



Article

Continuous k Nearest Neighbor Queries over Large-Scale Spatial–Textual Data Streams

Rong Yang  and Baoning Niu * 

College of Information and Computer, Taiyuan University of Technology, Taiyuan 030024, China; yangrong@tyut.edu.cn

* Correspondence: niubaoning@tyut.edu.cn; Tel.: +86-1580-340-9571

Received: 19 October 2020; Accepted: 18 November 2020; Published: 20 November 2020



Abstract: Continuous k nearest neighbor queries over spatial–textual data streams (abbreviated as CkQST) are the core operations of numerous location-based publish/subscribe systems. Such a system is usually subscribed with millions of CkQST and evaluated simultaneously whenever new objects arrive and old objects expire. To efficiently evaluate CkQST, we extend a quadtree with an ordered, inverted index as the spatial–textual index for subscribed queries to match the incoming objects, and exploit it with three key techniques. (1) A memory-based cost model is proposed to find the optimal quadtree nodes covering the spatial search range of CkQST, which minimize the cost for searching and updating the index. (2) An adaptive block-based ordered, inverted index is proposed to organize the keywords of CkQST, which adaptively arranges queries in spatial nodes and allows the objects containing common keywords to be processed in a batch with a shared scan, and hence a significant performance gain. (3) A cost-based k-skyband technique is proposed to judiciously determine an optimal search range for CkQST according to the workload of objects, to reduce the re-evaluation cost due to the expiration of objects. The experiments on real-world and synthetic datasets demonstrate that our proposed techniques can efficiently evaluate CkQST.

Keywords: spatial–textual queries; continuous queries; nearest neighbor query; data streams

1. Introduction

The continuous k nearest neighbor queries over spatial–textual data streams (abbreviated as CkQST) retrieve to and continuously monitor at most k nearest neighbor (abbreviated as kNN) objects at the user-specified location containing all the user-specified keywords, which have been widely used in a variety of location-based applications, such as location-aware targeting of advertisements, analysis of micro-blogs, and mobile navigation-services.

In an e-coupon recommendation system or a Weibo publish/subscribe system, users register his/her interests (e.g., favorite food or clothing brand for the former, and news or persons for the latter) as a query. A stream of spatial–textual objects (e.g., e-coupons or Weibos) generated are fed to the relevant users. Continuous queries over spatial–textual data streams studied by existing work [1–12] are primarily in terms of Boolean matching or approximate matching, which return an unpredictable number of objects or approximate results. The number of qualified objects containing all keywords specified by a user can be far larger than k , because the objects (e.g., tweets, news) usually contain much more keywords than queries do. This motivates us to study CkQST, which return at most k nearest neighbor objects containing all the query keywords.

Example 1. Figure 1 depicts a running example used throughout this paper. At timestamp t_0 , there are five subscribed 2-NN (i.e., $k = 2$) queries q_1, q_2, \dots, q_5 with a small circle representing their geo-location, and five objects o_1, o_2, \dots, o_5 with a small square representing their geo-location in Figure 1a, while corresponding keywords and expiration times are shown in Figure 1e. The spatial

region is organized by a three-layer quadtree, where the spatial nodes are numbered successively, and the root node is n_0 . Taking the evaluation of q_1 as an example, $\{o_4, o_1\}$ is returned. For q_1 , the spatial search range, thereafter “search range”, is defined as a minimal circle centered at the geo-location of q_1 and covering $\{o_4, o_1\}$, i.e., C_1 . At timestamp t_1 , an object o_6 arrives, as shown in Figure 1b, with keywords $\{w_1, w_2, w_3\}$ and expiration time t_2 . o_6 contains all the keywords of q_1 and q_4 , but only C_1 is hit by o_6 . The result and search range of q_1 are updated to $\{o_6, o_4\}$ and the circle C_1' , respectively, while the result and search range of q_4 are not affected. At timestamp t_2 , o_6 expires. For q_1 , the number of qualified objects in C_1' is less than 2, so the result should be re-evaluated. The result and search range of q_1 are updated to $\{o_4, o_1\}$ and the circle C_1 , respectively. Therefore, for CkQST, the spatial search range covering kNN objects changes dynamically with the arrival and expiration of qualified objects.

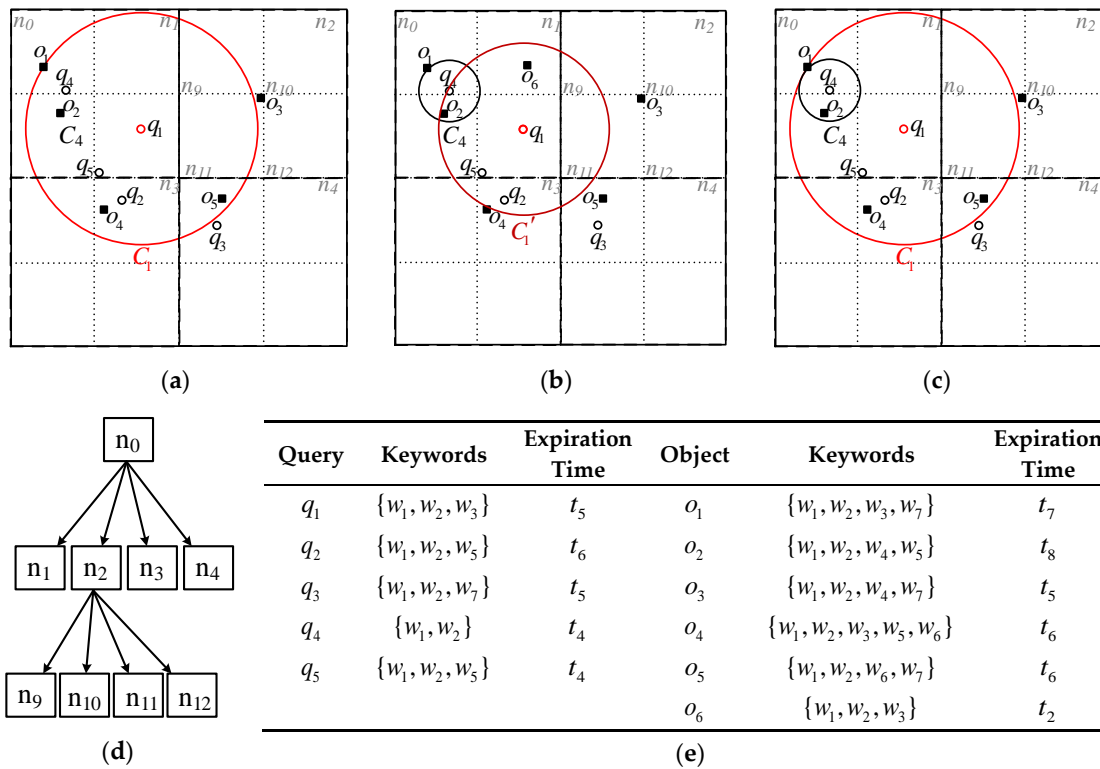


Figure 1. Running example. (a) Spatial description of queries and objects at timestamp t_0 ; (b) spatial description of queries and objects at t_1 ; (c) spatial description of queries and objects at t_2 ; (d) quadtree; (e) textual and time description of queries and objects.

Challenges. The solution framework for evaluating generic continuous queries over spatial–textual data streams consists of selecting an appropriate spatial index and a textual index to form a hybrid spatial–textual index, and exploiting it with appropriate spatial and/or textual filtering strategies to process the incoming objects according to the features of queries [1–12]. There are three key challenges in constructing such an index for CkQST.

First, regarding the spatial filtering, evaluating CkQST is essentially identifying queries whose search range is hit by the incoming objects. It is very important to efficiently organize the search ranges of CkQST; therefore, how to map the search range of CkQST to the spatial nodes is the focus. The search range of CkQST covering kNN objects changes frequently with the arrival and expiration of qualified objects, which requires the index to have both strong filtering ability and low update cost. For most spatial indexes, having strong filtering ability and low update cost are contradictory. There are two approaches to mapping the search range of queries to spatial nodes to improve the filtering ability and reduce the update cost of the index. (1) Queries are mapped to the leaf nodes in the spatial index, which minimizes the spatial region of the nodes covering the search range of queries to reduce the

number of objects to be verified [2,5,7–10,13–15]. (2) Queries are mapped to the spatial nodes according to the spatial distribution [10–12], the keyword distribution [6], or the corresponding cost model [1,3,4]. These approaches are appropriate in the scenarios where the search range of the queries rarely changes, but inappropriate to CkQST, where frequent update of the search range of the queries results in high costs.

Second, regarding the textual filtering, evaluating CkQST is essentially identifying queries whose keywords are fully contained in a given object. An inverted index is usually used to organize continuous queries [1,2,6,10]. A large number of queries make the posting lists very long, and the fast-arriving objects are verified against the corresponding posting lists in multiple rounds in a short time, which becomes the bottleneck of textual filtering. There are three ways to improve textual filtering capabilities. (1) Insert queries into the shortest posting list according to the frequency of query keywords to reduce the number of queries in posting lists, such as in the ranked-key inverted index [1,6]. The posting lists may still be long. (2) Increase the depth of textual partition, such as the ordered keyword trie [3,4,6]. It takes much time to construct the index, and nodes must be reconstructed if queries are updated, which is not appropriate for the scenarios like CkQST where the queries are frequently updated. (3) Organize queries in posting lists in the ascending order according to the ranking score [10]. However, there is no corresponding concept in CkQST. None of the above approaches can efficiently support CkQST textual filtering.

Third, the kNN re-evaluation is frequently triggered by object expiration. When an object expires, several CkQST have to be re-evaluated from scratch, which is expensive. Several techniques have been proposed to solve the similar problems in approximate top- k query (e.g., [10,13,16,17]). They all favor maintaining more than k results to reduce the chances for re-evaluation. However, they either maintain all the skyline objects in the entire region [13,16], or maintain a k -skyband containing skyline objects whose scores were larger than a threshold [10,17], which are not designed for the CkQST returning exact results.

In view of the challenges, we extend a quadtree with an ordered, inverted index to organize CkQST. Three key techniques are proposed to exploit the spatial–textual index and address the above three challenges. The contributions of this paper follow.

(1) To support the frequent change of search ranges of CkQST, a memory-based cost model is proposed to map the search ranges of CkQST to the quadtree nodes, which minimizes the verification cost and index update cost.

(2) To reduce the number of queries verified and process objects in batches, an adaptive block-based ordered, inverted index is proposed to organize the query keywords at quadtree nodes, which allow multiple objects containing common texts to be verified concurrently. For this index, an insertion strategy is proposed to adaptively insert queries in views of the skewed distributions of CkQST and objects.

(3) To reduce the re-evaluation cost, a cost-based k -skyband technique is proposed to judiciously determine the search range for CkQST according to the workload of objects, which minimize the verification cost, update cost, and the re-evaluation cost.

The experiments on real-world and synthetic datasets demonstrate that the proposed techniques can efficiently evaluate CkQST. Compared with the state-of-the-art techniques, when the number of CkQST reaches 20 M, the average index updating time caused by incoming objects decreases by 61%, and the average incoming object processing time decreases by 36%. Compared with the re-evaluation from scratch, the average processing time for expired objects decreases by 99.99%. The rest of this paper is organized as follows. Section 2 formally defines CkQST and presents a framework for evaluating CkQST. Section 3 presents three key techniques for evaluating CkQST. Section 4 reports the experimental studies. Finally, Section 5 concludes this paper.

2. The Framework for Evaluating CkQST

In this section, we formally define CkQST in Section 2.1 and present a framework to evaluate CkQST in Section 2.2.

