This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



# INSPIRE: A Framework for Incremental Spatial Prefix Query Relaxation

Yuxin Zheng, Zhifeng Bao, Lidan Shou, and Anthony K.H. Tung

**Abstract**—Geo-textual data are generated in abundance. Recent studies focused on the processing of spatial keyword queries which retrieve objects that match certain keywords within a spatial region. To ensure effective retrieval, various extensions were done including the allowance of errors in keyword matching and autocompletion using prefix matching. In this paper, we propose INSPIRE, a general framework, which adopts a unifying strategy for processing different variants of spatial keyword queries. We adopt the autocompletion paradigm that generates an initial query as a prefix matching query. If there are few matching results, other variants are performed as a form of relaxation that reuses the processing done in the earlier phase. The types of relaxation allowed include spatial region expansion and exact/approximate prefix/substring matching. Moreover, since the autocompletion paradigm allows appending characters after the initial query, we look at how query processing done for the initial query and relaxation can be reused in such instances. Compared to existing works which process variants of spatial keyword query as new queries over different indexes, our approach offers a more compelling way to efficient and effective spatial keyword search. Extensive experiments substantiate our claims.

**Index Terms**—Query relaxation, Spatial keyword search, typeahead search, fuzzy search

## 1 INTRODUCTION

As the growth in geographical applications like Google Earth and Foursquare, geo-textual data are generated in abundance. To retrieve such data effectively, recent studies focused on the processing of spatial keyword queries which retrieve objects that match certain keywords within a spatial region. In order to ensure effective and user-friendly retrieval, a popular paradigm is to support search-as-you-type as illustrated in Example 1.

**Example 1.** *In Figure 1a, a user wants to search for "Staples Center". He zooms in the region of Staples Center and types "staples ce". Object A is returned because it is in the viewport and starts with "staples ce" in its text.*

We call this query a Spatial Prefix (SP) query [1], and a point of interest (POI) is returned if it is located in the query region and the query text is a prefix of its textual content.

However, it is possible that no or few POIs are returned due to human error: (1) the query range (represented by the viewport) is wrongly specified by the user who is not familiar with the query region; (2) the query text does not satisfy the prefix condition due to typos in the query text or data uploaded by users. Query relaxation must be done to ensure that useful answers are returned. Such relaxation can be done in two ways: (1) expand the spatial region to a larger region; (2) relax the prefix condition to more general textual conditions.


- *Y. Zheng and A.K.H. Tung are with the School of Computing, National University of Singapore. E-mail: {yuxin, atung}@comp.nus.edu.sg*
- *Z. Bao is with School of Engineering & ICT, University of Tasmania, Australia. E-mail: zhifeng.bao@utas.edu.au*
- *L. Shou is with College of Computer Science and Technology, Zhejiang University, China. E-mail: should@zju.edu.cn*


To relax the prefix condition, the first natural choice is to relax it to the *substring query*, which requires that the query text is an *exact substring* of the textual content of an object. A second type of relaxation is performed by allowing mismatches in keyword search but limiting the matching to start at the prefix. This is called *approximate prefix query*. If both of these types of relaxation fail to produce enough number of results, we would then allow approximate string matching to be applied to any part of the textual content. This third type of relaxation is named *approximate substring query*.

**Example 2.** *Figure 1b shows a case of relaxing the spatial constraint. The considered spatial region is enlarged and object B is added to the result. Figures 1c-e show the cases of relaxing the prefix condition. In Figure 1c, the prefix constraint is relaxed to the substring matching and object C is added. In Figure 1d, the constraint is relaxed to the approximate prefix matching and object D is added. If at least three objects are required, the approximate substring matching is applied as in Figure 1e. Objects A, C, D and E are returned.*

As shown in Example 2, different queries may need different types and degrees of relaxation. Determining what relaxation to perform and performing each type of relaxation efficiently are major challenges. While existing works handle some of these types of relaxation as a single query [1], [2], [3], [4], performing all these different types of relaxation means re-issuing a new query every time. This is inefficient especially when these queries are answered using different indexes in different works. Moreover, in the context of search-as-you-type, users can append characters to the initial query even as the initial query is being processed, giving rise to a new query which we refer to as an *appending query*.

