the query and object. In this case, a query user may set a *threshold score* T_s and every object with relevance score less than or equal to the given threshold score is returned as an answer. Figure 1b



(a) Boolean range (b) Spatio-textual relevance score based range

Figure 1 An example of *CRSK-mo* query

shows an example where the objects' relevance scores are shown next to each object and the movement to a new location is shown by an arrow. Thus, the results of the query q_1 for the two timestamps are $RS_{q_1}^{t_1} = \{o_3, o_4, o_5\}$, and $RS_{q_1}^{t_2} = \{o_1, o_3, o_5\}$, respectively, where the threshold score of query q_1, is set to 0.5, i.e., $T_s = 0.5$. In this paper, we study *CRSK-mo* queries by considering this general notion of relevance (for more details, see Section 2).

1.2 Challenges

Key challenges of solving $CRSK - mo$ queries are as follows: (i) the range threshold T_s is determined based on both spatial proximity and textual similarity; any object far from a query location with a high textual similarity score can still be the answer (and vice versa), and thus it is hard to prune the search space, and (ii) frequent location updates may invalidate a query result, therefore, it requires continuously maintaining the up-to-date results while minimizing the total computation cost and communication cost between clients (objects) and the server.

To address the above challenges, in this paper, we propose a client-server based comprehensive solution for monitoring continuous range spatial keyword queries over moving spatio-textual objects. Inspired by the usefulness of *safe zone based approaches* [5, 7–9, 20, 31] for monitoring other types of spatial queries, we also develop a safe zone based approach where each object is assigned an area (called safe zone) such that the object does not affect the result of any continuous query in the system as long as the object remains in its safe zone. The advantage is that the system does not need to recompute the results (reducing computation cost) and the object does not need to report its location to the server (reducing communication cost) as long as the object is inside its safe zone.

In addition to safe zone, we also maintain another region on the server side, which we call, *buffer region*. The buffer region reduces frequent safe zone computations and also ensures that the workload assigned to each client device is manageable. We propose a novel framework that elegantly handles frequent updates from objects while answering CRSK-mo queries. We propose a grid based in-memory data structure that enables us to efficiently process multiple long-running registered queries over a large number of moving spatio-textual objects in tandem with the efficient construction of safe zones and buffer regions. Our experimental study shows that our approach significantly outperforms the competitive *PCR* method for a wide range of parameters.

1.3 Contributions

Our contributions in this paper can be summarized as follows:

- To the best of our knowledge, we are the first to study continuous monitoring of moving spatio-textual objects for multiple continuous range spatial keyword queries.
- We propose a grid based in-memory data structure that elegantly handles frequent location updates while processing multiple CRSK-mo queries.
- We conduct an extensive set of experiments to show that our proposed approach outperforms the competitive approach significantly.

The rest of the paper is organized as follows. Section 2 formulates the query studied in this paper. Section 3 reviews related work. Section 4 explains the overview of the solution, while Section 5 describes the proposed algorithm. Section 6 reports the results of experimental evaluation, and Section 7 concludes the paper with a discussion of future work.

2 Problem statement

Let O be a set of spatio-textual objects (users) and Q be a set of facilities or POIs (queries). Each spatio-textual object $o \in O$ is defined as a pair $(o.\lambda, o.\psi)$, where $o.\lambda$ is the current point location of the user and $o.\psi$ is a set of keywords representing her preferences. Similarly, a query object $q \in Q$ is also defined as a pair $(q.\lambda, q.\psi)$, where $q.\lambda$ is the location of the facility and $o.\psi$ is a set of keywords representing its attributes in the form of a textual description.

The geo-textual relevance between an object and a query is defined in terms of both spatial proximity and textual similarity. Let $dist(q, o)$ be the spatial distance between query location $q.\lambda$ and object location $o.\lambda$, and $text(q, o)$ be the textual similarity between the two keyword sets $q.\psi$ and $o.\psi$. To convert the textual similarity to the textual distance, we use textual score as $S_t(q, o) = 1 - text(q, o)$, here a smaller value of $S_t(q, o)$ signifies a higher textual similarity between q and o. We assume our working space is normalized so that both spatial distance score $dist(q, o)$, and textual distance score $S_t(q, o)$ lie between 0 and 1 (inclusive). Thus, geo-textual relevance score, $score(q, o)$ of o with respect to q can be expressed as follows:

$$score(q, o) = \alpha \cdot dist(q, o) + (1 - \alpha) \cdot S_t(q, o) \tag{1}$$

Here, α is a query parameter (user-defined) that lies between 0 and 1(exclusive) to control the preference of spatial proximity over textual similarity.

We compute the spatial proximity score $dist(q, o) = 1$ if $||q.\lambda, o.\lambda|| \geq R$ where $||q.\lambda, o.\lambda||$ is the normalized euclidean distance between q and o, and R is a system defined range. If $||q.\lambda, o.\lambda|| < R$, then $dist(q, o)$ is computed as follows,

$$dist(q, o) = \frac{||q.\lambda, o.\lambda||}{R} \tag{2}$$

The intuition of using R is as follows. Assume that each object has exactly three keywords. An object o's textual distance $S_t(q, o)$ will be 0 if query q contains all keywords. Its textual distance will be $1/3, 2/3$ or 1 if q contains 2, 1 or 0 of its keywords, respectively. Now, consider the example of objects in Los Angeles (the data set used in our experiments) and assume that the maximum distance between two points in the space is $100km$. Now consider two objects o_1 and o_2 such that distance between q to o_1 is 0.1 km and distance between q to o_2 is 15 km. Their spatial similarity scores (without considering R) will be their distances from q normalized in the range 0 to 1, i.e., $dist(q, o_1) = 0.1/100 = 0.001$ and $dist(q, o_2) = 15/100 = 0.15$. Now, assume that q contains 2 out of 3 keywords of o_1 (i.e., $S_t(q, o_1) = 1/3$) and q contains all three keywords of o_2 (i.e., $S_t(q, o_2) = 0$). If $\alpha = 0.5$, their total scores will be $score(q, o_1) = 0.5 \times 0.001 + 0.5 \times 1/3 = 0.167$ and $score(q, o_2) = 0.5 \times 0.15 + 0.5 \times 0 = 0.075$. Therefore, o_2 gets a better score although its distance from q is much larger compared to the distance of o_1. In other words, the scores are biased towards textual similarity, i.e., the objects that have better textual similarity have higher chance to be the result even if they are quite far from q.

Now, consider the same example, and assume that $R = 0.2$. In this case, $dist(q, o_1) = 0.001/0.2 = 0.005$ and $dist(q, o_2) = 0.75$ and $score(q, o_1) = 0.169$ and $score(q, o_2) = 0.375$. Note that R normalizes the spatial proximity score to reduce the bias towards the textual similarity score. In short, R was introduced to address the bias towards textual similarity. Its effect is similar to setting α to a higher value.

Note that, the textual similarity (*text*) can be computed using any information retrieval model. In this paper, we use a function [19] similar to the weighted *Jaccard coefficient*, described as follows:

$$text(q, o) = \frac{\sum_{t \in q.\psi \cap o.\psi} w(t)}{w(o.\psi) = \sum_{t \in o.\psi} w(t)} \tag{3}$$

where, $w(t)$ denotes the weight of keyword t, computed by obtaining the inverted document frequency (*idf*). And $w(o.\psi)$ indicates the weighted sum of object keywords (i.e., $o.\psi$). Table 1 summarizes the notations frequently used in the rest of the paper.

Definition 1 (***Range Spatial Keyword Query (RSKQ)***) Let O be a set of spatio-textual objects and, q be a range spatial keyword query, $q = \{\lambda, \psi, \alpha, T_s\}$ where λ is the query location, ψ is the set of keywords, α the query preference factor between spatial proximity and keyword set similarity, and T_s is the range *threshold score* combining both spatial

Table 1 The summary of notations

Notation	Definition
o	a spatial textual object
q	a continuous range spatial-keyword query
$o.\lambda(q.\lambda)$	the location of object o(query q)
$o.\psi$ $(q.\psi)$	a set of keywords for object o(query q)
$o.m$	capacity of object o
$q.T_s$	threshold score of query q
$q.\alpha$	query preference factor of query q
RS_q^t	result set of query q at timestamp t
Cl_q^o	conditional circle of object o w.r.t query q
r_q^o	radius of Cl_q^o
$BR(o)$	buffer region of object o
Cl_q^{max}	largest conditional circle of query q
r_q^{max}	radius of Cl_q^{max}
Ω	default range of buffer regions

and textual factors. The query q returns a set of objects, $RS_q \subseteq O$, whose geo-textual relevance scores are less than or equal to the given threshold score T_s, i.e, $\forall o^* \in RS_q, score(q, o^*) \leq T_s$.