2.1. Problem Definition

A spatial–textual object is defined as $o = (loc, \psi, t_e)$, where $o.loc$ is the geo-location, $o.\psi$ is a set of keywords (terms) from a vocabulary set \mathcal{V} , and $o.t_e$ is a timestamp indicating the expiration time of o . All the spatial–textual objects over the data streams are denoted as \mathcal{O} . A CkQST is defined as $q = (loc, \psi, k, t_e)$, where $q.loc$, $q.\psi$, and $q.t_e$ follow the similar meaning to o , $q.k$ is the number of returned objects, i.e., at most $q.k$ (abbreviated as k) results are maintained for q . The result list of q , denoted as $q(\mathcal{O})_k$, contains a set of k objects, each of which covers all the keywords in $q.\psi$. $q(\mathcal{O})_k$ is organized by a linked list, in which objects are arranged in the ascending order according to the distances to q . Formally, $\forall o \in q(\mathcal{O})_k ((\nexists o' \in \mathcal{O} \setminus q(\mathcal{O})_k) (o'.\psi \supseteq q.\psi \wedge dist(o', q) \leq dist(o, q)))$, where $dist(o, q)$ is the Euclidean distance between o and q . Let $q.kdist$ be the distance between q and its k^{th} nearest neighbor result. The search range for q , denoted as $q.SR_k$, is defined as a circle centered at $q.loc$ with radius $q.kdist$.

Spatial–textual objects are usually advertisements published by merchants or the latest breaking news, and CkQST are users' search requests. Hereafter spatial–textual object and CkQST are abbreviated as object and query, respectively, if there is no ambiguity. To simplify the calculation, the terms in the vocabulary set \mathcal{V} are mapped to integers between 1 and $|\mathcal{V}|$ according to the alphabetical order, where $|\mathcal{V}|$ is the number of terms in \mathcal{V} . We assume that the terms in \mathcal{V} , and the terms contained in queries and objects are sorted in increasing order. Specifically, for $\forall q \in \mathcal{Q}$, we use $q.\psi[i]$ to denote the i^{th} keyword of q , $q.\psi[i : j]$ to denote a subset of $q.\psi$, i.e., $\cup_{i \leq l \leq j} \{q.\psi[l]\}$, $q.\psi[: i]$ to denote $\cup_{1 \leq l \leq i} \{q.\psi[l]\}$, $q.\psi[i :]$ to denote $\cup_{i \leq l \leq |q.\psi|} \{q.\psi[l]\}$, and $|q.\psi|$ to denote the number of keywords in $q.\psi$. Objects follow the similar notations. Table 1 summarizes the notations used throughout this paper.

Problem Statement. Given a set of CkQST \mathcal{Q} and spatial–textual data streams \mathcal{O} , for each CkQST, find the kNN objects containing all the query keywords over \mathcal{O} whenever objects arrive or expire.

Table 1. Summary of notations.

Notation	Description
$q(q.loc, q.\psi, q.k, q.t_e)$	A CkQST (the geo-location, keywords, expiration time and the number of returned objects of q)
$o(o.loc, o.\psi, o.t_e)$	An object (the geo-location, keywords, and expiration time of o)
$q(\mathcal{O})_k, q(\mathcal{O})$	The result list and extended result list of q
$q.SR_k, q.SR$	The search range and extended search range of q
$q.\psi[i : j], q.\psi[i :], q.\psi[: i]$	The subset of $q.\psi$
$ q.\psi , o.\psi $	The number of keywords in $q.\psi$ and $o.\psi$
$\mathcal{V}, \mathcal{V} $	A vocabulary set and the number of terms in \mathcal{V}
N, n	The quadtree node
$PL_{w_1 w_2}$	The posting list of the ordered, inverted index
b, b_r	The block of a posting list
$b_r.minw, b_r.maxw$	The minimum and maximum $q_i.\psi[3]$ for any query q_i in b_r
$b_r.\psi$	The terms contained in $[b_r.minw, b_r.maxw]$
$ b $	The number of queries in b
$ B $	The number of blocks in a posting list
$C_V^{PL}(PL_{w_1 w_2}), C_U^{PL}(PL_{w_1 w_2})$	The verification cost and update cost of $PL_{w_1 w_2}$
$C_V^q(q, N), C_U^q(q, N)$	The verification cost and update cost within unit time interval if q is associated with N
$p_V^B(b_r)$	The probability that the block b_r is verified
$p_V^q(q N)$	The probability that q is verified if it is inserted into N
$p_V^w(w_j)$	The probability that these queries subjected to $q_i.\psi[3] = \{w_j\}$ are verified in b_r

2.2. The Framework for Evaluating CkQST

The framework for evaluating CkQST shown in Figure 2 consists of two indexes and four key techniques. The object index organizes the objects and can be implemented with any existing spatial–textual index, and we adopt the inverted linear quadtree (IL-quadtree) [18] as an example. The query index organizes queries, which is essentially a quadtree integrated with an ordered, inverted index described in Section 3.

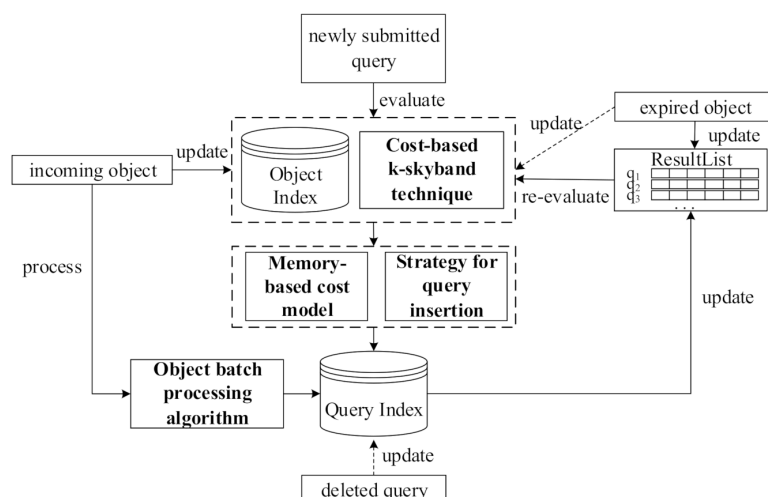


Figure 2. A framework for evaluating the continuous k nearest neighbor queries over spatial–textual data streams (CkQST).

The arrival and expiration of objects. When multiple objects arrive in a batch, they are inserted into the object index, and processed by the object-batch processing algorithm with the help of the query index to find all the affected queries and update the corresponding queries' results and search ranges. When objects expire, the result list of affected queries is checked. Those queries that cannot be refilled through their result list are re-evaluated from scratch against the object index. To save computational cost, the expired objects are removed lazily from the object index until they are accessed again.

The arrival and deletion of queries. When a new query is submitted, it is initially evaluated using the object index with several strategies. A cost-based k-skyband technique is used to find an optimal search range for the query to reduce the cost for updating the index by sacrificing a little bit of filtering performance. A memory-based cost model is used to get the corresponding mapped spatial nodes. An adaptive insertion strategy is used to get the posting list and the corresponding block to be inserted. These strategies can further improve the filtering performance of the index and reduce the cost for updating the index. When a query is deleted or its search range shrinks, a flag is set in the corresponding nodes, where a query table is maintained, and it is not removed from the query index until accessed again, which is called delayed deletion and is necessary in an update-friendly system. A query insertion request might cancel the marked items, which avoid the deletion of objects changing frequently. If a query is deleted, its result list is also removed.

3. The Query Index

According to the above discussions, the query index is essentially a quadtree extended with an ordered, inverted index. Three techniques are proposed to enhance the filtering ability and reduce the update cost of the index. Section 3.1 introduces the motivations. Section 3.2 describes the ordered, inverted index, followed by a detailed adaptive query inserting algorithm in Section 3.3. Section 3.4 proposes the memory-based cost model to quantitatively analyze how to find optimal associated nodes for CkQST. The algorithm for processing objects in batches is presented to improve the throughput in Section 3.5. The re-evaluation technique is introduced in Section 3.6.

3.1. Motivations

Organizing the search range of CkQST. The first issue of using a quadtree to organize the search range of CkQST is how to map the search range to the quadtree nodes. Given that $\forall q \in Q, q.SR_k$ can be mapped to any set of quadtree nodes $NS = \{n_1, n_2, \dots, n_i\}$, only if the union of the spatial region corresponding to these nodes in NS covers $q.SR_k$, which is also called that q is associated with NS . Associating a query with the quadtree nodes is challenging because it affects two computation costs: (1) Verification cost, i.e., the cost of verifying the query with the objects falling in the associated nodes. (2) Update cost, i.e., the cost of inserting or deleting the query in or from the associated nodes. If the search range is organized by nodes with large regions, the index update cost is small, and the verification cost is large; otherwise, if multiple nodes with small regions are used, the situation is reversed. Therefore, a cost model is required to trade off the verification cost and update cost, and find the optimal associated nodes for CkQST.

Organizing the keywords of CkQST. When new objects arrive, the cost of verifying these objects with queries in spatial nodes is expensive. How to reduce the verification cost is the key to improve the filtering ability of the index. We discuss three aspects of constructing an inverted index. (1) For an inverted index, queries in posting lists are usually unordered. For the five queries in Figure 1, we attached them to the posting list of a single keyword. Figure 3a is the inverted index in which queries are attached to the posting list corresponding to the first query keyword, and Figure 3b is the ranked-key inverted index in which queries are attached to the posting list corresponding to the least frequent keyword. If the incoming objects contain the corresponding term, all queries in posting lists are verified [1,2], which is inefficient. In this work, we use an ordered, inverted index to solve the above problem. Figure 3c is the ordered, inverted index if queries are attached to the posting list corresponding to the first keyword, i.e., queries in posting lists are organized in the ascending order according to the keywords. When o_6 with keywords $\{w_1, w_2, w_3\}$ arrives, the posting list corresponding to w_1 is verified. When o_6 is verified with q_2 , its keywords are smaller than q_2 , so we can terminate the verification early and speed up processing objects.

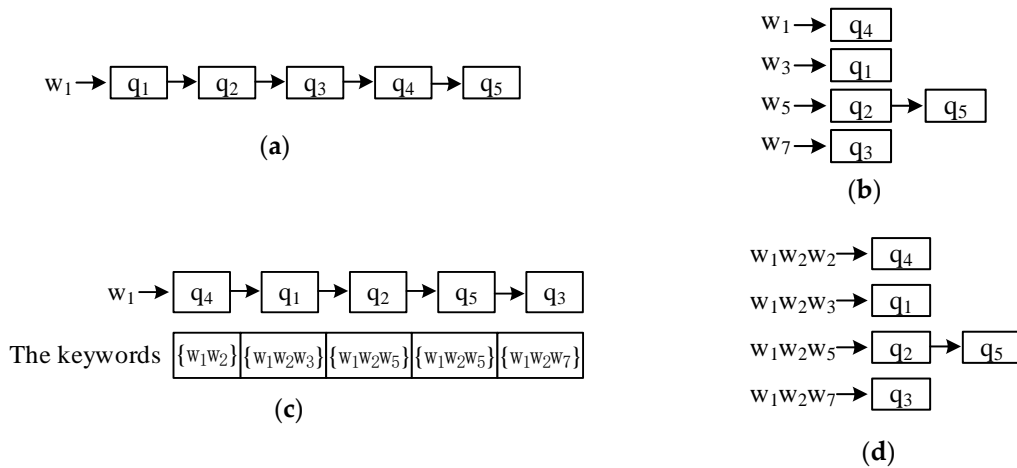


Figure 3. Inverted index. (a) Inverted index; (b) ranked-key inverted index; (c) ordered, inverted index; (d) ordered, inverted index constructed by three keywords. As q_4 contains less than three keywords, we expand its keywords by duplicating the last keyword to construct the ordered index.

(2) Compared with the ordered, inverted index constructed by single keyword, the ordered, inverted index constructed by multiple keywords has more advantages. The length is shorter and the verification probability is smaller. As Figure 3d shows, o_6 is verified with the first two posting lists and contains all the keywords of the queries in these posting lists. However, the number of posting lists might grow sharply. If the number of terms contained in vocabulary set \mathcal{V} is 1 M, and the number of query keywords are not more than 5, total $10^{6 \times 5}$ posting lists are required, which is difficult to

implement by hash table due to the need for large continuous memory. Like other works in [1–12], this paper uses the Map class in Microsoft Visual Studio [19] to build the ordered, inverted index. Lemma 1 describes the verification efficiency with the number of keywords for constructing the ordered, inverted index. Section 4.2 verifies this lemma through experiments. Based on the discussions, we select two keywords to construct the ordered, inverted index.