**Example 3.** *In an interactive search, the user types two more*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



*characters "nt" to the query in Example 2. The text of the appending query becomes "staples cent". Instead of processing the appending query from scratch, we can use the results of the previous query and start processing it from the last type of relaxation, which is the approximate substring matching. Therefore, objects A, C, D and E are returned.*

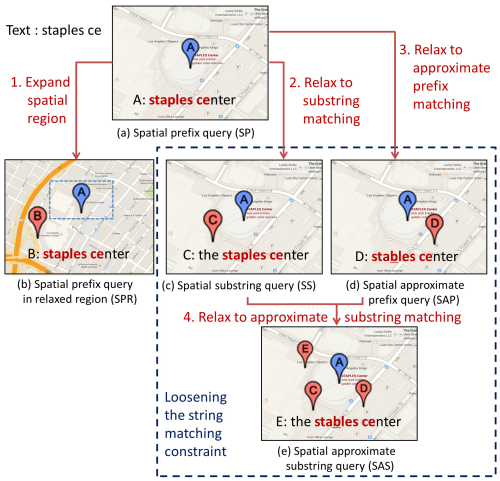



Fig. 1: Spatial Prefix Query and its relaxation

Again, it is obvious that the appending query in the example can be handled by issuing it as a completely new query, but doing so will obviously be not efficient.

In order to provide better solutions for the issues we discussed, we propose a general two-level inverted index that can support all types of relaxation that we have mentioned. The proposed structure has an object-level index to support functionalities and a node-level index to accelerate query processing. Using this two-level index, we propose several variants of the string filtering techniques to effectively remove data objects that do not satisfy the query condition.

Having observed that different types of relaxation have different levels of complexity and have dependencies among themselves, we propose INSPIRE, an INcremental Spatial PrefIx query RElaxation framework, to process the spatial prefix query and its relaxation. Similar to the strategy shown in Figure 1, the initial query is a spatial prefix query. If no or few results are returned, we incrementally process different types of relaxation until sufficient results are returned. In particular, the spatial region of the prefix query is first expanded, followed by relaxation of different degrees on the string dimension: the substring query, the approximate prefix query and the approximate substring query.

By ordering these types of relaxation in an appropriate order, we can use the dependencies between them to optimize our query processing through the reuse of previous results. To further reduce the computational cost, we adapt selectivity estimation techniques to determine whether certain relaxation should be conducted. If the estimated selectivity does not meet a given threshold, we can simply go to the next-level relaxation. We will refer to the intermediate result reuse and selectivity estimation as *intra-query optimization*.

If a query is an appending query that is formed from a previous query by appending characters, we also look into how processing can be reused in such cases. In particular, we notice that a time-consuming process for answering a query is to merge inverted lists to obtain candidate results. We thus further optimize the processing of merging lists in appending queries. This is formalized as our *inter-query optimization*.

Our main contributions are summarized as follows:

1. We first identify the relationships among different types of relaxation for the spatial prefix query, and then propose a framework to incrementally process the relaxation for a spatial prefix query (in Section 2).
2. We build a one-size-fits-all index (in Section 3) to support all types of relaxation (in Section 4).
3. During the incremental relaxation paradigm, we propose various result reuse methods to accelerate the processing of both non-appending and appending queries, namely intra-query optimization (in Section 5) and inter-query optimization (in Section 6).
4. We conduct a comprehensive experimental study over three real data sets to demonstrate the efficiency and effectiveness of our methods (in Section 7).

## 2 The INSPIRE Framework

Our objective is to build a one-size-fits-all search engine for spatial prefix query and its various types of relaxation. In this section, we first introduce the preliminaries for such a search engine. Then we formally state our problem and introduce our INSPIRE framework.

### 2.1 Preliminary

**Hilbert-encoded Quadtree:** Quadtree [5] is widely adopted to facilitate fast retrieval of objects in multi-dimensional space. In a two-dimensional Quadtree, each node represents a bounding box covering a part of the space. The root node covers the whole space, and the leaf node contains indexed points, while the internal node has four children, one for each quadrant obtained by dividing the area covered in half along both axes.

A Hilbert curve [6] is a space-filling curve, which maps multi-dimensional data to one dimension. The Hilbert curve is defined recursively: each cell is divided into four subcells at each subsequent level. The sequential order of a cell in a curve is identified by its Hilbert code.

**Definition 1** (Hilbert Code ($hc$)). *Given a Quadtree node $n$, its Hilbert code $hc$ is a concatenation of the Hilbert code of its parent node and $n$'s local order. Hilbert code $hc_1$ is defined to be ahead of $hc_2$, if $hc_1$ has a smaller value at the first different bit of $hc_1$ and $hc_2$.*

**Example 4.** *The Hilbert curve of level-two in Figure 3 is constructed from the Hilbert curve of level-one in Figure 2. When the local order is defined in range $[0, 3]$, the Hilbert codes of the nodes in the left-upper corner are 1.0, 1.1, 1.2 and 1.3 respectively. Node 1.0 is ahead of node 1.1.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



We integrate a Hilbert curve with a Quadtree as in [7]. As a result, we sequentialize the cells of a Quadtree to efficiently retrieve places located in a given spatial region. We call this a *Hilbert-encoded Quadtree*, on which we will build our novel index to handle the spatial prefix query and its various types of relaxation.