Definition 2 (***Continuous Range Spatial Keyword Query on Moving Objects (CRSK-mo)***) Let O be a set of *moving* spatio-textual objects, and Q be a set of long running static range spatial keyword queries, for each $q \in Q$, the continuous range spatial keyword query over moving objects (CRSK-mo) finds a set $RS_q^t \subseteq O$ of objects for *every time instance* t, where $\forall o^* \in RS_q^t, score(q, o^*) \leq T_s$.

2.1 Client-server model

We utilize the client-server paradigm, in which we have two types of client objects: facilities (static) and users (moving). The facilities (e.g., restaurants, shops, etc.) are static clients who issue queries to the server. The users are moving clients who use their GPS-enabled mobile phones to continuously track their respective locations, and send updates to the server. The latter type of clients is referred as spatio-textual moving objects in this paper. Figure 2 shows the schematic diagram of our system model where clients send their updates and issue queries to the central server, and the server is responsible for maintaining moving object and processing the queries. Finally, the server returns the result sets to the clients.

The current location of object o is represented using x-y coordinates, i.e., $(o.\lambda.x, o.\lambda.y)$. Initially, an object sends its location and preferences (keywords) as a tuple $(o.\lambda, o.\psi)$. Since objects frequently change their positions (i.e., moving), an object o sends its location update to the server as $< oID, \lambda.x_{cur}, \lambda.y_{cur} >$, where (x_{cur}, y_{cur}) is the current location of object o.

3 Related work

In this section, we review the existing techniques and indexing structures that are relevant to our work. First, we introduce some background work on continuous query processing on spatial data, and then we discuss the existing indexing structures and techniques for spatial keyword query processing, and finally, we briefly discuss the spatial keyword aware publish/subscribe systems.

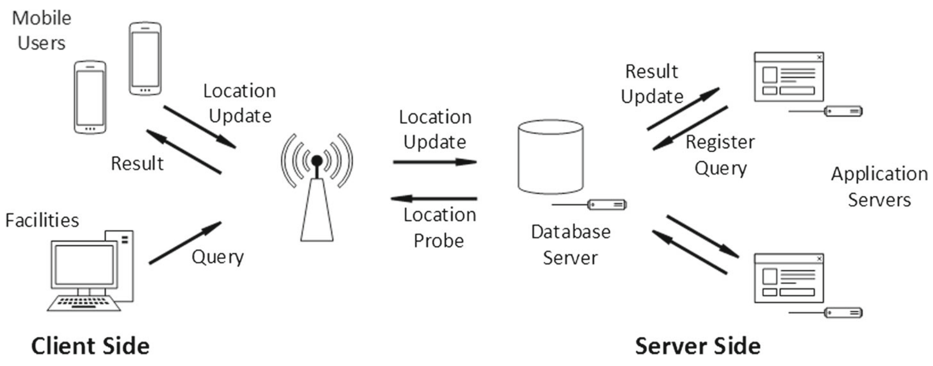


Figure 2 The System Architecture

Continuous spatial queries Continuous query processing over moving objects has been extensively studied in the spatial database domain in the context of both stationary and moving queries. To evaluate continuous static queries over moving spatial objects, Ṡaltenis [28] and Tao et al. [29] suggested a technique to index moving objects trajectories. However, maintaining the index continuously for moving objects is expensive. Prabhakar et al. [26] and Wu et al. [33] introduced a strategy to index the queries instead of the objects which significantly reduces the index maintenance cost. These techniques impose a high communication and computational overhead on the server and also affect the battery life of hand-held devices due to frequent location updates and computations. To overcome these issues, Hu et al. [18] and Prabhakar et al. [26] proposed an efficient and attractive technique called *safe zones*. The intuition behind the safe zone is to compute a region for an object depending on all the query boundaries, such that as long as the object lies within the safe zone, none of the query results is changed. Intuitively, the object does not need to send a location update to the server unless it crosses the safe zone. The safe zone technique reduces the communication and computational overhead at the server by minimizing frequent location updates. In addition, Gedik and Lui [14] and Cheema et al. [6] utilized the safe zone concept to continuously evaluate moving queries.

All the aforementioned techniques considered the spatial information but did not take into account the textual information. Therefore, these techniques cannot be used in continuous spatial keyword query processing over moving spatio-textual objects. However, we have adopted the concept of safe zone in the context of spatio-textual queries.

Spatial keyword query Spatial keyword queries have been extensively studied in different contexts that include boolean range query, boolean k- nearest neighbour (NN) query, and top-k NN query. A boolean range query [13, 30, 32] returns all the objects that match all the given query keywords and within the spatial range of the query. A boolean kNN [3, 23] query returns k objects in order of spatial proximity whose keywords match with the query keywords. A top-k NN query [12, 16, 27] returns k most relevant objects in terms of both spatial proximity and textual relevance. A comprehensive survey of different types of spatial keyword queries can be found in [10]. For efficient processing of spatial keyword queries, a plethora of indexing techniques, e.g., IR-tree [12, 13], CIDR-tree [12], and aR-tree [27] have been proposed. Since these works focus on one-time disk based query processing, these techniques are not applicable in our work where queries are evaluated continuously in which an in-memory data structure is essential.

Wu et al. [31] and Huang et al. [20] investigated *moving top-k spatial keyword queries* over stationary geo-textual objects. Wu et a. [31] proposed a technique that uses *multiplicatively weighted (MW) Voronoi* cells. MW-Voronoi cells were generated based on weighted distance concept, where the weight for each point was computed using an ad-hoc ranking function. They proposed two algorithms to optimize the safe zone computation by reducing the search space. However, the proposed technique used polygons to approximate circles and thereby could not find the exact safe zone. Huang et al. [20] introduced a method that uses *Hyperbolas* to compute safe zones. Initially, they identified the dominant zones for objects and then used those regions to compute the safe zone for a query. They also proposed some pruning techniques and utilized indexing structures to optimize the computational time of the safe zones. In these two studies, they evaluate each query against the set of spatio-textual objects since queries are moving. In contrast, we evaluate each spatio-textual object against the set of queries in order to maintain an up-to-date result set with response to the movements of the objects.

Location aware publish/subscribe queries Another type of spatial keyword query that is closely related to our problem is the location aware publish/subscribe queries (e.g., [11, 15, 17, 19, 22]). These queries report geo-tagged event notification to the relevant subscribers, where the relevance is measured either by boolean matching or similarity based method. Guo et al. [17] studied the problem of efficient processing of continuous moving range queries over dynamic event streams. They propose an efficient index called BEQ-tree to support spatial subcription matching. Elaps updates the moving subscriber with events within the given spatial range and also match with the given boolean expression. Thus, their work cannot be extended to support our problem since our range is a combination of spatial and textual score. Moreover, the similarity based methods are used to address the top-*k* publish/subscribe problem where objects are ranked according to the spatial and textual similarity scores. Chen et al. [11] addressed a problem that takes into account the spatial proximity, the textual relevance and also the object recency in which the score of the object decays as the time passes. Hu et al. [19] used prefix filtering and spatial pruning techniques to address a problem where only the events which are within the pre-given similarity threshold are returned to the subscriber. However, these techniques are different from ours since we focus on moving spatio-textual objects.

4 Solution overview

In this section, we present a comprehensive solution for monitoring continuous range spatial keyword queries over moving objects. Processing continuous queries over moving objects is more challenging since a slight movement of an object may invalidate a query result. Thus, it requires monitoring the locations of the objects and maintaining the results continuously as the objects move. To address this challenge, we utilize the concept of *safe zones* [2, 18, 21, 26]. The safe zone of an object is an area such that as long as the object remains inside this area, it does not affect the result of any query. Hence, the object does not need to send location updates to the server unless it leaves the safe zone. Thus, the safe zone based approach reduces both the query processing cost and the communication cost between clients and the server.

In CRSK-mo query processing, the query range is determined based on both spatial proximity and textual similarity; any object far from a query location with a high textual similarity score can still be an answer for the query (and vice versa). The key idea of our approach is to exploit the spatial and textual upper bounds between queries and objects to form safe zones for each object. We also introduce the concept of buffer regions, which is maintained in the server side to avoid frequent re-computations of safe zones. Moreover, we utilize these bounds to quickly prune queries through efficient in-memory data structures. We describe our solution in the following subsections.

Section 4.1 presents the concept of safe zones that form the basis of our algorithm. Section 4.2 presents the pruning rules based on spatial proximity and textual similarity that are used by our algorithm to prune the search space.

4.1 Safe zone of an object

In a range spatial keyword query, an object o is a result for a given query q when the spatio-textual relevance score of the object is less than or equal to the given query threshold score, i.e., $score(q, o) \leq T_s$. To continuously monitor an object for a registered query, we need to essentially monitor the corresponding inequality over the time. Hence, we have the following lemma formalizing it.