(3) Usually, there are many queries in posting lists, but only a small number match the incoming objects. Therefore, quickly locating the queries to be verified in posting lists is another way to improve the efficiency of evaluating CkQST. The queries in posting list are partitioned into multiple blocks such that objects are verified with the queries in a few blocks rather than the whole posting list. The only problem is how to partition these queries in posting lists. It is inefficient to have too many or too few queries in a block. An adaptive insertion strategy is proposed in Section 3.3.

3.2. Ordered, Inverted Index

The formal definition of an ordered, inverted index constructed using two keywords follows. Queries are attached to the posting list of their first two keywords, and arranged in ascending order according to their keywords.

Definition 1 (Ordered Posting List/Ordered, Inverted Index). Given a set of queries q_1, q_2, \dots, q_i to be inserted into a quadtree node, if $q_1.\psi[1 : 2] = q_2.\psi[1 : 2] \dots = q_i.\psi[1 : 2] = \{w_{i1}, w_{i2}\}$, and $q_1.\psi[3 :] \leq q_2.\psi[3 :] \leq \dots \leq q_i.\psi[3 :]$, the posting list determined by the two terms w_{i1}, w_{i2} at the node is denoted as $PL_{w_{i1}w_{i2}}$, in which these queries are successively inserted. $PL_{w_{i1}w_{i2}}$ is called an ordered posting list. Specifically, if these queries only contain one keyword, the corresponding posting list is denoted as $PL_{w_{i1}w_{i1}}$. All the ordered posting lists constitute the ordered, inverted index.

Hereafter the ordered posting list is abbreviated as posting list, if there is no ambiguity. To quickly locate the queries to be verified, posting lists are divided into multiple blocks.

Definition 2 (Block). Given any ordered posting list $PL_{w_{i1}w_{i2}}, w_{i1} \neq w_{i2}, b_r$ is the r^{th} block of $PL_{w_{i1}w_{i2}}$. For any query $q \in b_r, q.\psi[3] \in [b_r.\text{min}w, b_r.\text{max}w]$, where $b_r.\text{min}w = \min_{q_i \in b_r, q_i.\psi[3]}$, $b_r.\text{max}w = \max_{q_i \in b_r, q_i.\psi[3]}$. $b_r.\psi$ denotes all the keywords satisfying $b_r.\psi = \{w | w \in [b_r.\text{min}w, b_r.\text{max}w]\}$. Specially, if q only contains one or two keywords, it is inserted into the block b_0 of the corresponding posting list.

Lemma 1. If the number of keywords for constructing an ordered, inverted index is $m (m \geq 1)$, there are at most $|\mathcal{V}|^m$ posting lists at a node. For any object o containing more than two keywords, the verification cost can be estimated by Equation (1). Where $|\mathcal{B}|$ is the number of blocks in a posting list, $|b|$ is the number of queries in b , Q contains queries whose keywords are contained in o , and the number of keywords is less than m . The proof is shown in Appendix A.

$$O(|o.\psi|^m \cdot \log|\mathcal{V}|^m + |o.\psi|^{m+1} \cdot (\log|\mathcal{B}| + \frac{|b|}{(|\mathcal{B}||b.\psi|)^{m-1}}) + |Q|) \quad (1)$$

For $\forall w_j \in b.\psi$, the verification probability, denoted as $p_V^w(w_j)$, is maintained, i.e., the probability of verifying these queries subjected to $q_i.\psi[3] = \{w_j\}$ in b_r . For $\forall b_r$, the verifying probability, denoted as $p_V^B(b_r)$, is maintained, i.e., the probability that the block b_r is verified, which can be estimated by Equation (2).

$$\min \left\{ \max_{w \in b_r.\psi} p_V^w(w) + \frac{1}{|b_r.\psi| - 1} \left(\sum_{w \in b_r.\psi} p_V^w(w) - \max_{w \in b_r.\psi} p_V^w(w) \right), 1 \right\} \quad (2)$$

The following theorems claim that the incoming object is verified with few queries in posting lists. For any incoming object, the blocks being verified can be located according to block keyword interval

$[b_r.minw, b_r.maxw]$ (see Theorem 1). The object verification with a block or a posting list can be terminated if the keywords are smaller than that of some query being verified. (see Theorem 2).

Theorem 1. Given $\forall o \in \mathcal{O}$ and a posting list $PL_{w_1w_2}$ being verified with o , for $\forall b_r$ in $PL_{w_1w_2}$, if $\exists q$ in b_r , o contains all keywords of q , only if $\exists w_j \in o.\psi$, $w_j \in [b_r.minw, b_r.maxw]$.

Proof. Suppose that o contains all the keywords of q , but there is no term in $o.\psi$ satisfying $w_j \in [b_r.minw, b_r.maxw]$. Based on Definition 2, $q.\psi[3] \in [b_r.minw, b_r.maxw]$, o does not contain the third keyword of q , so o does not contain all the keywords of q . It is contradicted by the hypothesis, so the theorem is proved. \square

Theorem 2. Given $\forall o \in \mathcal{O}$ and a posting list $PL_{w_1w_2}$ being verified with o , for $\forall b_r$ in $PL_{w_1w_2}$, we have the following conclusions. (1) If $b_r.minw > \max\{w \in o.\psi\}$, o is not the result of queries in the blocks starting from b_r , and $PL_{w_1w_2}$ can be terminated verifying. Specifically, for $\forall q_i$ in b_r , if $q_i.\psi[3] > \max\{w \in o.\psi\}$, $PL_{w_1w_2}$ can be terminated verifying. (2) If $b_r.maxw < \min\{w \in o.\psi \mid w > w_{i2}\}$, o is not the result of queries in b_r .

Proof. (1) If $b_r.minw = \min_{q_i \in b_r} q_i.\psi[3] > \max\{w \in o.\psi\}$, i.e., for any q_i in b_r , o does not contain its third keyword, o is not the result of q_i . So does the block b_{r+j} , since $b_{r+j}.minw > b_r.minw > \max\{w \in o.\psi\}$. Similarly, for any q_i in b_r , $q_{i+j}.\psi[3] > q_i.\psi[3] > \max\{w \in o.\psi\}$, o is not the result of queries after q_i . (2) If $b_r.maxw = \max_{q_i \in b_r} q_i.\psi[3] < \min\{w \in o.\psi \mid w > w_{i2}\}$, o is not the result of the queries in b_r . \square

3.3. Adaptive Query Insertion Algorithm

Given any posting list at a node, we consider two extreme situations: (1) the posting list only contains a block which contains all queries; (2) the posting list contains many blocks, each of which only contain one query. The former has poor filtering ability and the latter has high update cost. Neither is what we expect. In the real world, people are concerned with different interests and often pay high attention to the breaking news or topical issues, so the keywords of the queries and objects vary over time. For each query, we adaptively insert it into the posting lists according to the historical queries and objects. We expect that the increase of the verification cost and update cost of the posting list is minimal after the query being inserted.

Given a posting list $PL_{w_1w_2}$, the update cost is denoted as $C_U^{PL}(PL_{w_1w_2})$, and the verification cost is denoted as $C_V^{PL}(PL_{w_1w_2})$, which can be estimated by Equation (3), where $C_V^B(b_r) = p_V^B(b_r) * (\log|\mathcal{B}| + |b_r|)$ represents the verification cost of the block b_r in $PL_{w_1w_2}$.

$$C_V^{PL}(PL_{w_1w_2}) = \sum_{b_r} C_V^B(b_r) \quad (3)$$

Theorem 3. Let q be the query to be inserted into $PL_{w_1w_2}$, $q.\psi[1 : 3] = \{w_{i1}, w_{i2}, w_j\}$, $PL_{w_1w_2}$ has $|\mathcal{B}| (|\mathcal{B}| \geq 1)$ blocks, ΔC_V^{PL} is the increase of verification cost of $PL_{w_1w_2}$ after q being inserted. We have the following conclusions.

Case 1: If $\exists b_r \in PL_{w_1w_2}$ satisfies $w_j \in [b_r.minw, b_r.maxw]$, q is inserted into b_r . $\Delta C_V^{PL} = p_V^B(b_r)$.

Case 2: If $\nexists b \in PL_{w_1w_2}$ satisfies $w_j \in [b.minw, b.maxw]$, but $\exists b_r \in PL_{w_1w_2}$ satisfies $b_r.maxw < w_j$, q is inserted into the tail of b_r . The updated block is denoted as b_r' . $\Delta C_V^{PL} = (p_V^B(b_r') - p_V^B(b_r)) * (\log|\mathcal{B}| + |b_r|) + p_V^B(b_r')$.

Case 3: Similar to case 2, if $\nexists b \in PL_{w_1w_2}$ satisfies $w_j \in [b.minw, b.maxw]$, but $\exists b_r \in PL_{w_1w_2}$ satisfies $w_j < b_r.minw$, q is inserted into the head of the b_r . $\Delta C_V^{PL} = (p_V^B(b_r') - p_V^B(b_r)) * (\log|\mathcal{B}| + |b_r|) + p_V^B(b_r')$.

Case 4: If $\nexists b \in PL_{w_1w_2}$ satisfies $w_j \in [b.minw, b.maxw]$, a new block b is constructed in $PL_{w_1w_2}$, and q is inserted into b . $\Delta C_V^{PL} = \log((|\mathcal{B}| + 1) / |\mathcal{B}|) * \sum_{b_r} p_V^B(b_r) + p_V^w(w_j) * \log(|\mathcal{B}| + 1)$.

Proof. (1) Case 1: If q is inserted into b_r , the verifying probability $p^B(b_r)$ does not change since $w_j \in [b_r.minw, b_r.maxw]$, but the number of queries in b_r increases by 1.

$$\Delta C_V^{PL} = p_V^B(b_r) * (\log|\mathcal{B}| + |b_r| + 1) - p_V^B(b_r) * (\log|\mathcal{B}| + |b_r|) = p_V^B(b_r)$$

(2) Case 2:

$$\begin{aligned} \Delta C_V^{PL} &= C_V^B(b_r') - C_V^B(b_r) = p_V^B(b_r') * (\log|\mathcal{B}| + |b_r'|) - p_V^B(b_r) * (\log|\mathcal{B}| + |b_r|) \\ &= (p_V^B(b_r') - p_V^B(b_r)) * (\log|\mathcal{B}| + |b_r|) + p_V^B(b_r') \end{aligned}$$

(3) Similar to case 2.

(4) Case 4: For any b_i in $PL_{w_{i1}w_{i2}}$, $C_V^B(b_i) = p_V^B(b_i) * (\log|\mathcal{B}| + |b_i|)$, $C_V^{B'}(b_i) = p_V^B(b_i) * (\log(|\mathcal{B}| + 1) + |b_i|)$

$$\Delta C_V^B = C_V^{B'}(b_r) - C_V^B(b_r) = p_V^B(b_r) * ((\log(|\mathcal{B}| + 1) + |b_r|) - (\log|\mathcal{B}| + |b_r|)) = p_V^B(b_r) * \log(|\mathcal{B}| + 1) / |\mathcal{B}|$$

$$\begin{aligned} \Delta C_V^{PL} &= \sum_{b_r} (C_V^{B'}(b_r) - C_V^B(b_r)) + C_V^B(b) = \sum_{b_r} (p_V^B(b_r) * \log((|\mathcal{B}| + 1) / |\mathcal{B}|)) + C_V^B(b) \\ &= \log((|\mathcal{B}| + 1) / |\mathcal{B}|) * \sum_{b_r} p_V^B(b_r) + p_V^w(w_j) * \log(|\mathcal{B}| + 1) \end{aligned}$$

Theorem 3 shows all the cases where the verification costs increase if a query is inserted into the posting list. The increase of update costs ΔC_U^{PL} corresponding to the above four cases are $O(|b_r| + 1)$, $O(1)$, $O(1)$, and $O(1 + \log|\mathcal{B}|)$ respectively. To compare the verification costs and update costs, we introduce a normalization parameter θ_U ($0 < \theta_U \leq 1$) to represent the ratio of the update operation to the verification operation, i.e., if a query is inserted into a node, there will be $1/\theta_U$ objects being verified with it. A query is adaptively inserted into the posting list according to the following theorem.