**Example 5.** *Figure 4 shows a Hilbert-encoded Quadtree of eleven objects. A Quadtree is first built. The maximum capacity of each leaf node is set to two. Then a Hilbert curve is built to link the nodes of the Quadtree, which is plotted as the dotted curve. Each leaf node is attached with a Hilbert code to represent its sequential order in the Hilbert curve.*

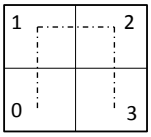



Fig. 2: Level-one

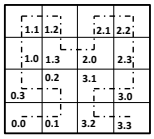



Fig. 3: Level-two

**String Filtering:** The approximate search is used to handle cases when queries have errors. Most works build solutions based on the concept of *q-grams* [8], [9], [10], [11].

**Definition 2** ((Positional) $q$-gram). *A q-gram $tk$ is a sequence of q characters in string $s$. A positional q-gram (tk,p) is a pair of q-gram $tk$ with its position $p$ in string $s$.*

To support approximate string queries, Gravano et al. proposed an efficient solution for edit similarity join [8]. The edit distance constraint is relaxed to a weaker constraint based on the number of matching $q$-grams.

**Definition 3** (Matching (Positional) $q$-gram). *A q-gram $tk_1$ matches $tk_2$ if $tk_1 = tk_2$. Given an edit distance threshold $\tau$, a positional q-gram $(tk_1, p_1)$ matches $(tk_2, p_2)$ if $tk_1 = tk_2$ and $|p_1 - p_2| \leq \tau$.*

However, one problem of [8] is that a string candidate pair cannot be discarded unless all their $q$-grams have been compared. To quickly filter out candidate pairs, a prefix-based filtering is proposed [9]. We summarize some widely adopted filters in the approximate string search as follows.

**Definition 4** (Filters of String Approximation). *Given strings $s$, $t$, and an edit distance threshold $\tau$, the following filters are used to check whether the edit distance between $s$ and $t$ is within $\tau$.*
*(1) Length Filtering (**LF**) [8]: $||s| - |t|| \leq \tau$.*
*(2) Count Filtering (**CF**) [8]: $s$ and $t$ must share at least $LB_{s,t,\tau,q}$ = $(\max(|s|, |t|) - q + 1) - q * \tau$ matching q-grams.*
*(3) Position Filtering (**PoF**) [8]: $s$ and $t$ must share at least $LB_{s,t,\tau,q}$ matching positional q-grams.*
*(4) Prefix Filtering (**PrF**) [9]: the first $(q * \tau + 1)$ positional q-grams of $s$ and $t$ must have at least one match.*

## 2.2 Problem Statement

The problem we are going to address is to efficiently answer the spatial prefix query and its various types of relaxation. It can be reduced to two problems: (1) To design an incremental relaxation paradigm, and the relaxation is triggered if no or few results are returned. (2) To provide the search-as-you-type feature for the interactive search.

TABLE 1: Table of Notations

| | | | |
|---|---|---|---|
| $n$ | a node of a Quadtree | $O$ | a spatial object |
| $n.hc$ | the hilbert code of $n$ | $p\top$ | reserved position |
| $\theta$ | result size threshold | $\tau$ | edit distance threshold |
| $\omega$ | string query type | $\Re$ | prefix ratio |
| $Q'_\omega$ | appending query | $Q_\omega$ | previously-formed query |
| $Res(Q_\omega)$ | result set of $Q_\omega$ | $\epsilon$ | scarce threshold |
| $S(Q_\omega)$ | estimated selectivity of $Q_\omega$ | | |
| $S_\omega(n, s)$ | estimated selectivity of string $s$ in $n$ for string type $\omega$ | | |
| $q_c$ ($q_p$) | $q$ value for $q$-gram (positional $q$-gram) | | |
| $NS(n, Q_\omega)$ | node snippet of node $n$ with respect to $Q_\omega$ | | |
| $LB_{s,\tau,q}$ | min match threshold, $(\|s\| - q + 1) - q * \tau$ | | |
| $UB_{p\top,\tau,q}$ | length bound, $(p\top - \tau + q - 1)$ | | |