Lemma 1 *An object $o \in O$ is a result of query q (i.e., $o \in RS_q$) iff $dist(q, o) \leq \frac{T_s}{\alpha} - \frac{(1-\alpha)}{\alpha} \cdot S_t(q, o)$.*

Proof Let q be a range spatial keyword query and an object o be one of the results of the query q, i.e., $o \in RS_q$. To satisfy the query condition, the object o must follow the condition, $score(q, o) \leq T_s$ as depicted in the Definition 1. Thus, we can rewrite (1) as follows: $\alpha \cdot dist(q, o) + (1-\alpha) \cdot S_t(q, o) \leq T_s$. Hence, $dist(q, o) \leq \frac{T_s}{\alpha} - \frac{(1-\alpha)}{\alpha} \cdot S_t(q, o)$. $\square$

Based on the above lemma, we define a circle called *conditional circle*, Cl_q^o, centred at the query location $q.\lambda$ with the radius of r_q^o, where the radius is defined as follows:

$$r_q^o = \frac{T_s}{\alpha} - \frac{(1-\alpha)}{\alpha} \cdot S_t(q, o) \tag{4}$$

Intuitively, a conditional circle, Cl_q^o, is *a spatial area* such that object o does not affect the result of the query q as long as o does not enter or leave the area. We identify this area as the *conditional area*. Figure 3a shows an example of the conditional area (shaded in gray) for object o when the object resides inside (i.e $dist(q, o) \leq r_q^o$) the conditional circle, i.e., Cl_q^o. In this case o remains as a result of q as long as it resides inside the cirlce. Otherwise, the object o is outside the Cl_q^o, then $o \notin RS_q$ and the conditional area is the shaded area as shown in Figure 3b.

Intuitively, the safe zone of an object involving multiple queries is constructed by taking the intersection of the conditional areas of the object with respect to all the queries. For example, Figure 4 shows conditional circles of object o_1 and o_2 for queries q_1, q_2, q_3 and q_4. Since o_1 only lies inside the $Cl_{q_2}^{o_1}$ and $Cl_{q_3}^{o_1}$, the safe zone of the object o_1 is the shaded area as shown in Figure 4a. As long as the object o_1 resides inside this area, it does not affect the results of any query. The object o_1 can determine whether it is inside the safe zone by checking whether it is inside the two conditional circles $Cl_{q_2}^{o_1}$ and $Cl_{q_3}^{o_1}$.

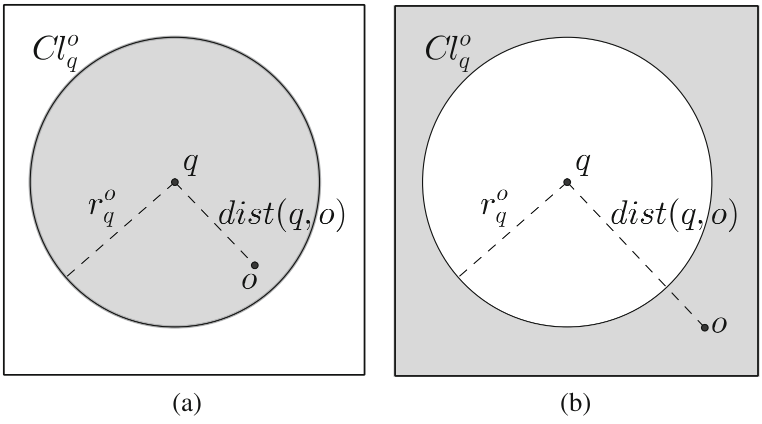


Figure 3 Example of conditional areas

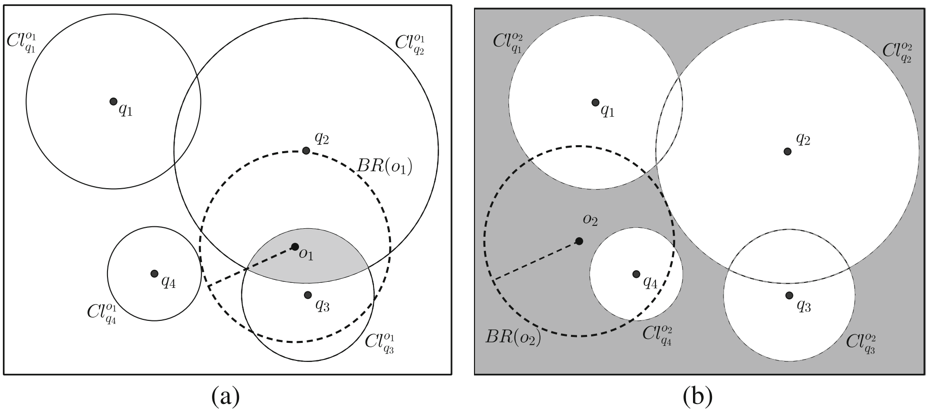


Figure 4 Buffer regions for (**a**) object o_1, (**b**) object o_2

The number of conditional circle boundaries that an object can monitor entirely depends on the computational capability of the object (i.e., client device). An object with low computational capability may result an overwhelming workload on the processor and short battery life, if the safe zone assigned to that object involves a large number of conditional circles. For example, assume that both objects o_1 and o_2 have the same set of keywords, i.e., conditional circles for each query is identical for both objects because the textual similarity is same. Figure 4b shows the safe zone of object o_2 shaded in gray, which is outside of all the conditional circles. In such a scenario, the object may need to monitor a large number of conditional circles, which is computationally expensive.

To address this problem, we introduce a concept called *Buffer Region (BR)*, to support clients' mobile devices with heterogeneous computational capabilities. Based on the computational capability of each registered object, the server assigns a value called *capacity* (denoted by m) which is the maximum number of conditional circles that particular object can monitor at a time. Thus, the capacity of the object is used to bound the number of conditional circles involved in constructing the safe zone of that object. Thereby, each device is assigned with a reasonable computational workload.

Definition 3 (***Buffer Region (BR)***) Let $o \in O$ be a spatio-textual object, where $o = \{\lambda, \psi, m\}$. The Buffer region is a circle centered at $o.\lambda$ and the radius is the Euclidean distance from $o.\lambda$ to the $m - 1^{th}$ nearest conditional circle. We denote the buffer region of the object o by $BR(o)$.

The buffer region of an object includes nearest $m-1$ conditional circles ensuring that the number of conditional circles involved in constructing the safe zone of the object does not exceed the capacity of the object. Figure 4 shows the buffer regions (i.e., dotted circle) and the safe zones (i.e., shaded area) of object o_1 and o_2. After the buffer region is constructed, it is stored in the server and the safe zone is sent to the client device of the particular object. For example, assume o_1 is an object in O, where $o_1.m = 4$. Then the buffer region of object o_1 involves three (i.e., $m - 1$) conditional circles (see Figure 4a). Thus, o_1 will monitor four circle boundaries including the buffer region boundary. Hence, the radius of the buffer region is $dist(o_1, Cl_{q4}^{o_1})$. Figure 4b shows the buffer region for object o_2 assuming its capacity is also four. Thus, it reduces the number of conditional circles that object o_2 has to monitor to check whether it is inside the safe zone. Note that the buffer region concept reduces the frequent safe zone computations and ensures the workload assigned to each client device is manageable.

4.2 Pruning rules

Processing CRSK-mo queries involves computing the results of the queries and constructing the buffer regions for each object to reduce the computational and communication overhead. Naively, for each object, we can compute conditional circles for all the queries to determine the affected queries (i.e., the queries for which the object is a result) and also to construct the buffer regions. Since the number of objects and queries are large in numbers, this naive approach is highly inefficient. To avoid this limitation, in this section, we introduce two simple pruning rules based on the bounds derived from spatial proximity and textual similarity between an object and a query. Using these pruning rules we filter a large number of irrelevant queries and obtain a set of candidate queries. Then we compute conditional circles for these candidate queries to identify the queries for which the object is a result.

According to (4), the radius of the conditional circle depends on threshold score, textual relevance, and preference parameter(α). Since the textual relevance is the only value that can vary from one object to another object, we can obtain an upper bound for the radius of the conditional circle when we set the textual relevance as zero (i.e., $S_t(q, o) = 0$). Thus, we define Cl_q^{max} as the *largest conditional circle* of query q, where the radius r_q^{max} can be defined as follows,

$$r_q^{max} = \frac{T_s}{\alpha} \tag{5}$$

Lemma 2 *(**Pruning Rule 1**) Let an object $o \in O$ be outside the Cl_q^{max} of query q, then object o cannot be a result for the query q.*

Proof Let an object o be a result of query q. According to Lemma 1, $dist(q, o) \leq r_q^o$. Hence, $r_q^o \leq r_q^{max}$, and $dist(q, o) \leq r_q^{max}$. It concludes that object o is not a result of the query if $dist(q, o) > r_q^{max}$. □

In line with Pruning Rule 1, if the object o is outside the Cl_q^{max}, then the query q can be pruned. Otherwise, the conditional circle of object o for query q, i.e., Cl_q^o is computed to verify whether the object is a result of the query (according to Lemma 1). Figure 5 shows an example for Pruning Rule 1. The solid circles depict the largest conditional circles of queries q_1, q_2 and q_3 while dotted circles depict the conditional circles of object o_1 for each query. Since object o_1 is inside the $Cl_{q_1}^{max}$ and $Cl_{q_2}^{max}$, queries q_1 and q_2 are identified as candidates while query q_3 is filtered out. Since o_1 is inside the $Cl_{q_1}^{o_1}$, object o_1 can only be a result for q_1.