Theorem 4. Let q be the query to be inserted into $PL_{w_{i1}w_{i2}}$, $q.\psi[1 : 3] = \{w_{i1}, w_{i2}, w_j\}$. If $\exists b_r$ in $PL_{w_{i1}w_{i2}}$ satisfies $w_j \in [b_r.minw, b_r.maxw]$, q is inserted into b_r . Otherwise, the minimum $\Delta C_V^{PL} + \theta_U \Delta C_U^{PL}$ of the cases 2–4 is taken.

Given $\forall q \in Q$, if q contains no more than two keywords, it is directly inserted into the b_0 of the corresponding posting list. Algorithm 1 shows how a query q containing more than two keywords is adaptively inserted into a posting list. If $PL_{q.\psi[1]q.\psi[2]}$ does not exist, a new block b is constructed, and q is inserted into b (lines 1–2). Otherwise, a block in $PL_{q.\psi[1]q.\psi[2]}$ is found for q to minimize $\Delta C_V^{PL} + \theta_U \Delta C_U^{PL}$ (lines 3–12). First, we find the block, denoted as b_r , whose $b_r.minw$ is the smallest—no smaller than $q.\psi[3]$ (line 3). If $q.\psi[3] = b_r.minw$, q is inserted into b_r (line 4). If $q.\psi[3] \in [b_{r-1}.minw, b_{r-1}.maxw]$ ($r > 1$), q is inserted into block b_{r-1} (lines 5–6). Otherwise, we compute $\Delta C_V^{PL} + \theta_U \Delta C_U^{PL}$ according to cases 2–4 in Theorem 3 and select the minimum case (lines 7–12). It is worth noting that when compared with the first block of the list, there are only cases 3–4, and if $q.\psi[3]$ is larger than $b_r.minw$ of all the blocks, there are only cases 2 and 4.

Computation complexity. In the worst case, the computation cost of Algorithm 1 is shown as Lemma 1. That is, in posting lists constructed by two keywords, the complexity of inserting a query at a node is $O(|o.\psi|^2 \log|\mathcal{V}|^2 + |o.\psi|^3 (\log|\mathcal{B}| + |b|/|\mathcal{B}| |b.\psi|))$. The algorithm can adaptively adjust $|\mathcal{B}|$ and $|b|$.

Algorithm 1: *InsertQueryPL*($q, PL_{q,\psi[1]q,\psi[2]}$)

```

1 if  $\nexists PL_{q,\psi[1]q,\psi[2]}$  then
2   construct new block  $b$  in  $PL_{q,\psi[1]q,\psi[2]}$  insert  $q$  into  $b$ ; return;
3  $b_r \leftarrow \min\{b.minw \geq q.\psi[3]\}$ ;
4 if  $q.\psi[3] == b_r.minw$  then  $q$  is inserted into  $b_r$ ; return;
5 if  $r > 1$  and  $q.\psi[3] \leq b_{r-1}.maxw$  then
6    $q$  is inserted into  $b_{r-1}$ ; return;
7 if  $\nexists b_r$  then compute  $\min\{\Delta C_V^{PL} + \theta_U \Delta C_U^{PL}\}_{case2,case4}$  on  $b_{\emptyset}$ ;
8 if  $r == 1$  then compute  $\min\{\Delta C_V^{PL} + \theta_U \Delta C_U^{PL}\}_{case3,case4}$ ;
9 if  $r > 1$  then compute  $\min\{\Delta C_V^{PL} + \theta_U \Delta C_U^{PL}\}_{case2,case3,case4}$ ;
10 if case 2 then  $q$  is inserted into the tail of  $b_{r-1}$ ;
11 if case 3 then  $q$  is inserted into the head of  $b_r$ ;
12 if case 4 inserted into a new block.

```

3.4. The Memory-Based Cost Model

A memory-based cost model associates queries with the optimal quadtree nodes. Given the search range of CkQST, the model traversals the quadtree from the root node, compares the sum of the verification cost and index update cost if the query is associated with the current node and its child nodes, and selects the smaller one. The verification cost is the product of the number of verified objects and the expectation of the verification cost, and the update cost is the expectation of the update cost if the query is inserted into the corresponding block of the posting list.

Definition 3 (Minimum Bounding Node). Given $\forall q \in \mathcal{Q}$ and search range $q.SR_k$, if $\exists N, q.SR_k \subseteq N.R$, and for any child node n_i of N , $q.SR_k \not\subseteq n_i.R$, N is the minimum bounding node of q , where $N.R$, $n.R$ are the region where N and n locate respectively. The minimum bounding node of q is the node, which covers its search range, but any of its child nodes cannot completely cover the search range.

Verification cost. Given $\forall q \in \mathcal{Q}$ and its minimum bounding node N , $q.\psi[1 : 3] = \{w_{i1}, w_{i2}, w_{i3}\}$, if q is associated with N , the verification cost within unit time interval, denoted as $C_V^q(q, N)$, can be estimated by Equation (4). We assume that the query and object contain more than two keywords, and the average verification cost is unit time.

$$C_V^q(q, N) = Num_o^N(N) * p_V^q(q|N) * E_V(q|N) \quad (4)$$

Specifically, if the query or the object contains one or two keywords, the verification cost is estimated by Equation (5). This case is simple, so we omit the details.

$$C_V^q(q, N) = Num_o^N(N) * p_V^B(b_0) \quad (5)$$

where $Num_o^N(N)$ is the number of objects falling in N within the unit time interval. $p_V^q(q|N)$ is the probability that q is verified if it is inserted into b_r in N , i.e., the probability that the objects contain the terms w_{i1} , w_{i2} , and $w_j \in [w_{i3}, b_r.maxw]$, and can be estimated by Equation (6).

$$p_V^q(q|N) = p_V^B(b_r) * \frac{b_r.maxw - w_{i3} + 1}{|b_r.\psi|} \quad (6)$$

where $|b_r.\psi|$ is the number of keywords contained in $b_r.\psi$. $E_V(q|N)$ is the verification cost if q is inserted into b_r in N , and can be estimated with the expectation of verification cost of the queries in b_r , i.e., Equation (7).

$$E_V(q|N) = \sum_{w_j} p_V^w(w_j) * (Num_q)_{\leq w_j} \quad (7)$$

where $(Num_q)_{\leq w_j}$ is the number of queries subjected to $q_i.\psi[3] \leq w_j$ in b_r . Similarly, if the query q is associated with a set of non-overlapping nodes, denoted as NS , the verification cost is denoted as $C_V^q(q, NS)$ and can be estimated by Equation (8).

$$C_V^q(q, NS) = \sum_{n_i \in NS} C_V^q(q, n_i) \quad (8)$$

For $\forall q \in Q$, we find the optimal associated nodes starting from its minimum bounding node, and check whether the query is associated with the current node or associated with its child nodes. The difference of two verification costs is estimated by Equation (9).

$$\Delta C_V^q = \sum_{n_i \in INS \cup n} C_V^q(q, n_i) - \sum_{n_i \in INS \cup n.child} C_V^q(q, n_i) \quad (9)$$

where INS keeps the intermediate result, $n.child$ contains the child nodes of n that intersect with the search range of the query. It is worth noting that if $\Delta C_V^q \leq 0$, we terminate the iteration.

Update cost. When inserting or deleting queries in nodes, it will incur an index update cost. We delay deleting queries until these queries are accessed again, so the deletion cost is ignored. If a query q is associated with its minimum bounding node N , and is inserted into a block b_r of posting list $PL_{q.\psi[1]q.\psi[2]}$ in N , the insertion cost consists of two parts, the time to find the corresponding block and the time to find the insertion position. The update cost, denoted as $C_U^q(q, N)$, can be estimated by Equation (10).

$$C_U^q(q, N) = \log|\mathcal{B}| + \frac{1}{|b_r|} \sum_{i=1}^{|b_r|} i = \log|\mathcal{B}| + \frac{|b_r| + 1}{2} \quad (10)$$

If q is associated with a set of non-overlapping nodes, denoted as NS , the update cost is denoted as $C_U^q(q, NS)$ and can be estimated by Equation (11).

$$C_U^q(q, NS) = \sum_{n_i \in NS} C_U^q(q, n_i) \quad (11)$$

Similarly to ΔC_V^q , the difference of two update costs between the query being associated with the node and associated with the child nodes is estimated by Equation (12).

$$\Delta C_U^q = \sum_{n_j \in INS \cup n.child} C_U^q(q, n_j) - \sum_{n_j \in INS \cup n} C_U^q(q, n_j) \quad (12)$$

Given $\forall q \in Q$ and search range $q.SR_k$, we start from the minimum bounding node, and computes ΔC_V^q and ΔC_U^q between the query being associated with the node and associated with the child nodes. If $\Delta C_V^q \geq \theta_U \cdot \Delta C_U^q$, the child nodes are the optimal. Otherwise the node is optimal. The computation cost consists of two parts, finding the minimum bounding node of q , and finding an optimal association in the descendant nodes of minimum bounding node. The computation cost of the first part is $O(\theta_h)$, and the second part is $O((4^{\theta_h} - 1)/3)$, i.e., in the worst case, the node will be partitioned until the leaf node, where θ_h is the height of the quadtree.

3.5. Processing Objects in Batches

For these objects being verified with the same posting list, an object processing algorithm, which is a group matching technique that follows the filtering and verification strategy, is proposed to process objects in batches.

A data structure $\langle bid, w, oset \rangle$ is defined to group the objects being verified with the same posting list, where bid ($bid > 0$) is the block id, w ($w \in [b_{bid}.minw, b_{bid}.maxw]$) is a term, $oset$ is a set of objects being verified with block b_{bid} and containing w . For the convenience of the description, $wset_{bid}$ is a set of terms satisfying $w \in [b_{bid}.minw, b_{bid}.maxw]$, $oset_{bid,w}$ is a set of objects which are verified with the queries in block b_{bid} and contain w .

Algorithm 2 describes how to process a set of objects represented by $\langle bid, w, oset \rangle$, which are verified with the queries in the posting list $PL_{w_1 w_2}$. If b_{bid} is b_0 , the queries in b_0 is verified with the objects in $oset$ (lines 1–5). Otherwise, for any term w_j in $wset_{bid}$, according to Theorem 2, if $b_{bid}.minw > w_j$, we check the next term (line 7); if $b_{bid}.maxw < w_j$, we check the next block (line 8); otherwise, for each query q in b_{bid} , if $q.\psi[3] > w_j$, we check the next term (line 10); if $q.\psi[|q.\psi|] < w_j$, we check the next query (line 11); otherwise, we verify whether the object is the results of q . If yes, $\langle q, o \rangle$ is added to QOS (lines 12–13). Moreover, the result list and search range of q are updated.

Computation complexity. Algorithm 2 describes how to process objects in batches in an ordered posting list. In the worst case, objects are processed individually. As Lemma 1 shows, in posting lists constructed by two keywords, for an object, the time complexity of finding the qualified queries at a node is $O(|o.\psi|^2 \log|\mathcal{V}|^2 + |o.\psi|^3 (\log|\mathcal{B}| + |b|/|\mathcal{B}||b.\psi|))$.

Algorithm 2: ObjectProcessing($PL_{w_1 w_2}, \{\langle bid, w, oset \rangle\}$).

```

1   if  $b_{bid} = b_0$ 
2     for any  $q$  in  $b_{bid}$ 
3       for any  $o$  in  $oset$ 
4         if  $o$  is a result of  $q$  then  $QOS \leftarrow \langle q, o \rangle$ ;
5   return  $QOS$ ;
6   for  $w_j \in wset_{bid}$ 
7     if  $b_{bid}.minw > w_j$  then continue;
8     if  $b_{bid}.maxw < w_j$  then break;
9     for  $\forall q \in b_{bid}$ 
10      if  $q.\psi[3] > w_j$  then break;
11      if  $q.\psi[|q.\psi|] < w_j$  then continue;
12      for  $\forall o \in oset_{bid, w_j}$ 
13        if  $o$  is a result of  $q$  then  $QOS \leftarrow \langle q, o \rangle$ ;
14   return  $QOS$ .