The frequently used notations are summarized in Table 1. Each spatial object $O$ is defined as a tuple ($O.id$, $O.loc$, $O.str$), where $O.id$ is an identification, $O.loc$ = $(lat, lng)$ the latitude and longitude, and $O.str$ the textual content. A spatial prefix query is defined as:

**Definition 5** (Spatial Prefix Query). *A Spatial Prefix (SP) query $Q_p$ = (rng, str) consists of two parts: (1) the spatial region $Q_p.rng$, which can be retrieved from the user's viewport; (2) the query string $Q_p.str$.*

A spatial object $O$ is an answer to a SP query $Q_p$ if: (1) $O.loc$ is in $Q_p.rng$, and (2) $Q_p.str$ is a prefix of $O.str$. When a spatial prefix search returns no or few results, we relax the query in order to obtain sufficient results. The spatial prefix query can be relaxed in two ways: (1) relax the spatial constraint, or (2) relax the prefix constraint.

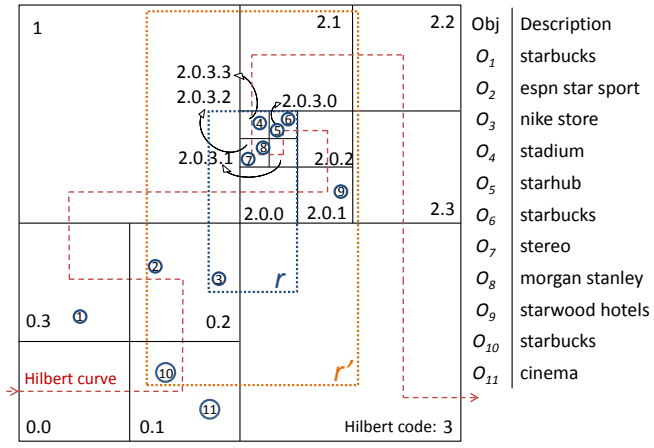



Fig. 4: Hilbert-encoded Quadtree of 11 objects

**Definition 6** (Relaxation of Spatial Region). *Given an SP query $Q_p$ = (rng, str), the relaxation on the spatial region leads to a Spatial Prefix query in a Relaxed region (SPR) $Q^r_p$ = (rng', str), where $Q_p.rng \subset Q^r_p.rng'$.*

**Definition 7** (Relaxation of Prefix Constraint). *Given an SP query $Q_p$ = (rng, str), the relaxation on the prefix constraint leads to a relaxed query $Q_\omega$ = (rng, str, ($\tau$)), where $\omega$ is one of the following string queries: exact substring query (s), approximate prefix query (ap) [11] and approximate substring query (as) [12]. $\tau$ is an edit distance threshold for*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



*the latter two types of approximations. Therefore, the three types of relaxation of the prefix constraint are:*
*(1) Spatial Substring (SS) query: $Q_s$.*
*(2) Spatial Approximate Prefix (SAP) query: $Q_{ap}$.*
*(3) Spatial Approximate Substring (SAS) query: $Q_{as}$*

**Example 6.** *Figure 4 shows a spatial database. The textual contents of the objects are shown on the right, while the locations are shown on the left. Table 2 illustrates an example of an SP query, $Q_p$ = (r, "star"), and its relaxation.*

TABLE 2: Spatial Prefix Query and its Relaxation

| Query type | Notation | Query Result |
|---|---|---|
| SP | $Q_p$ = (r, "star") | $O_5, O_6$ |
| SPR | $Q_p^r$ = (r', "star") | $O_5, O_6, O_9, O_{10}$ |
| SS | $Q_s$ = (r, "star") | $O_5, O_6$ |
| SAP | $Q_{ap}$ = (r, "star", 1) | $O_4, O_5, O_6, O_7$ |
| SAS | $Q_{as}$ = (r, "star", 1) | $O_3, O_4, O_5, O_6, O_7, O_8$ |

As reported in [13], users usually input a query letter by letter, and newly typed characters are appended to the previous query forming a new query at any moment. We call this type of query an *appending query*. Processing of appending queries is important to achieve the search-as-you-type paradigm for the interactive search.