However, Pruning Rule 1 may include a large number of candidate queries since it only considers the spatial proximity. So we present our next pruning rule that exploits textual relevance to filter the irrelevant queries. Next, we determine an upper bound (denoted by $maxT$) for the textual similarity between an object and a query as follows.

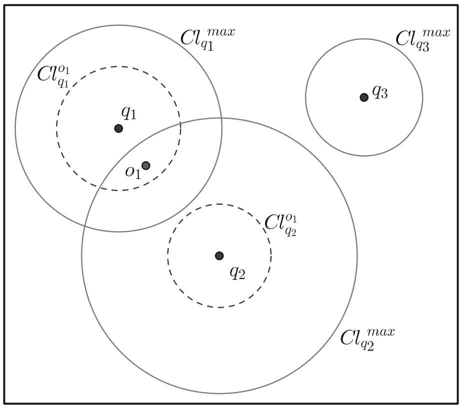


Figure 5 An example for pruning rules

From the scoring function (1), if an object o is a result, then $\alpha \cdot dist(q, o) + (1 - \alpha) \cdot S_t(q, o) \leq T_s$. Hence, when the $dist(q, o) = 0$, (1) can be rewritten as follows.

$$(1 - \alpha) \cdot S_t(q, o) \leq T_s$$
$$S_t(q, o) \leq \frac{T_s}{(1 - \alpha)}$$

Thus, the $maxT_q$ can be expressed as follows.

$$maxT_q = \frac{T_s}{(1 - \alpha)} \tag{6}$$

Lemma 3 *(**Pruning Rule 2**) An object o has potential of becoming a result of query q if* $S_t(q, o) \leq maxT_q$.

Proof Let o be an object with $S_t(q, o) > maxT_q$. By (6), $S_t(q, o) > \frac{T_s}{(1-\alpha)}$. Hence, we have $\frac{T_s}{\alpha} - \frac{(1-\alpha)}{\alpha} \cdot S_t(q, o) < 0$. By (4), $r_q^o < 0$ concludes that object o can never be a result of query q. □

If an object does not satisfy Pruning Rule 2 for a query, then that object can never be a result for the query as the conditional circle of that object does not exist (i.e., $r_q^o < 0$). Thus, we filter those queries in order to reduce the search space. Consider the previous example (see Figure 5), if $S_t(q_2, o_1) > maxT_{q2}$ then q_2 will be filtered out even though the Cl_{q2}^{max} overlaps with object o_1.

5 Algorithm

In this section, we discuss our algorithm for processing CRSK-mo queries. First, we present the server-side query processing framework, and then we propose a filter-verification algorithm that constructs buffer regions for each object while computing the result sets for each query. After that, we discuss continuous monitoring of range spatial keyword queries with respect to location updates of the objects. Finally, we present client-side processing of the system.

5.1 The framework

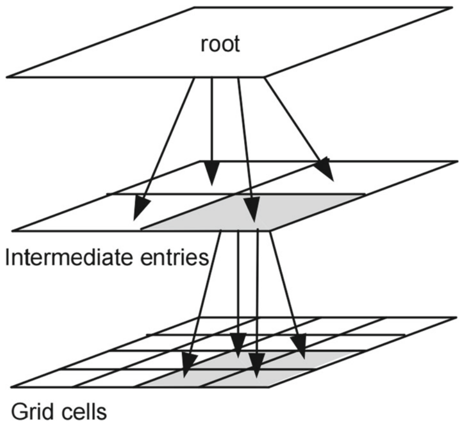
We use a *gird-based index structure* to index the CRSK-mo queries. We prefer grid-based index over the other index structures like an R-tree as it supports frequent location updates and also it is usually preferred in continuous query processing [24, 25, 34]. In our index, the data space is partitioned into $2^n \times 2^n$ grid cells (where $n \geq 0$) as shown in Figure 6. To access a particular cell quickly, we consider the grid as a conceptual tree as used in [4]. The root of the conceptual tree is a rectangle that covers the whole work space (i.e., all the cells). The root cell is divided into four equal grid cells that represent the next level of the tree. The process continues until each entry of the leaf level represents one grid cell.

Since the grid-tree is a conceptual visualization of the grid, the root entry and intermediate entries do not physically exist. So that the information is stored only in leaf level grid cells. Hence, in each grid cell, we store all the queries whose Cl_q^{max} circles overlap with the particular cell. So that when an object lies inside the cell, we can filter out all the other queries according to Pruning Rule 1. To efficiently access the queries based on keywords, instead of using a flat list we use an *inverted list* (i.e., $iQList$) to store these overlapping queries. Figure 7 shows our grid based index structure that consist of inverted lists at each grid cell. In the inverted index, for each keyword k_i, we store the queries whose description contains k_i. Accordingly, each keyword contains a posting (query) list ordered by the query id. Thus, we access the inverted list by using *document at a time* access method. Thereby, we consider only the queries contains at least one matching keyword with the objects. Moreover, by using the inverted index and Pruning Rule 2, we obtain the candidate set more efficiently.

5.2 CRSK-mo processing

We assume that all registered queries are indexed using our grid-based index structure, and objects arrive into the system in a stream-like fashion. As soon as an object arrives, it is

Figure 6 Conceptual grid-tree of a 4 × 4 grid [4]



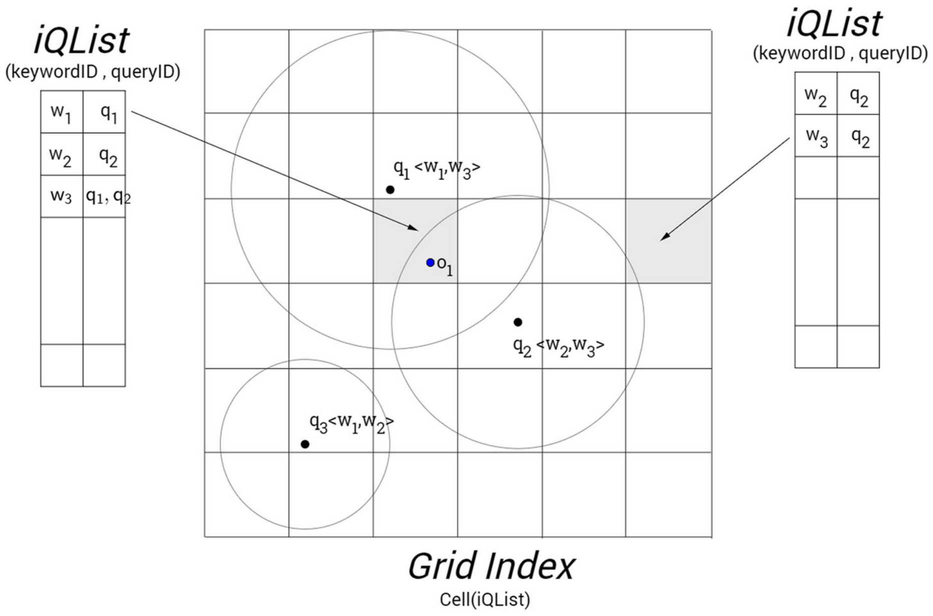


Figure 7 The index structure

immediately evaluated by our algorithm to see whether it can affect any query result. At the same time, our algorithm determines the buffer region (BR) for the object in tandem with the query evaluation. The buffer region is stored in the server and the safe zone is sent to the device of the corresponding object. Finally, query results are reported to the respective query client.

Our approach consists of two phases: filtering phase and verification phase. In the filtering phase, all the queries whose Cl_q^{max} do not overlap with the object location (Pruning Rule 1) and the queries whose textual relevance is greater than its $maxT_q$ (Pruning Rule 2) are filtered out. Then, in the verification phase, conditional circles for each candidate query are computed and the result sets of candidate queries are updated. Finally, the buffer region and the safe zone of the object are constructed.

Algorithm 1 shows the pseudocode of our algorithm. The algorithm takes an object o, and grid-based index G as input, and updates the result sets of the queries and returns the buffer region and safe zone of the object o. We start traversing the conceptual grid-tree from the root. The root entry is first inserted into the priority queue, Q_p. The elements in the priority queue are maintained in increasing order of their minimum Euclidean distance from the object. If the dequeued element is an intermediate cell and satisfies a system defined default range denoted by Ω (later we explain the intuition of this value), then we insert its children into the queue, where $mindist(chlidCell, o)$ is the key (Lines 5-7). If the dequeued element is a cell, we use the inverted list $iQList$ of the cell to select a candidate list of queries $cQList$, and for each $q \in cQList$, we compute the conditional circle Cl_q^o, and finally, the object o is inserted into the result set for the query q if the object falls inside the conditional circle Cl_q^o. Note that, when we process the inverted list of each cell, we record the queries already processed in order to avoid the redundant access.