```

3.6. Cost-Based k-Skyband Technique

To reduce the re-evaluation cost, a cost-based k-skyband technique is proposed to judiciously determine an optimal search range for CkQST such that the overall cost defined in the cost model can be minimized. Specifically, for $\forall q \in \mathcal{Q}$, three parameters are defined: an extended search range is denoted as $q.SR$, where $q.SR \supseteq q.SR_k$; a k-skyband, i.e., an extended result list, denoted as $q(\mathcal{O})$, where $q(\mathcal{O}) \supseteq q(\mathcal{O})_k$; the number of objects containing all query keywords within $q.SR$ in the initial timestamp is denoted as $q.\theta_k$, where $q.\theta_k \geq q.k$.

Definition 4 (Loose Matching). Given $\forall o \in \mathcal{O}$ and $\forall q \in \mathcal{Q}$, o loosely matches q only if $q.\psi \subseteq o.\psi$ and $o.loc \in q.SR$. All the objects that loosely match q are denoted as $q(\mathcal{O})_{sup} = \{o \in \mathcal{O} | q.\psi \subseteq o.\psi \wedge o.loc \in q.SR\}$.

Definition 5 (Dominance). Given $\forall q \in \mathcal{Q}$ and two objects o_1, o_2 , which loosely match q , o_1 dominates o_2 only if $dist(q, o_1) < dist(q, o_2)$ and $o_1.t_e \geq o_2.t_e$ or $dist(q, o_1) \leq dist(q, o_2)$ and $o_1.t_e > o_2.t_e$.

Definition 6 (k-skyband/Extended Result list). Given $\forall q \in \mathcal{Q}$, for any incoming object o' , we insert it into the k-skyband $q(\mathcal{O})$ only if: (1) o' loosely matches q ; (2) o' is dominated by less than k other objects. For $\forall q \in \mathcal{Q}$, if an object o' loosely matches q , and there are k objects in $q(\mathcal{O})$ dominating o' , o' would not be a result at any timestamp. Therefore, we would not insert these objects into the result list.

Theorem 5. Given $\forall q \in \mathcal{Q}$ and an extended search range $q.SR$, we always have the following conclusions. (1) $q(\mathcal{O})_k \subseteq q(\mathcal{O})$; (2) $q(\mathcal{O}) \subseteq q(\mathcal{O})_{sup}$; (3) $|q(\mathcal{O})| < k$ iff $|q(\mathcal{O})_{sup}| < k$.

Proof. (1) At any timestamp, for $\forall o \in q(\mathcal{O})_k$, we have $q.\psi \subseteq o.\psi$ and $o.loc \in q.SR_k \subseteq q.SR$, i.e., o loosely matches q , and less than k objects dominate o . So $o \in q(\mathcal{O})$. (2) According to Definition 4, for $\forall o \in q(\mathcal{O})$, o loosely matches q , so $o \in q(\mathcal{O})_{sup}$, $q(\mathcal{O}) \subseteq q(\mathcal{O})_{sup}$. (3) Since $q(\mathcal{O}) \subseteq q(\mathcal{O})_{sup}$, if $|q(\mathcal{O})_{sup}| < k$, we have $|q(\mathcal{O})| \leq |q(\mathcal{O})_{sup}| < k$. On the other hand, if $|q(\mathcal{O})| < k$, $|q(\mathcal{O})_{sup}| \geq k$, i.e., $\exists o \in q(\mathcal{O})_{sup}$ and $o \notin q(\mathcal{O})$, which means o loosely matches q , but o is dominated by more than k other objects, which is contradicted by $|q(\mathcal{O})| < k$. The theorem is proved. \square

According to Theorem 5, the extended result list is the super set of the exact result list, from which we can extract the kNN objects, and the number of objects in extended result list is less than k only if the number of objects in $q(\mathcal{O})_{sup}$ is less than k .

Given $\forall q \in \mathcal{Q}$, an extended search range $q.SR$, and the corresponding extended result list $q(\mathcal{O})$, three costs are defined in the cost-based k-skyband technique: the verification cost of q within $q.SR$, the update cost of $q(\mathcal{O})$, and the re-evaluation cost.

Verification cost. The verification cost of q within $q.SR$, within the unit time interval, denoted as $C_V^R(q|q.SR)$, is estimated by Equation (13), i.e., the verification cost if q is inserted into all the leaf nodes that intersect with $q.SR$.

$$C_V^R(q|q.SR) = \sum_{n.R \cap q.SR \neq \emptyset} C_V^q(q, n) \quad (13)$$

Update cost. Similarly to $q(\mathcal{O})_k$, the extended result list $q(\mathcal{O})$ is organized by a linked list. For $\forall o \in q(\mathcal{O})$, a dominance counter is defined to count the number of objects dominating o . If an incoming object o' is inserted into $q(\mathcal{O})$, the dominance counters of all the objects in $q(\mathcal{O})$ with $dist(q, o') < dist(q, o)$ and $o'.t_e \geq o.t_e$, or $dist(q, o') \leq dist(q, o)$ and $o'.t_e > o.t_e$ will increase by 1, and the objects with dominance counter equal to k will be evicted, which can be processed in $O(|q(\mathcal{O})|)$ time. When an object in $|q(\mathcal{O})|$ expires, it is deleted from $q(\mathcal{O})$ until it is accessed again. The cost is negligible. The update cost of $q(\mathcal{O})$ within unit time interval, denoted as $C_U^L(q|q(\mathcal{O}))$, is estimated by Equation (14). Where $freq_U^o$ is the number of object updates within unit time interval, $p_M^q(q.SR)$ is the probability that the objects loosely match q within the search range $q.SR$, and is estimated by $p_M^q(q.SR) = q.\theta_k / Num_o^N(n_0)$, $1/2$ is the probability that a qualified object arrival, $|q(\mathcal{O})|$ can be estimated by $|q(\mathcal{O})| = \max\{k \cdot \ln(\theta_k/k), \theta_k\}$.

$$C_U^L(q|q(\mathcal{O})) = freq_U^o * 1/2 \cdot p_M^q(q.SR) \cdot |q(\mathcal{O})| \quad (14)$$

Re-evaluation cost. The re-evaluation cost within the unit time interval is denoted as $1/\theta_t C_{Le}(q)$. Where θ_t is the re-evaluation period, i.e., the shortest time required between two consecutive independent evaluations, and $1/\theta_t$ is the frequency of re-evaluation. $C_{Le}(q)$ is the re-evaluation cost, and is approximated to the verification cost in $q.SR_k$, i.e., $C_{Le}(q) = C_V^R(q|q.SR_k) = \sum_{n.R \cap q.SR_k \neq \emptyset} C_V^q(q, n)$.

The overall cost in the cost-based k-skyband technique, denoted as $C_{Re}(q)$, is shown in Equation (15). When $C_{Re}(q)$ is minimal, the search range is optimal.

$$C_{Re}(q) = C_V^R(q|q.SR) + C_U^L(q|q(\mathcal{O})) + 1/\theta_t C_{Le}(q) \quad (15)$$

In the following, we discuss how to get θ_t . θ_t is the re-evaluation period, i.e., the shortest time that $|q(\mathcal{O})|$ is reduced to $k - 1$ since the last re-evaluation. For $\forall q \in \mathcal{Q}$, the update process of number of objects in $q(\mathcal{O})$ can be modeled as a simple random walk, which is a stochastic sequence S_t , with S_0 being the original status, defined by $S_t = \sum_{i=1}^t X_i$, where X_i is the object update, which is an independent and identically distributed random variable. In $q(\mathcal{O})$, if an object is inserted, $X_i = 1$; if an object expires or is dominated, $X_i = -1$; otherwise $X_i = 0$. It's difficult to estimate X_i due to the eviction of objects by the dominance relationship in $q(\mathcal{O})$. For example, an object is inserted, but the number of objects decreases due to the eviction of objects with dominance counters reaching k . According to Theorem 5, the number of objects in $q(\mathcal{O})$ is less than k only if the number of objects in $q(\mathcal{O})_{sup}$ is less than k , and the

objects in $q(\mathcal{O})_{\text{sup}}$ don't dominate each other. Therefore we estimate the shortest time that $|q(\mathcal{O})_{\text{sup}}|$ is reduced to $k - 1$, denoted as θ_t' , where $\theta_t' = \theta_t$. The object update in $q(\mathcal{O})_{\text{sup}}$ at any timestamp can be estimated as Equation (16).

$$\text{prob}(X_i) = \begin{cases} 1/2p_M^q(q.SR) & \text{if } X_i = 1 \\ 1/2p_M^q(q.SR) & \text{if } X_i = -1 \\ 1 - p_M^q(q.SR) & \text{if } X_i = 0 \end{cases} \quad (16)$$

The number of object updates required to reduce the number of objects from $q.\theta_k$ to $k - 1$ in $q(\mathcal{O})$, denoted as $\mathbb{Z}(q)$, is estimated by Equation (17). θ_t is estimated by $\theta_t = \mathbb{Z}(q) / \text{freq}_U^q$.

$$\mathbb{Z}(q) = \frac{2 \cdot (q.\theta_k - k + 1) \cdot q.\theta_k}{p_M^q(q.SR)} + \frac{(q.\theta_k - k + 1) \cdot (q.\theta_k - k + 2)}{p_M^q(q.SR)} \quad (17)$$

For $\forall q \in \mathcal{Q}$, the variables in Equation (15) are $q.SR$ and $q.\theta_k$. To minimize Equation (15), we employ the incremental estimation algorithm to compute the optimal $q.\theta_k$ and the corresponding $q.SR$.

To accommodate our extended search range with the objects processing algorithm and index construction and maintenance algorithm, we replace $q(\mathcal{O})_k$ with $q(\mathcal{O})$ and replace $q.SR_k$ with $q.SR$.

4. Experiments

In this section, we conduct a set of comprehensive experiments to evaluate the efficiency and scalability of the key techniques. Section 4.1 introduces the experimental environment. Section 4.2 evaluates the effect of three tuning parameters and the re-evaluation technique. Section 4.3 evaluates the efficiency and scalability of our index techniques.

4.1. Experimental Settings

All experiments are implemented in VC++, and run on a Win10 machine with an Intel I7-8700K 3.7 GHz CPU and 32 GB memory. In accordance with previous works (e.g., [2–12]), we load the query indexes into main memory to support real-time response.

Datasets. Three datasets are collected for experimental evaluations. The statistics are shown in Table 2. TWEETS contains twitters collected from Twitter [8]. TWEETS is the default dataset. GN is obtained from the US Board on Geographic Names, in which each record contains a geo-location and some terms (<http://geonames.usgs.gov/>). GOWALLA is a synthetic dataset, in which each record contains a geo-location collected from the Gowalla (<https://snap.stanford.edu/data/loc-gowalla.html>), and less than 50 terms randomly assigned from 20 Newsgroups (<http://people.csail.mit.edu/jrennie/20Newsgroups>). Based on the datasets, we generate queries and objects.

Table 2. Datasets statistics.

Datasets	TWEETS	GN	GOWALLA
Size of dataset	20 M	2.29 M	644.3 K
Vocabulary size	1.80 M	202.4 K	61.2 K
Average number of keywords in objects	9	4	26

Query Workload. For each sample dataset, we take the geo-location as the geo-location of the query and randomly select j terms of the sample data as the query keywords, where $1 \leq j \leq 5$. The number of returned kNN results k is set to a default value. At any timestamp, the expired queries are randomly selected.

Object Workload. For each sample dataset, we take all terms as the object keywords, and take the geo-locations deviating from the original geo-location by 0.01% to 1% of the maximum distance in the region. At any timestamp, the expired objects are randomly selected.

Set of Queries and Objects. For each dataset, unless otherwise specified, we select 5 M objects and queries to construct the query index and object index initially, and generate three test sets, each of which contains 2 M objects and 2 M queries. The evaluation criteria take the average performance of three test sets.

Baseline. We compare our index techniques with IQ-tree [1], Ap-tree [3] and FAST [6]. By default, for Ap-tree, the fanout, partition threshold, and KL-Divergence threshold are set to 200, 40, and 0.001. We use the number of verifications to replace the number of I/O in the cost model of IQ-tree. In the following sections, we use AOIQ-tree to represent the index integrated the quadtree with the ordered, inverted index, and three key techniques. We compare the cost-based k-skyband technique with the Kmax [16] when they are integrated in the AOIQ-tree.

Evaluation criteria. We report four criteria: (1) the index construction time (i.e., ICT), i.e., the time of inserting queries into index after finding their search range; (2) the average incoming object processing time (i.e., $AOPT$), i.e., the time of finding the affected queries and modifying their corresponding parameters when an object arrives; (3) the average index updating time caused by objects (i.e., $AIUT$), i.e., the time of updating query index after processing objects; (4) index size, i.e., the memory used for constructing the query index. By default, the number of keywords for constructing ordered, inverted index m , the number of kNN results returned for CkQST k , the height of the quadtree θ_h , the ratio of the update operation to the verification operation θ_U , and the number of object updates within unit time interval $freq_U^o$ are set to 2, 20, 10, 0.001, and 20,000.

4.2. Experimental Tuning

In this section, a series of experiments are conducted to evaluate the effect of parameters in techniques on the AOIQ-tree.

Effect of m . Figure 4 shows the evaluation criteria of the AOIQ-tree when m takes 1, 2, 3, 4, and 5. According to Lemma 1, if m is small, the number of queries in posting lists is large, so it takes a long time to verify queries in posting lists; contrarily, if m is large, it takes a long time to find the posting lists to be verified. Therefore, the optimal m is neither too small nor too large. As shown in Figure 4, when m takes 2, the performance of the index is the best. The larger m is, the larger the index size. When $m > 3$, the verification cost and update cost in a single posting list decrease, so the cost model maps queries to nodes with large regions, so ICT , $AIUT$ and index size decrease, while $AOPT$ increases.

Effect of θ_U . Figure 5 shows the evaluation criteria of the AOIQ-tree when θ_U takes 0.0001, 0.001, 0.01, and 0.1. When θ_U takes 0.0001, the verification cost plays a major role in finding the associated nodes, therefore, queries are associated with many small nodes, so ICT is long, $AOPT$ and $AIUT$ are short, and index size is large. When θ_U increases, the index update cost is more important, so queries are associated with fewer larger nodes, so ICT and index size decrease, while $AOPT$ and $AIUT$ increase.

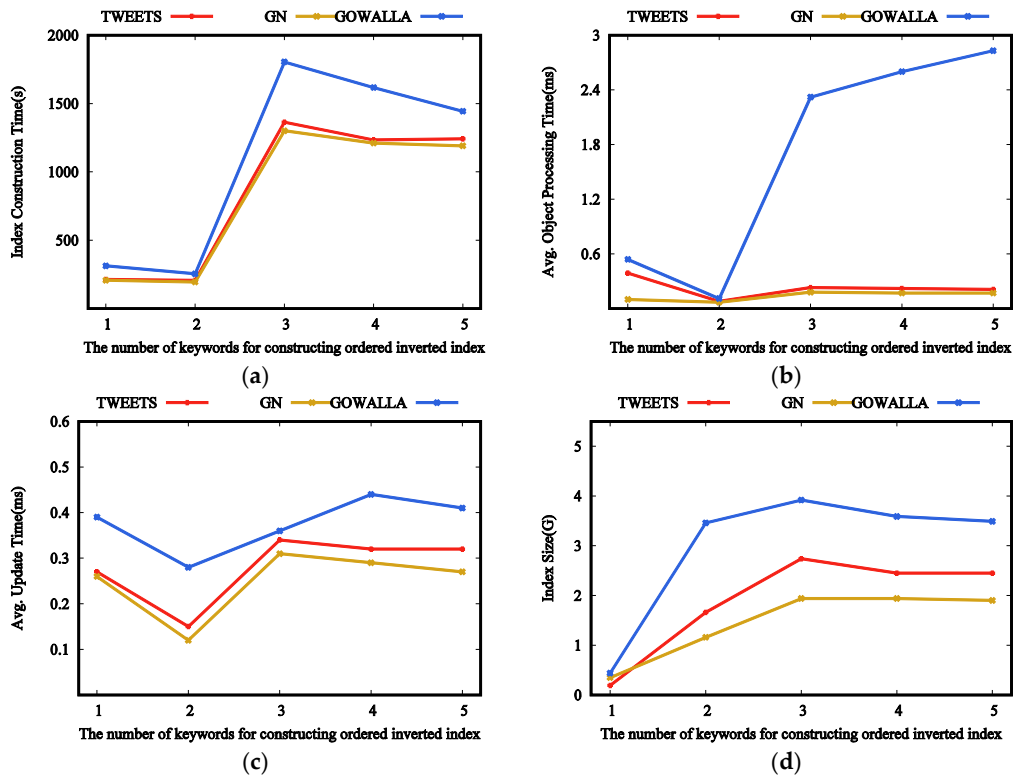


Figure 4. Effect of the number of keywords on constructing the ordered, inverted index: (a) index construction time; (b) average object processing time; (c) average update time; (d) index size.

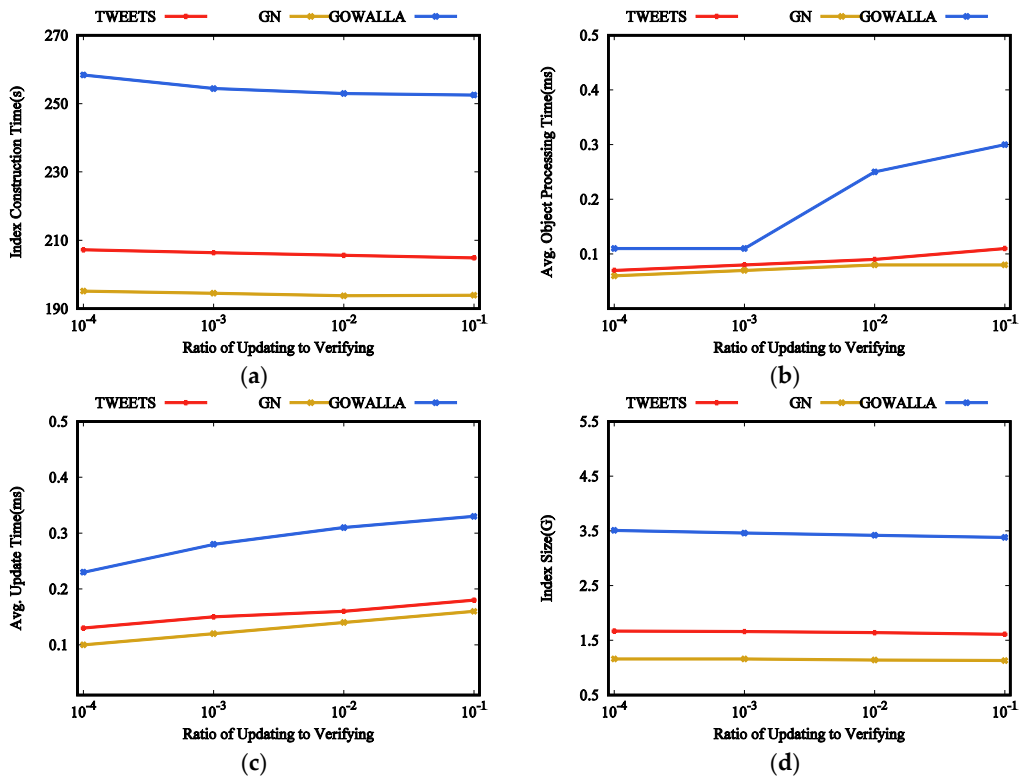


Figure 5. Effect of the ratio of the update operation to the verification operation: (a) index construction time; (b) average object processing time; (c) average update time; (d) index size.

Effect of k . Figure 6 shows the evaluation criteria of the AOIQ-tree when k takes 10, 20, 30, 40, and 50. As k is small, the number of returned objects for queries is few, the search range of queries is small, and queries are associated with many small nodes, so the ICT is long, $AOPT$ and $AIUT$ are short, and index size is large. On the contrary, when k is large, the search range of the queries becomes larger, and queries are associated with few larger nodes, the ICT and index size decrease, and the $AOPT$ and $AIUT$ increase.

Effect of re-evaluation techniques. We evaluate the re-evaluation performance in three cases, denoted as AOIQ-tree, AOIQ-tree_Kmax, and AOIQ-tree_Skyband. AOIQ-tree only keeps k objects in the result list, AOIQ-tree_Kmax keeps $k_{\max} = 2k$ objects in the result list, and the number of objects in the result list for AOIQ-tree_Skyband is calculated according to the cost-based k -skyband technique. Figure 7 shows the ICT , $AOPT$, and $EOPT$ in three cases with varied k , where $EOPT$ is the average processing time for expired objects, i.e., the average time of modifying the parameters of the affected queries or re-evaluating the queries if the number of objects in their result list is less than k when an object expires. The number of objects maintained in AOIQ-tree_Kmax is more than AOIQ-tree_Skyband, which is more than AOIQ-tree. The ICT of AOIQ-tree_Kmax is shortest, and that of AOIQ-tree is longest. The $AOPT$ of AOIQ-tree and AOIQ-tree_Skyband are shorter than that of AOIQ-tree_Kmax. The $EOPT$ of AOIQ-tree_Kmax and AOIQ-tree_Skyband are shorter than that of AOIQ-tree. This phenomenon is related to the numbers of objects maintained in their result list. Compared with AOIQ-tree, the $EOPT$ of the other two techniques are much less, and if k takes 10, 20, the average update time is close to 0.

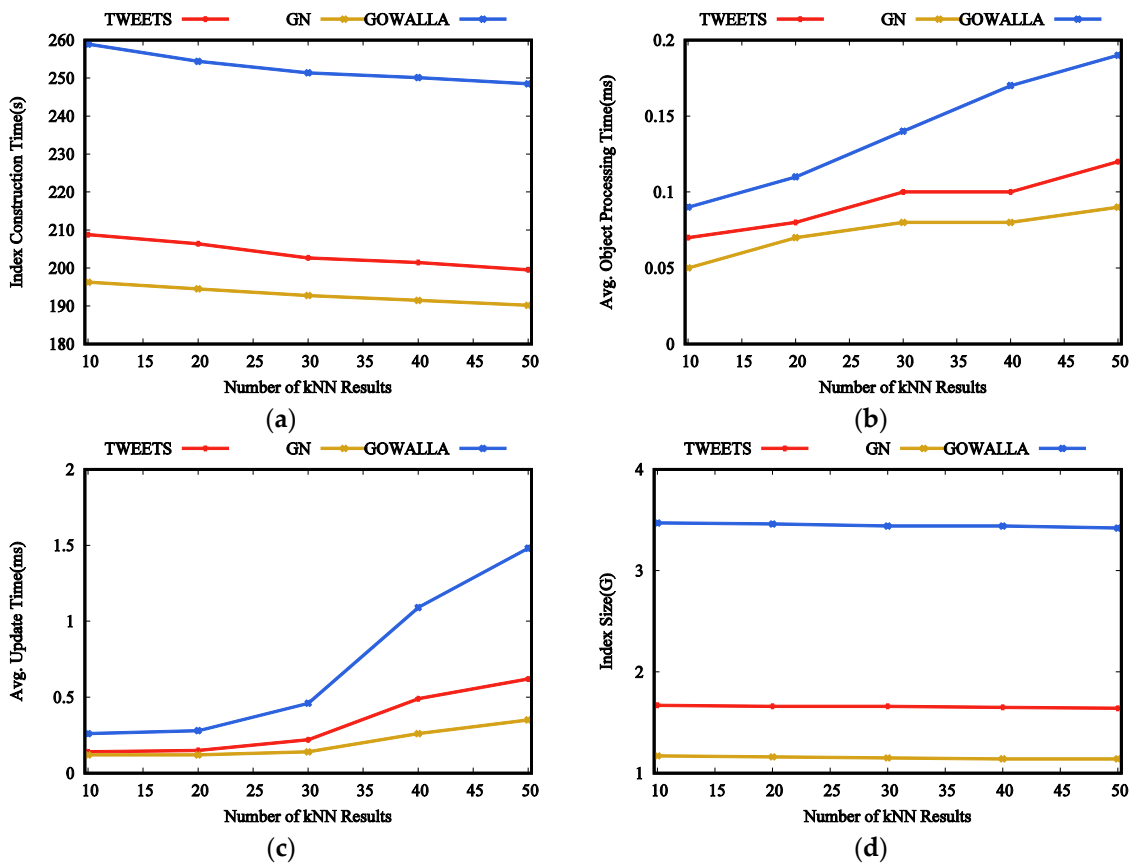


Figure 6. Effect of the number of kNN results returned for CkQST: (a) index construction time; (b) average object processing time; (c) average update time; (d) index size.