Algorithm 1 RSKQInit(o)

```
Data: Object $o$, Grid-index $G$
Result: $RS_q$ for all $q \in Q$, $BR(o)$
$Q_p$.Enqueue($G.root$,0)
while $Q_p$ NOT Empty do
    $element \leftarrow Q_p.Dequeue()$
    if $element$ is an intermediate cell then
        foreach $childCell \in element$ do
            if $mindist(childCell, o) \leq \Omega$ then
                $Q_p.Enqueue(childCell, mindist(childCell, o))$
            end
        end
    else if $element$ is a cell then
        $cQList = getCandidateList(o, iQList)$
        foreach $q \in cQList$ do
            if $S_t(q, o) \leq maxT_q$ then
                Compute $Cl_q^o$ ;                // compute conditional circle
                Update $RS_q$ ;                        // update query answer
                if $mindist(Cl_q^o, o) \leq \Omega$ then
                    $Q_p.Enqueue(q, mindist(Cl_q^o, o))$
                end
            end
        end
    else
        $o.CLList.add(element, element.key)$ ;     // add to candidate set
        if $sizeof(o.CLList) = m - 1$ then
            break;
        end
    end
end
$BR(o) \leftarrow computeBR(o.CLList)$ ;          // computing buffer region
```

To compute the buffer region in tandem with the CRSK-mo query processing, we continue inserting the conditional circle into the priority queue, where $mindist(Cl_q^o, o)$ is used as the key. If the dequeued item is a Cl_q^o then it is added to the list maintaining conditional circles for computing the buffer region. When this list size becomes $m - 1$, we stop the process since we have sufficient Cl_q^os to construct the buffer region of object o, $BR(o)$ (Lines 22-24). Note that, it may happen that an object has less than $m - 1$ nearby queries so that it is required to traverse a wider area to compute its buffer region. In such a scenario, we take a system defined default range for the buffer region, denoted by Ω, that bounds the traversal area. From the conditional circle list, $CLList$ of object o, we compute the buffer region of the object, $BR(o)$ in Line 28. Thus, the safe zone of the object o is also determined while the buffer region is generated.

5.3 Continuous monitoring

Since the objects may frequently update their positions, we need to update the results of all the queries. In this section, we discuss our approach for handling frequent location updates of moving objects for processing CRSK-mo queries. When an object o sends its location to the server, the server performs the following steps to update the results (see Algorithm 2). First, the server checks whether the new location is inside the current $BR(o)$, if it is true, then it checks against each conditional circle that forms the $BR(o)$ to identify which queries have been affected by this location update. Then the affected queries are updated accordingly. If the new reported location is outside the current $BR(o)$, a new buffer region needs to be computed using Algorithm 1.

Algorithm 2 Continuous monitoring

```
   Data: Location Update $< o, \lambda_{new}, \lambda_{old} >$
   Result: Update $RS_q$ for all $q \in Q$, $BR(o)$
   // determining the affected queries
 1 if Object $o.\lambda_{new}$ is inside $BR(o)$ then
 2     foreach $Cl_q^o \in o.CLList$ do
           // Update the query results
 3         if $o.\lambda_{old}$ is outside $Cl_q^o$ AND $o.\lambda_{new}$ is inside $Cl_q^o$ then
 4             Insert $o$ into $RS_q$
 5         else if $o.\lambda_{old}$ is inside $Cl_q^o$ AND $o.\lambda_{new}$ is outside $Cl_q^o$ then
 6             Delete $o$ from $RS_q$
 7     end
 8 else
 9     $BR(o) \leftarrow computeBR()$
10 end
```

5.4 Client side

In our system, we assume that the client tracks its own location and nearby conditional circles. Since the client has a limited computational capability and wants to be a part of a limited number of nearby conditional circles, each client is assigned with a capacity (i.e m) to limit the number of conditional circles that involve in constructing the safe zone. The client initially sends its current point location and a set of keywords to the server. Then the server evaluates the object against all registered queries, and sends the safe zone to the object. Subsequently , the client sends its location update to the server when it leaves the current safe zone and then the server sends back a new safe zone.

6 Experimental evaluation

In this section, we evaluate the performance of our algorithm (Our) by comparing with two competitive algorithms. Section 6.1 explains the PCR approach. Section 6.2 introduces the parameters and the settings we used in our experiments while Section 6.3 describes how the default parameters were determined. Section 6.4 compares our algorithm with a spatial filtering based algorithm. Section 6.5 presents a detailed discussion on empirical studies.

6.1 Pre-Circular Range(PCR) approach

The straightforward approach for processing CSRK-mo queries involves evaluating each incoming object updates against all the registered queries, which is very expensive in terms of computational and communication overhead. Thus, we develop a competitive approach called Pre-Circular Range(PCR) to compare with our technique. In the PCR approach, for each object, we first identify a set of queries that can be affected by the movement of the object for a certain period of future time. We set a circular area, $Cir(o)$ around the object where the object can belong in a defined period of time. Thus, we first identify a set of queries whose results can be affected by the movement of the object within this circular area. In general, there can be four categories of queries: (i) a query whose Cl_q^o is completely inside the $Cir(o)$, (ii) a query whose Cl_q^o partially overlaps with $Cir(o)$, (iii) a query whose Cl_q^o is completely outside $Cir(o)$, and (iv) a query whose Cl_q^o completely contains $Cir(o)$. Naturally, the movement of object o within $Cir(o)$ only affects the queries, say $AQ(o)$, that fall under category (i) and (ii), as the boundaries of these Cl_q^os can be crossed by the object. Then we select the nearest $m-1$ queries from $AQ(o)$ to construct the $BR(o)$. If the object goes outside $BR(o)$, we need to assign a new $Cir(o)$ and repeat the above procedure. The communication between the client and the server remains similar to our approach where an object sends a location update to the server as it leaves the safe zones and so on.

6.2 Experiment settings

The experiments were conducted on a dataset which was generated as follows. We used a real world dataset(from Yelp) that contains check-in data in Los Angeles to extract the POIs. The keywords for POIs were collected from the descriptions of the relevant user check-ins. Thus, the trajectories of the moving objects were generated using the brinkhoff data generator [1] based on the road network of the Los Angeles. Then we selected a set of POIs for each object by taking the nearest POI to the object trajectory at different timestamps assuming the users checked-in to those places. Finally, we obtained the keywords for each object (user) by randomly selecting keywords from the checked-in POIs.

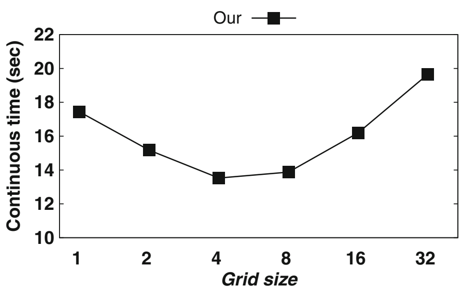
We have varied a wide range of parameters to test the supremacy of our approach over the PCR approach in a wide variety of real world settings. The details about the parameter values are given in Table 2. All the experiments were conducted in an Intel processor of 3.30GHz and 4GB of RAM, running Linux Ubuntu. We used C++ to implement all the algorithms and used in-memory setting by adopting the grid index structure since continuous result computation is essential.

We have measured the query processing time on the server side in two different metric: *initial time* and the *continuous monitoring time*. The initial time is the time spent to compute results for all the queries and construct safe zones for all the objects for the first time. Since all the queries are continuously monitored for 100 timestamps, in each timestamp server needs to maintain an up-to-date result set for all the queries as it receives location updates. In which server updates the query results, compute new buffer regions and send new safe

Table 2 The parameters used for experiments

Parameter	Default	Range
Threshold score	0.5	0.1–0.9
Preference parameter	0.5	0.1–0.9
Keywords per object	5	2–10
Keywords per query	15	10–30
Number of queries	10K	5–20K
Number of objects	100K	50–200K
Speed	medium	slow, medium, fast

Figure 8 Effect of grid size



zones to the particular objects. The continuous monitoring time sums up the time consumed by the server to maintain an up-to-date result set for the duration of 100 timestamps. In Section 6.5, we use the term *continuous time* to represent the *continuous monitoring time* for brevity.

6.3 Default parameters

In this section, we describe how the default parameters were determined to obtain the best performance of the proposed framework for all simulations.

6.3.1 Grid size

We conducted experiments to study the effect of the grid size. Figure 8 shows the effect of grid size where we change the grid size from 1×1 to 32×32 and report the continuous time. The performance degrades if grid size is too small or too large. This is because if grid cells are too large then the number of queries in each cell (and the size of inverted index) increases resulting in a poor performance. On the other hand, when the grid cells are too small, the algorithm needs to access more cells of the grid to compute the safe zone resulting in a higher computation time. Based on these experiments, we chose 4×4 as our default grid size in the experiments.

6.3.2 Omega

In Figure 9a, we conducted experiments by varying the default range (i.e., Ω) and study its effect on the total cost at the server side and the total cost at the client side. Figure 9a shows

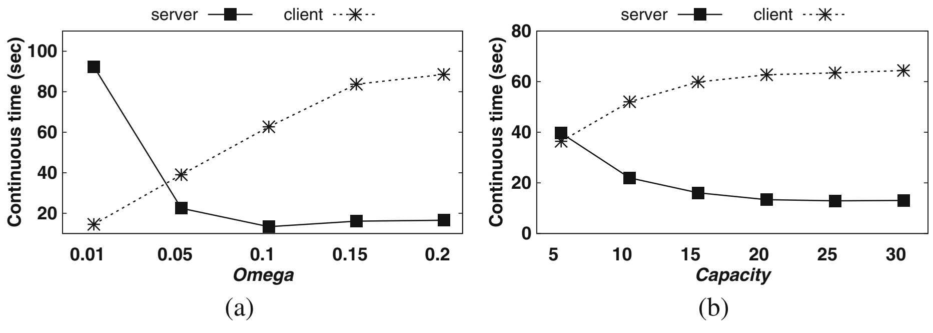


Figure 9 Varying Ω and m

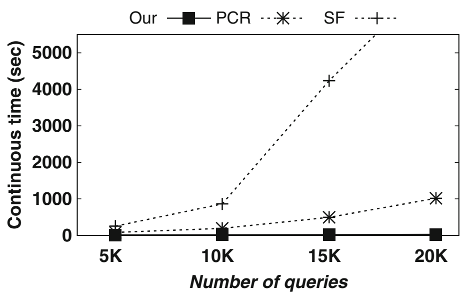


Figure 10 The performance of our approach, PCR and SF

that the total cost at server side reduces as Ω increases. This is because, as Ω increases, the buffer region size increases which results in requiring to recompute the buffer regions fewer times. In contrast, the computation cost at the client side increases as Ω increases because the area that the client device needs to monitor becomes larger. In our experiments, the default value of Ω is set to 0.1 with an aim to minimize the total cost at the server side.

6.3.3 *Capacity*

Recall that the capacity of an object(i.e., m) is decided based on the computational power of each client device (i.e., object) because the system consists of objects with heterogeneous computational capabilities. In Figure 9b, we study the effect of capacity on the total computation cost on the server and the total computation cost on all client devices for all 100 timestamps.

Figure 9b shows that the total cost on server side is reduced as the capacity increases. This is because the size of buffer region increases as the capacity increases and, as a result, the server needs to update the buffer regions fewer times. In contrast, the total cost on client devices increases as the capacity increases. This is because, to check whether a client is inside its safe zone or not, it needs to check its location against m circles and this cost increases as m increases. Therefore, there is a trade off in choosing a suitable value of m. In this paper, we choose $m = 20$ to optimize the total cost at the server side. In real world scenarios, the client devices may be asked for their preferred values of m based on their computational capabilities.

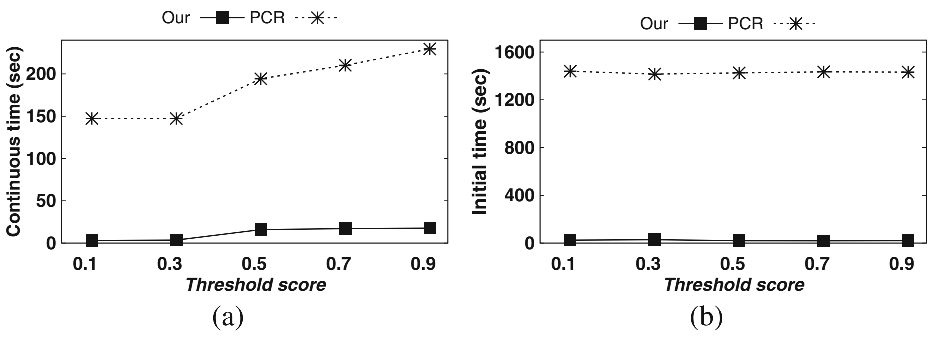


Figure 11 Varying threshold score

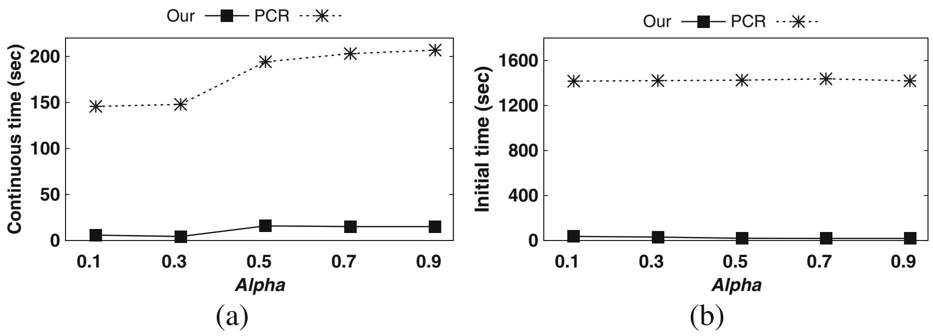


Figure 12 Varying alpha

6.4 Comparison with spatial filtering based approaches

Even though a simple spatial filtering based approach (e.g., range queries) may not work, we designed another competitor that first filters based on a particular range ρ and then retrieves the results among the filtered queries. The range ρ is set assuming the maximum possible textual similarity for each object, i.e., ρ is set such that an object o which has distance greater than ρ from q cannot be an answer even if it has maximum textual relevance. We call this approach spatial filtering (SF). This approach first applies spatial filtering based on distance ρ and then processes the candidates within the range to determine if they are the results or not.

Figure 10 compares the performance of our approach, PCR and SF methods for different number of queries. Our algorithms and PCR both outperform SF approach and scale much better with the increase in the number of queries. The performance of SF severely deteriorates as the number of queries increases mainly because more queries are found in the filtering range ρ and require verification. Since the performance of SF is comparatively much worse than PCR, we compare our algorithm only with PCR in the forthcoming experiments.

6.5 Performance evaluation

In this section, we present the experimental results of proposed algorithm and compare our approach with the pre-circular range(PCR) approach by varying different parameters.

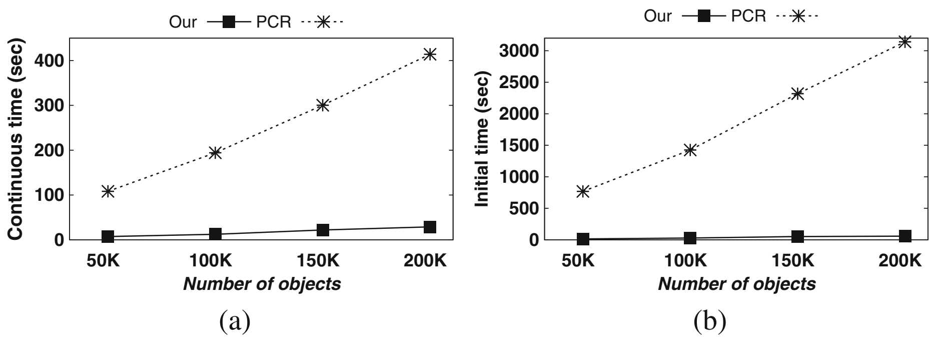


Figure 13 Varying the number of objects

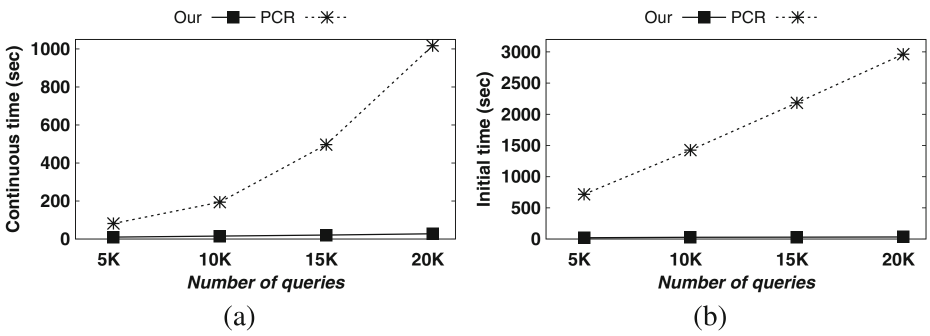


Figure 14 Varying the number of queries

6.5.1 *Effect of query parameters*

First, we evaluate the performance of our algorithm by varying the threshold score from 0.1 to 0.9 . As the Figure 11 shows, our algorithm significantly outperforms PCR approach in terms of continuous time and initial time due to the efficient filtering techniques. With the increase of threshold score value, the continuous time shows an increasing trend for both algorithms as higher threshold scores incur large result sets.