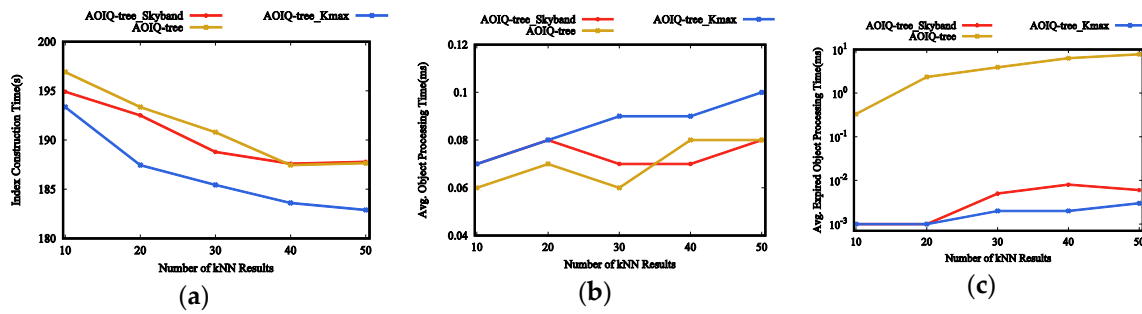


Figure 7. Effect of the number of kNN results returned for CkQST: (a) index construction time; (b) average object processing time; (c) average expired object processing time.

4.3. Performance Evaluation

Evaluation on different datasets. We evaluate the efficiency of the key techniques against three datasets in Figure 8. The number of queries is 5 M. As shown in Figure 8, besides the index size, AOIQ-tree is always the best one. IQ-tree has a good spatial filtering performance, but its textual filtering ability is weak; AP-tree comprehensively considers the spatial and textual distribution of queries, but the index construction and update cost are expensive; FAST has a good textual filtering performance, but its spatial filtering ability is weak. The memory-based cost model in AOIQ-tree can minimize the verification cost and update cost, which makes the number of queries in spatial nodes neither too many nor too few; the ordered, inverted index in AOIQ-tree is constructed by two keywords, which makes the verification cost close to the ordered keyword trie, and the update cost close to the ranked-key inverted index. AOIQ-tree takes up the most memory, which is determined by the structure of its posting lists. The evaluation criteria of GN are the smallest, and Gowalla are the largest, which is because the data in Gowalla contain far more keywords than the other two datasets. For Gowalla, the *AOPT* of AP-tree is shorter than that of FAST. This is because a large number of queries in Gowalla only contain frequent keywords, compared with FAST, AP-tree comprehensively considers the spatial and textual distribution of queries, i.e., index filtering is more powerful, so *AOPT* is shorter.

Effect of number of queries. To evaluate the scalability of the key techniques on the number of queries and objects, we increase the number of queries and objects from 1 M to 20 M to construct the object index and query index. As Figure 9 shows, the *ICT*, *AOPT*, and *AIUT* of all indexes increase as the number of queries increases, and AOIQ-tree is much more scalable. For instance, it only takes 0.23 ms on average to process the incoming objects when the number of queries reaches 20 M, which is 54% faster than Ap-tree and 36% faster than FAST. This shows that our techniques have good scalability. The *AIUT* of AOIQ-tree is the shortest, and Ap-tree is the longest. That's because AOIQ-tree associates queries with optimal nodes to adapt to the objects on data streams, and some of the Ap-tree nodes are re-constructed if many queries updates in these nodes. Compared with AP-tree, only some queries update in FAST nodes, so AOIQ-tree's *AIUT* is shorter.

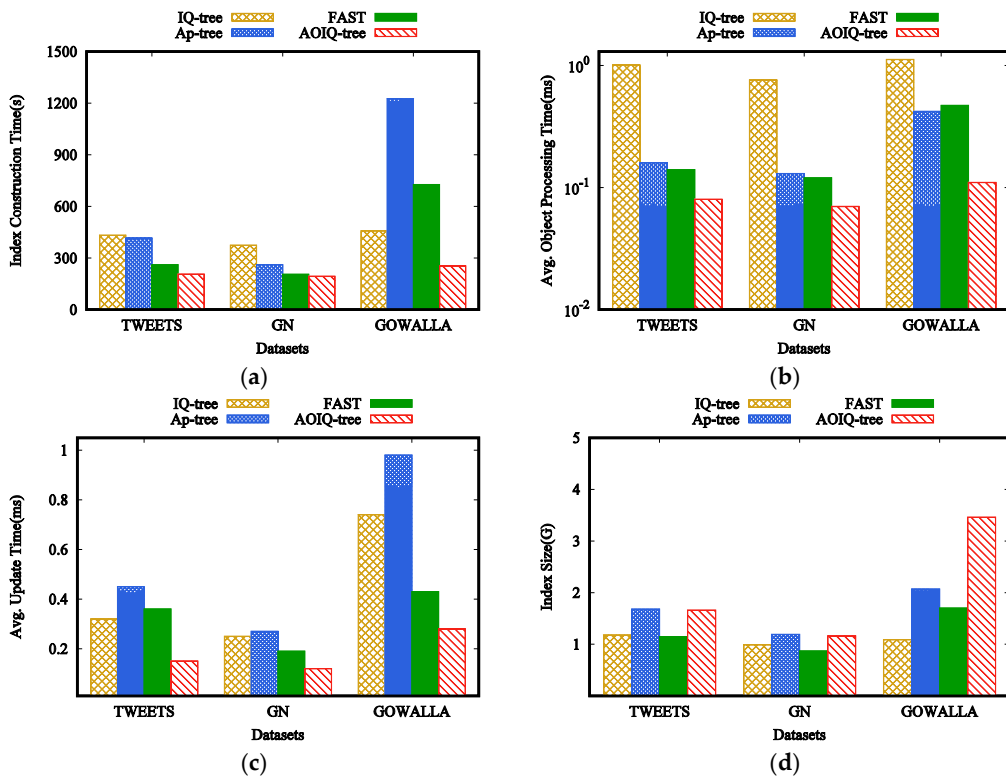


Figure 8. Evaluation of different datasets: (a) index construction time; (b) average object processing time; (c) average update time; (d) index size.

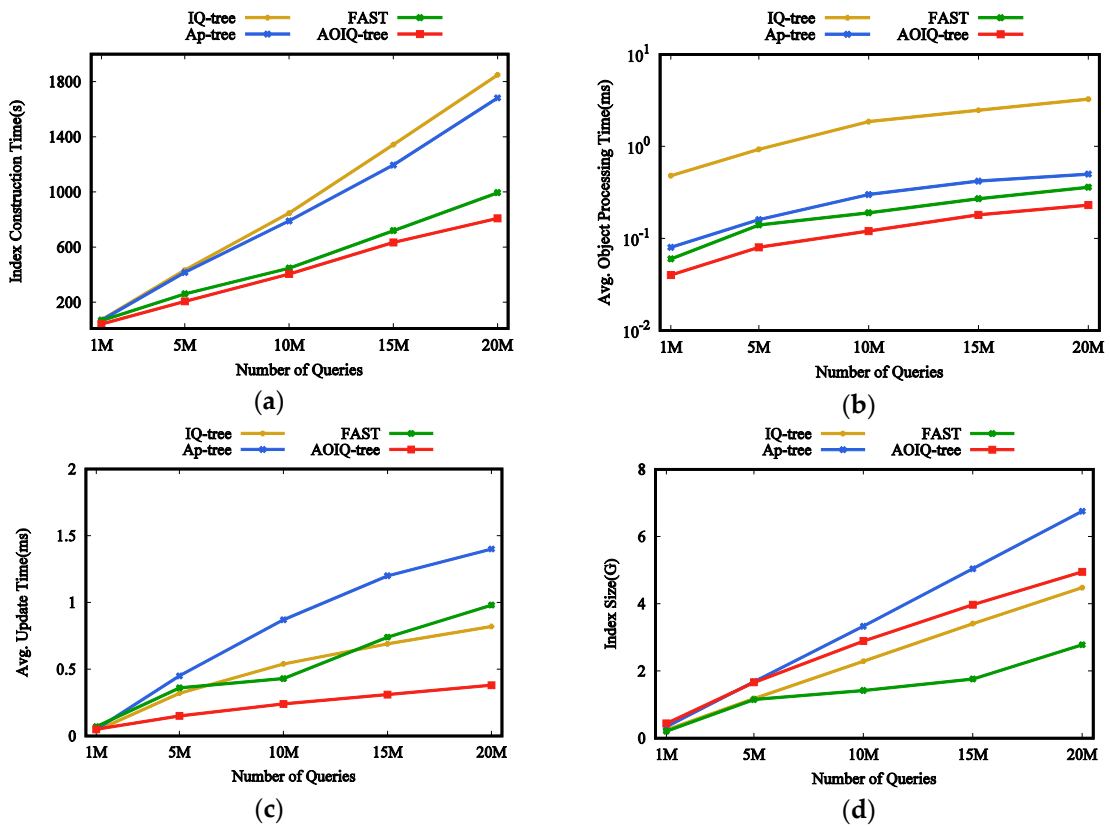


Figure 9. Effect of number of queries: (a) index construction time; (b) average object processing time; (c) average update time; (d) index size.

Effect of number of query keywords $|q.\psi|$. To evaluate the scalability of key techniques on $|q.\psi|$, we increase $|q.\psi|$ from 1 to 5. As shown in Figure 10, the evaluation criteria of IQ-tree are insensitive to the number of keywords since it focuses on the spatial distribution of the queries. Ap-tree, FAST, and our index consider the keyword distribution of the queries, so the evaluation criteria vary with $|q.\psi|$. As $|q.\psi|$ increases, the *ICT* and index size increase, and the *AOPT* decreases. That is because Ap-tree continuously calculates how to partition queries into nodes according to query keywords, and increases the number of textual nodes and height of the tree. For FAST, as $|q.\psi|$ increases, more keywords being attached to posting lists become frequent, and queries are more likely to be inserted into the multiple higher-level nodes. For AOIQ-tree, the time of sorting the queries increases when the ordered, inverted index is constructed.