Moreover, we varied the value of alpha from 0.1 to 0.9. Figure 12 shows the results. It can be clearly seen that our algorithm performs better compared to PCR when the alpha is increased. Furthermore, the continuous time of our algorithm is approximately 12 times faster than PCR while the initial time of our algorithm outperforms PCR in two orders of magnitude for both settings.

6.5.2 *Scalability*

In this experiment, we evaluate the scalability of our algorithm in terms of continuous time. First, we scale the number of objects from 50K to 200K. Figure 13 shows the performance of both algorithms decrease as the number of objects is increased. But our algorithm performs much better in all cases due to the efficient pruning techniques. PCR performs 10 times slower than our algorithm when the algorithms run with 200K objects. Moreover, the initial time of our algorithm is much smaller compared to PCR.

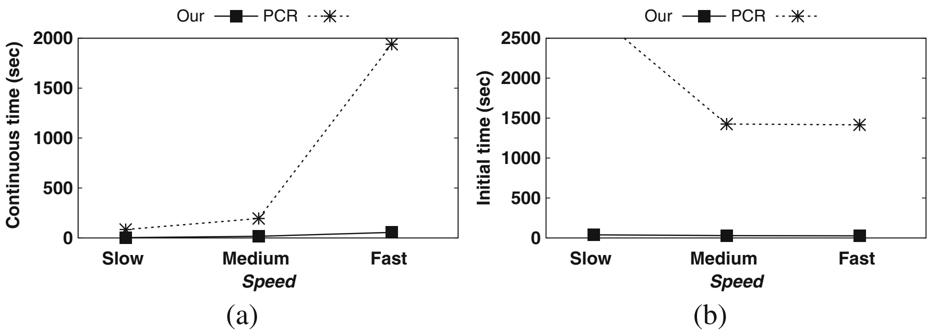


Figure 15 Varying the speed

Figure 16 Varying timestamps

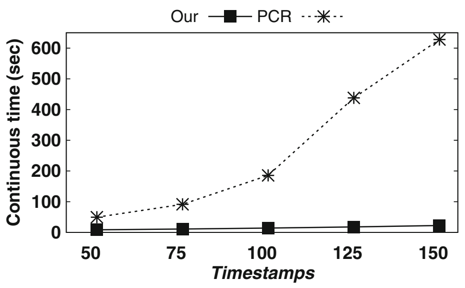


Figure 14 studies the performance of our algorithm with respect to the number of queries. We scaled the number of queries from 5K to 20K. Obviously, the continuous time of the both algorithms increases as the the number of queries increases. Note that, our algorithms performs much better compared to PCR due to the effective pruning techniques. Moreover, our algorithm performs approximately 40 times faster when the number of queries is 20K and also the initial time of our algorithm outperforms PCR in two orders of magnitude.

6.5.3 *Varying speed*

In this experiment, we study the effect of the speed of the objects on the performance of our algorithm. Figure 15 shows an increasing trend in continuous times for both algorithms. This happens because the probability of an object leaving its buffer region is proportional to the moving speed of the object. Thus, when the speed increases, the performance starts to decrease as the number of times the server regenerates the buffer regions increases. The performance of PCR decreases dramatically since its computation cost of regenerating a buffer region is really high. Moreover, our algorithm performs 40 times faster than PCR when the objects move fast.

6.5.4 *Varying timestamps*

In Figure 16, we vary the monitoring time interval from 50 to 200 timestamps and study its effect on both approaches. As shown in Figure 16, our approach outperforms PCR and scales better. The performance of the PCR degrades drastically since the number of times the buffer regions are generated increases as the size of the time interval is increased.

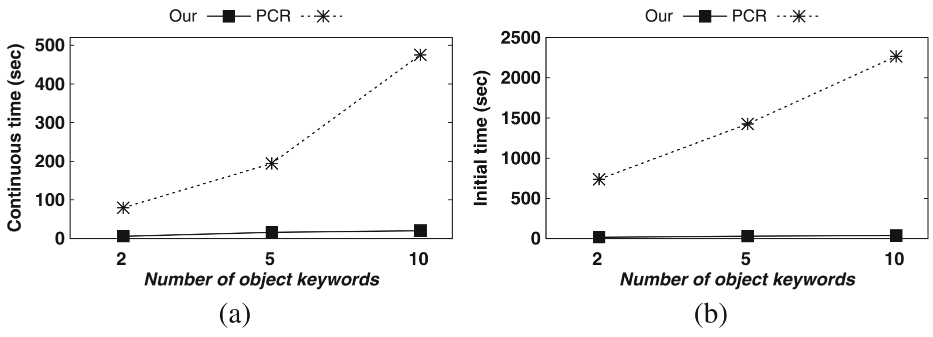


Figure 17 Varying the number of object keywords

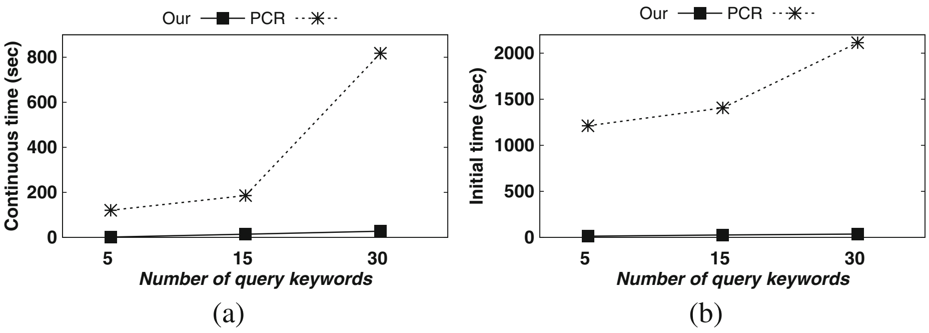


Figure 18 Varying the number of query keywords

6.5.5 *Varying number of keywords*

Figures 17 and 18 illustrate the effect of number of keywords on the performance of the algorithms. Both algorithms present an increasing trend as the number of keyword is increased. Our algorithm performs much better compared to PCR since our algorithm uses efficient pruning rules and inverted indices to filter the queries. Moreover, our algorithm performs approximately 30 times faster than PCR when each query has 30 keywords.

6.5.6 *Effectiveness of safe zones*

In this experiment, we illustrate the effect of safe zones on communication cost. In Figure 19, we study the effect of capacity (which affects the safe zone size) and the effect of total number of objects(users) on total communication cost. The baseline approach requires every object to send its location at every timestamp. In contrast, the safe zone based approaches (our and PCR) require the objects sending their locations only when they leave their respective safe zones. Since PCR is designed such that it always assigns the same safe zone as our approach, it has the same total communication cost as our approach. Figure 19 shows that safe zones significantly reduce the total communication cost. In Figure 19a, the total communication cost reduces as the increases because the safe zone size increases with the increase in capacity.

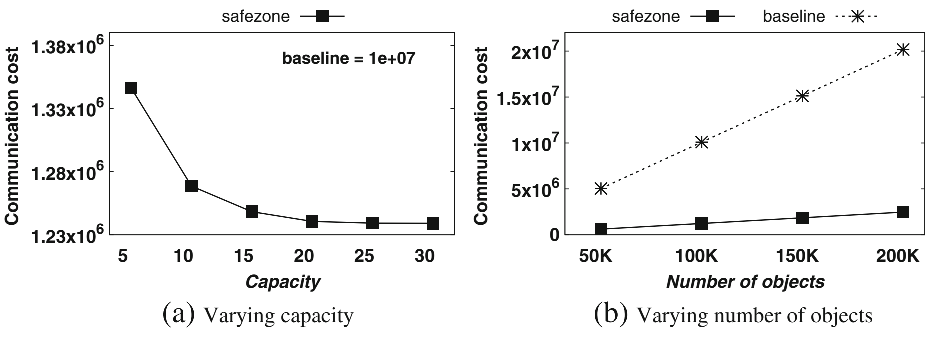
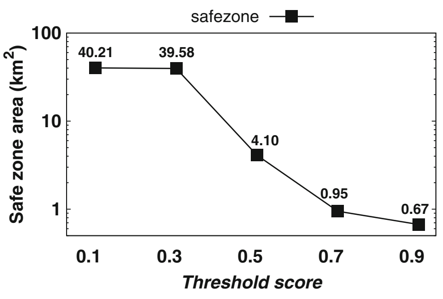
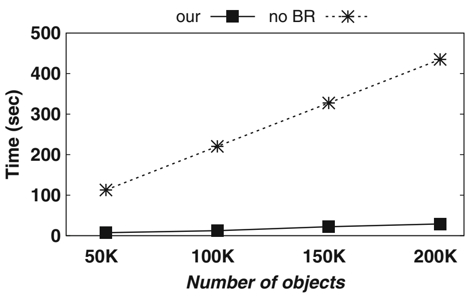


Figure 19 Communication cost



(a) Average area of safe zones



(b) Effectiveness of buffer regions

Figure 20 Experiments on safe zones and buffer regions

Moreover, Figure 20a shows the *average* size of safe zones for different thresholds. Note that we designed our competitor PCR such that it assigns the same sized safe zone as our approach and then retrieves queries to guarantee that the results are unaffected as long as the objects remain in their respective safe zones. Therefore, the safe zone size for both approaches is the same. Figure 20a shows that the size of safe zones reduces as the threshold increases. This is mainly because, as the threshold increases, the object is an answer for more queries which results in a reduced safe zone size. Nevertheless, even for large thresholds, the safe zone is reasonably large ($0.67km^2$ – roughly $800m \times 800m$) which is critical for its effectiveness, e.g., the safe zone based approach is effective when an object stays in it for longer.