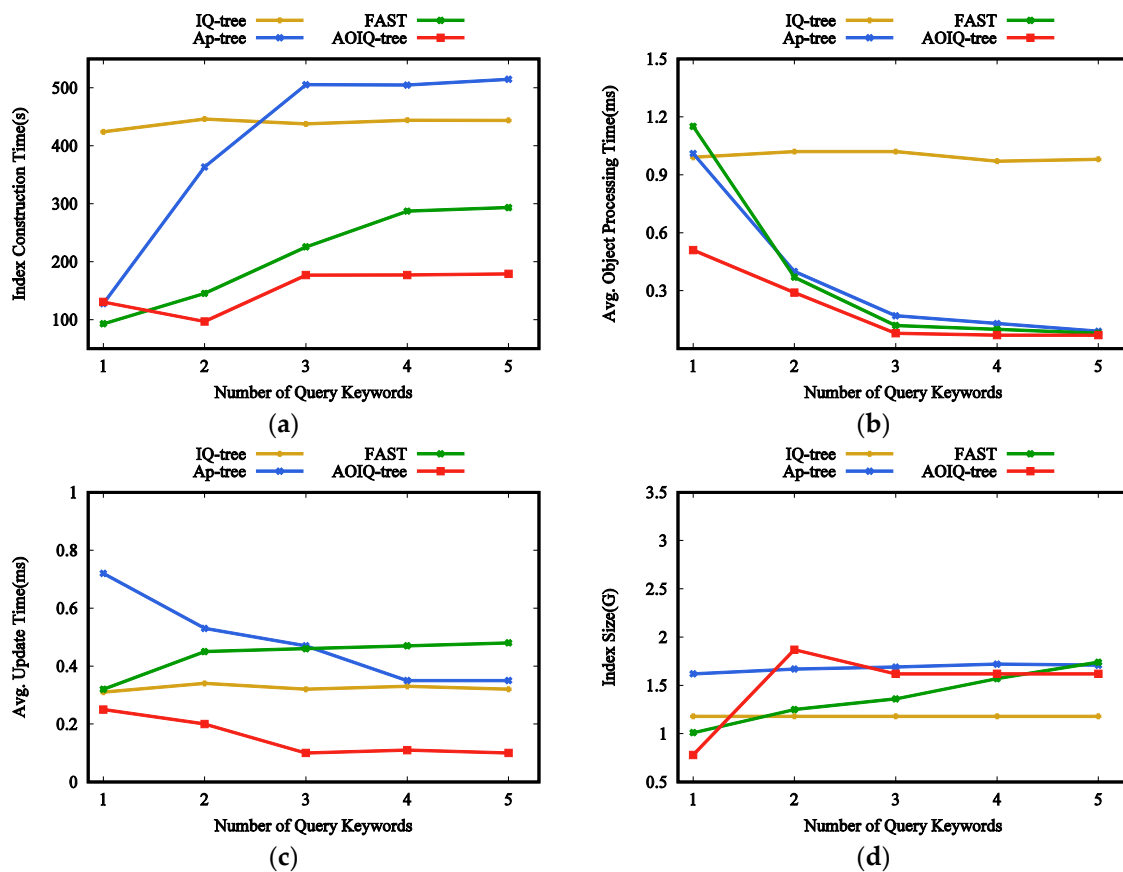


Figure 10. Effect of number of query keywords: (a) index construction time; (b) average object processing time; (c) average update time; (d) index size.

5. Conclusions and Future Research Perspectives

The challenging for evaluating CkQST is how to strike the balance between the filtering ability and the update cost of the spatial–textual index. To address the challenging, we use quadtree and inverted index to organize millions of CkQST with three techniques. A memory-based cost model maps the search range of CkQST to the quadtree nodes to balance the spatial filtering ability of the indexes and the cost for updating the indexes. The balance can be further tuned by the cost-based k-skyband technique, which judiciously determines the search range for CkQST according to the workload of objects. An adaptive block-based ordered, inverted index enhances the textual filtering ability. The experimental results on the real-world and synthetic datasets show that the proposed techniques are effective and scalable, and can significantly improve the evaluation efficiency of CkQST. The future work for evaluating the continuous query over spatial–textual data streams includes solving the challenges of continuous queries in mobile and other relevant scenarios, and exploring efficient

evaluation techniques using hardware technologies such as Graphics Processing Unit and distributed clusters and trade-off strategy for precision and evaluation efficiency.

Author Contributions: Rong Yang proposed the methods, implemented the algorithms for the experiments, and wrote the manuscript; Baoning Niu provided suggestions for the methods and experiments, reviewed and modified the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (Grant No. 62072326), National Key Research and Development Plan of Shanxi Province (Grant No. 201903D421007).

Acknowledgments: The authors would like to thank three anonymous reviewers for their valuable suggestions and comments.

Conflicts of Interest: The authors declare that they have no conflict of interest.

Appendix A

In the proof of Lemma 1, without loss of generality, we suppose that all the terms in $o.\psi$ are contained in different blocks and divide the object processing at quadtree nodes into three steps: finding the posting lists to be verified, finding the blocks to be verified in all posting lists, and finding the queries to be verified in all blocks.

Proof of Lemma 1. If $|o.\psi| = 1$, let $o.\psi = \{w_{i1}\}$. The object is verified with the queries in the posting list determined by $\{w_{i1}\}$, the verification cost is $O(\log|\mathcal{V}|^m + |Q|)$, where Q contains these queries whose query keyword are $\{w_{i1}\}$. If $|o.\psi| = 2$, let $o.\psi = \{w_{i1}, w_{i2}\}$, the object is verified with the queries in the posting list determined by $\{w_{i1}\}$, $\{w_{i2}\}$, and $\{w_{i1}, w_{i2}\}$, the verification cost is $O(3\log|\mathcal{V}|^m + |Q|)$, where Q contains these queries whose query keywords are $\{w_{i1}\}$, $\{w_{i2}\}$, or $\{w_{i1}, w_{i2}\}$. Specifically, if $m = 1$, the object is verified with the posting lists determined by $\{w_{i1}\}$ and $\{w_{i2}\}$. For the posting list determined by $\{w_{i1}\}$, the object is verified with two blocks. One block contains queries whose keywords are $\{w_{i1}\}$, and the other contains queries whose keywords may contain $\{w_{i1}, w_{i2}\}$. The verification cost is $O(2\log|\mathcal{V}| + \log|\mathcal{B}| + |b| + |Q|)$. If $|o.\psi| \geq 3$, the posting lists that the object is verified with can be divided into three categories. First, the posting lists are determined by less than m terms. The verification cost is $\sum_{i=1}^{m-1} \binom{i}{|o.\psi|} \cdot (\log|\mathcal{V}|^m + |Q|)$; second, the posting lists are determined by m terms and one of the term is $|o.\psi|$. The verification cost is $\binom{m-1}{|o.\psi|-1} \cdot (\log|\mathcal{V}|^m + |Q|)$. Third, the posting lists are determined by m terms and the terms do not contain $|o.\psi|$. The verification cost is $O\left(|o.\psi|^m \cdot \left(\log|\mathcal{V}|^m + |o.\psi| \cdot \left(\log|\mathcal{B}| + \frac{|b|}{|\mathcal{B}|^{m-1} \cdot |b \cdot \psi|^{m-1}}\right)\right) + |Q|\right)$, where Q contains these queries that contain less than or equal to m keywords and these keywords are contained in o . The Lemma is proved.

References

1. Chen, L.S.; Cong, G.; Cao, X. An efficient query indexing mechanism for filtering geo-textual data. In Proceedings of the 32nd ACM SIGMOD International Conference on Management of Data (SIGMOD'13), New York, NY, USA, 22–27 June 2013; ACM Press: New York, NY, USA, 2013; pp. 749–760.
2. Li, G.L.; Wang, Y.; Wang, T.; Feng, J.H. Location-aware publish/subscribe. In Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD'13), Chicago, IL, USA, 11–14 August 2013; pp. 802–810.
3. Wang, X.; Zhang, Y.; Zhang, W.J.; Lin, X.M.; Wang, W. AP-Tree: Efficiently support location-aware publish/subscribe. *VLDB J.* **2015**, *24*, 823–848. [[CrossRef](#)]
4. Deng, Z.; Wang, M.; Wang, L.Z.; Huang, X.H.; Han, W.; Chu, J.D.; Zomaya, A.Y. An efficient indexing approach for continuous spatial approximate keyword queries over geo-textual streaming data. *Int. J. Geo-Inf.* **2019**, *8*, 57. [[CrossRef](#)]

5. Guo, L.; Zhang, D.X.; Li, G.L.; Tan, K.-L.; Bao, Z.F. Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In Proceedings of the 34th ACM SIGMOD International Conference on Management of Data (SIGMOD'15), Melbourne, Australia, 31 May–4 June 2015; ACM Press: New York, NY, USA, 2015; pp. 843–857.
6. Mahmood, A.R.; Aly, A.M.; Aref, W.G. FAST: Frequency-Aware Indexing for Spatio-Textual Data Streams. In Proceedings of the 34th IEEE International Conference on Data Engineering (ICDE'18), Paris, France, 16–19 April 2018; IEEE Press: Piscataway, NJ, USA, 2018; pp. 305–316.
7. Hu, H.; Liu, Y.; Li, G.; Feng, J.; Tan, K.L. A location-aware publish/subscribe framework for parameterized spatio-textual subscriptions. In Proceedings of the 31st IEEE International Conference on Data Engineering (ICDE'15), Seoul, Korea, 13–17 April 2015; IEEE Press: Piscataway, NJ, USA, 2015; pp. 711–722.
8. Chen, L.; Cong, G.; Cao, X.; Tan, K.L. Temporal spatial-keyword top-k publish/subscribe. In Proceedings of the 31st IEEE International Conference on Data Engineering (ICDE'15), Seoul, Korea, 13–17 April 2015; IEEE Press: Piscataway, NJ, USA, 2015; pp. 255–266.
9. Chen, L.S.; Shang, S. Approximate spatio-temporal top-k publish/subscribe. *World Wide Web* **2019**, *22*, 2153–2175. [[CrossRef](#)]
10. Wang, X.; Zhang, W.J.; Zhang, Y.; Lin, X.M.; Huang, Z.F. Top-k spatial-keyword publish/subscribe over sliding window. *VLDB J.* **2017**, *26*, 301–326. [[CrossRef](#)]
11. Chen, Z.D.; Cong, G.; Zhang, Z.J.; Fu, T.Z.J.; Chen, L.S. Distributed Publish/Subscribe Query Processing on the Spatio-Textual Data Stream. In Proceedings of the 33rd IEEE International Conference on Data Engineering (ICDE'17), San Diego, CA, USA, 19–22 April 2017; IEEE Press: Piscataway, NJ, USA, 2017; pp. 1095–1106.
12. Mahmood, A.; Daghistani, A.; Aly, A.M.; Tang, M.J. Adaptive processing of spatial-keyword data over a distributed streaming cluster. In Proceedings of the 21st ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL'18), Seattle, WA, USA, 6–9 November 2018; ACM Press: New York, NY, USA, 2018; pp. 219–228.
13. Böhm, C.; Ooi, B.C.; Plant, C.; Yan, Y. Efficiently processing continuous k-NN queries on data streams. In Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE'07), Istanbul, Turkey, 15–20 April 2007; IEEE Press: Piscataway, NJ, USA, 2007; pp. 156–165.
14. Xiong, X.P.; Mokbel, M.F.; Aref, W.G. SEA-CNN: Scalable processing of continuous k-nn Queries in spatio-temporal databases. In Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE'05), Tokyo, Japan, 5–8 April 2005; IEEE Press: Piscataway, NJ, USA, 2005; pp. 643–654.
15. Yu, X.H.; Pu, K.Q.; Koudas, N. Monitoring k-nearest neighbor queries over moving objects. In Proceedings of the 21st IEEE International Conference on Data Engineering (ICDE'05), Tokyo, Japan, 5–8 April 2005; IEEE Press: Piscataway, NJ, USA, 2005; pp. 631–642.
16. Yi, K.; Yu, H.; Yang, J.; Xia, G.; Chen, Y. Efficient maintenance of materialized top-k views. In Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE'03), Bangalore, India, 5–8 March 2003; IEEE Press: Piscataway, NJ, USA, 2003; pp. 189–200.
17. Mouratidis, K.; Bakiras, S.; Papadias, D. Continuous monitoring of top-k queries over sliding windows. In Proceedings of the 25th ACM SIGMOD International Conference on Management of Data (SIGMOD'06), Portland, OR, USA, 27–29 June 2006; ACM Press: New York, NY, USA, 2006; pp. 635–646.
18. Zhang, C.Y.; Zhang, Y.; Zhang, W.J.; Lin, X.M. Inverted linear Quadtree: Efficient top k spatial keyword search. *IEEE Trans. Knowl. Data Eng.* **2016**, *28*, 1706–1721. [[CrossRef](#)]
19. Microsoft Ignite. Available online: <https://docs.microsoft.com/zh-cn/cpp/standard-library/map-class?view=vs-2019> (accessed on 10 September 2020).

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).