6.5.7 *Effectiveness of buffer regions*

In order to show the effect of buffer regions on client workload, we evaluate the effect of buffer regions on the total computation cost at the client devices. Specifically, we compare our approach that uses the buffer region with a version (denoted as "No BR") that does not use buffer regions (i.e., default range Ω and the capacity m are both set to infinity). In Figure 20b, we study the effect of total number of users (client devices) on the total computation cost on *all clients for all 100 timestamps*. Figure 20b shows that the buffer region reduces the client side computation cost by up to four times.

7 Conclusion

We have proposed an efficient solution for processing continuous range spatial keyword queries over moving spatio-textual objects (namely, $CRSK$-*mo* queries). To efficiently process $CRSK$-*mo* queries, we have exploited 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. We have also devised efficient pruning rules to quickly prune objects and queries through smart in-memory data structures for faster processing of queries. Our experimental results show that our approach achieves high performance and good scalability compared to the competitive PCR approach. As for future work, we will extend our work to support top-k queries. Moreover, we are also interested in studying this problem in an environment where keywords are frequently changed.

Acknowledgments Muhammad Aamir Cheema is supported by ARC DE130101002 and DP130103405.

References

1. Brinkhoff, T.: A framework for generating network-based moving objects. GeoInformatica **6**(2), 153–180 (2002)
2. Cai, Y., Hua, K.A., Cao, G., Xu, T.: Real-time processing of range-monitoring queries in heterogeneous mobile databases. IEEE Trans. Mob. Comput. **5**(7), 931–942 (2006)
3. Cary, A., Wolfson, O., Rishe, N.: Efficient and scalable method for processing top-k spatial boolean queries. In: International Conference on Scientific and Statistical Database Management, pp. 87–95. Springer (2010)
4. Cheema, M.A., Lin, X., Zhang, Y., Wang, W., Zhang, W.: Lazy updates: An efficient technique to continuously monitoring reverse knn. Proc. VLDB Endow. **2**(1), 1138–1149 (2009)
5. Cheema, M.A., Lin, X., Zhang, Y., Wang, W., Zhang, W.: Lazy updates: An efficient technique to continuously monitoring reverse knn. PVLDB **2**(1), 1138–1149 (2009). http://www.vldb.org/pvldb/2/vldb09-720.pdf
6. Cheema, M.A., Brankovic, L., Lin, X., Zhang, W., Wang, W.: Continuous monitoring of distance-based range queries. IEEE Trans. Knowl. Data Eng. **23**(8), 1182–1199 (2011)
7. Cheema, M.A., Brankovic, L., Lin, X., Zhang, W., Wang, W.: Continuous monitoring of distance-based range queries. IEEE Trans. Knowl. Data Eng. **23**(8), 1182–1199 (2011). doi:10.1109/TKDE.2010.246
8. Cheema, M.A., Zhang, W., Lin, X., Zhang, Y., Li, X.: Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks. VLDB J. **21**(1), 69–95 (2012). doi:10.1007/s00778-011-0235-9
9. Cheema, M.A., Lin, X., Zhang, W., Zhang, Y.: A safe zone based approach for monitoring moving skyline queries. In: Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, pp. 275–286. Genoa (2013). doi:10.1145/2452376.2452409
10. Chen, L., Cong, G., Jensen, C.S., Wu, D.: Spatial keyword query processing: an experimental evaluation. In: Proceedings of the VLDB Endowment, vol. 6, pp. 217–228. VLDB Endowment (2013)
11. Chen, L., Cong, G., Cao, X., Tan, K.L.: Temporal spatial-keyword top-k publish/subscribe. In: 2015 IEEE 31st International Conference on Data Engineering, pp. 255–266. IEEE (2015)
12. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial Web objects. Proc. VLDB Endow. **2**(1), 337–348 (2009)
13. De Felipe, I., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: IEEE 24th International Conference on Data Engineering, 2008. ICDE 2008, pp. 656–665. IEEE (2008)
14. Gedik, B., Liu, L.: Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In: Advances in Database Technology-EDBT 2004, pp. 67–87. Springer (2004)
15. Guo, L., Chen, L., Zhang, D., Li, G., Tan, K.L., Bao, Z.: Elaps: An efficient location-aware pub/sub system. In: 2015 IEEE 31st International Conference on Data Engineering, pp. 1504–1507. IEEE (2015)
16. Guo, L., Shao, J., Aung, H.H., Tan, K.L.: Efficient continuous top-k spatial keyword queries on road networks. GeoInformatica **19**(1), 29–60 (2015)
17. Guo, L., Zhang, D., Li, G., Tan, K.L., Bao, Z.: Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 843–857. ACM (2015)
18. Hu, H., Xu, J., Lee, D.L.: A generic framework for monitoring continuous spatial queries over moving objects. In: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 479–490. ACM (2005)
19. Hu, H., Liu, Y., Li, G., Feng, J., Tan, K.L.: A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In: 2015 IEEE 31st International Conference on Data Engineering, pp. 711–722. IEEE (2015)
20. Huang, W., Li, G., Tan, K.L., Feng, J.: Efficient safe-region construction for moving top-k spatial keyword queries. In: Proceedings of the 21st ACM international conference on Information and knowledge management, pp. 932–941. ACM (2012)
21. Jung, H., Kim, Y.S., Chung, Y.D.: Qr-tree: An efficient and scalable method for evaluation of continuous range queries. Inform. Sci. **274**, 156–176 (2014)
22. Li, G., Wang, Y., Wang, T., Feng, J.: Location-aware publish/subscribe. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 802–810. ACM (2013)

23. Li, Z., Lee, K.C., Zheng, B., Lee, W.C., Lee, D., Wang, X.: Ir-tree: An efficient index for geographic document search. IEEE Trans. Knowl. Data Eng. **23**(4), 585–599 (2011)
24. Mokbel, M.F., Xiong, X., Aref, W.G.: Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pp. 623–634. ACM (2004)
25. Mouratidis, K., Papadias, D., Hadjieleftheriou, M.: Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In: Proceedings of the 2005 ACM SIGMOD international conference on Management of data, pp. 634–645. ACM (2005)
26. Prabhakar, S., Xia, Y., Kalashnikov, D.V., Aref, W.G., Hambrusch, S.E.: Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. IEEE Trans Comput **51**(10), 1124–1140 (2002)
27. Rocha-Junior, J.B., Gkorgkas, O., Jonassen, S., Nørvåg, K.: Efficient processing of top-k spatial keyword queries. In: Advances in Spatial and Temporal Databases, pp. 205–222. Springer (2011)
28. Šaltenis, S.: Indexing the positions of continuously moving objects. In: Encyclopedia of GIS, pp. 538–543. Springer (2008)
29. Tao, Y., Papadias, D., Sun, J.: The tpr*-tree: An optimized spatio-temporal access method for predictive queries. In: Proceedings of the 29th International Conference on Very Large Data Bases, vol. 29, pp. 790–801. VLDB Endowment (2003)
30. Wang, X., Zhang, Y., Zhang, W., Lin, X., Wang, W.: Ap-tree: Efficiently support continuous spatial-keyword queries over stream. In: 2015 IEEE 31st International Conference on Data Engineering (ICDE), pp. 1107–1118. IEEE (2015)
31. Wu, D., Yiu, M.L., Jensen, C.S., Cong, G.: Efficient continuously moving top-k spatial keyword query processing. In: 2011 IEEE 27th International Conference on Data Engineering (ICDE), pp. 541–552. IEEE (2011)
32. Wu, D., Yiu, M.L., Cong, G., Jensen, C.S.: Joint top-k spatial keyword query processing. IEEE Trans. Knowl. Data Eng. **24**(10), 1889–1903 (2012)
33. Wu, K.L., Chen, S.K., Yu, P.S.: On incremental processing of continual range queries for location-aware services and applications. In: The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, pp. 261–269. IEEE (2005)
34. Yu, X., Pu, K.Q., Koudas, N.: Monitoring k-nearest neighbor queries over moving objects. In: 21st International Conference on Data Engineering (ICDE'05), pp. 631–642. IEEE (2005)