**Definition 8** (Appending Query $Q'_\omega$)**.** *Given two queries $Q_\omega$ and $Q'_\omega$. $Q'_\omega$ is an appending query of $Q_\omega$ if*
*(1) $Q_\omega.rng$=$Q'_\omega.rng$, and (2) $Q_\omega.str$ is a prefix of $Q'_\omega.str$.*

**Example 7.** *In the interactive search, the user then issues another query $Q'_p$ = (r, "starbu"), which is an appending query of $Q_p$ = (r, "star"). The answer to $Q'_p$ is $O_6$.*

### 2.3 Framework

We use $Res(Q_\omega)$ to denote the result set of a spatial query $Q_\omega$. For a spatial prefix query and its various types of relaxation, we identify the following relationships:

**Observation 1** (Inter-relaxation Relationships)**.**
*(1) $Res(Q_p) \subseteq Res(Q_p^r)$,*
*(2) $Res(Q_p) \subseteq Res(Q_s) \subseteq Res(Q_{as})$,*
*(3) $Res(Q_p) \subseteq Res(Q_{ap}) \subseteq Res(Q_{as})$*

Besides, we observe the *inter-query relationship* between a previously-formed query and its appending query. Further details will be presented in Section 6.1.

**Observation 2** (Inter-query Relationship)**.**
*(1) $Res(Q_p) \supseteq Res(Q'_p)$, (2) $Res(Q_p^r) \supseteq Res(Q_p'^r)$,*
*(3) $Res(Q_s) \supseteq Res(Q'_s)$, (4) $Res(Q_{ap}) \supseteq Res(Q'_{ap})$,*
*(5) $Res(Q_{as}) \supseteq Res(Q'_{as})$*

The above two containment relationships motivate us to propose an incremental relaxation strategy, called INcremental Spatial PrefIx query RElaxation (INSPIRE). As shown in Figure 5, the INSPIRE framework processes the spatial prefix query in the following way. Given an incoming query $Q'_\omega$, we first check whether $Q'_\omega$ is an appending query of another query $Q_\omega$ that has been processed previously.

**Case 1**: If $Q'_\omega$ is not an appending query, we process it as a new SP query. If the result size of $Q'_p$ is not less than a certain threshold $\theta$, the results are returned. Otherwise, the relaxation is applied incrementally: the SPR query is first processed, followed by the SS, SAP and SAS queries.

**Case 2**: If $Q'_\omega$ is an appending query of $Q_\omega$, we process it from where $Q_\omega$ stops. We first reuse $Res(Q_\omega)$ to form the candidate results of $Q'_\omega$. If sufficient results are found, processing for $Q'_\omega$ stops; otherwise, we further relax $Q'_\omega$ in an incremental way as what we have done in Case 1.

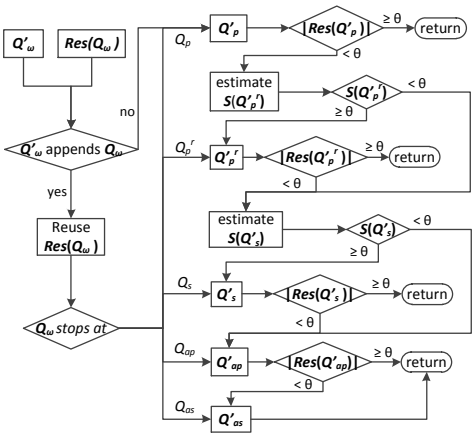



Fig. 5: INSPIRE

In Case 1, the incremental relaxation for a non-appending query may contain several phases, and Observation 1 identifies the commonality among relaxation at different levels. It motivates us to reuse the results of the less relaxed queries (computed in earlier phases) to process the query in the new phase. Furthermore, we propose a method to estimate the selectivity of a query, denoted as $S(Q'_\omega)$, ahead of processing it; as a result, if $S(Q'_\omega)$ is smaller than the threshold, we can skip the relaxation and go to the relaxation at the next level. The reuse and estimation form the *intra-query optimization*, which will be presented in Section 5.

In Case 2, Observation 2 brings two optimization opportunities: (1) We can reuse the results of previously-formed query $Q_\omega$ w.r.t. the appending query $Q'_\omega$. (2) We do not process $Q'_\omega$ from scratch; instead, we process $Q'_\omega$ from where $Q_\omega$ stops. They are formalized as our *inter-query optimization*, which will be presented in Section 6.

## 3 Two-Level Inverted Index

To support INSPIRE, we need a one-size-fits-all index that can answer the spatial prefix query and support the computation of its different types of relaxation. We propose an index structure comprising two parts: a *Hilbert-encoded Quadtree* in the main memory, which is described in Section 2.1, and a two-level inverted index on the disk. We introduce such an index in this section.

### 3.1 Spatial $q$-gram

As shown in Section 2.1, building an inverted index on $q$-grams is a natural choice in string search. To answer a spatial prefix query and its relaxation, we propose a two-level inverted index, which is essentially an adaption of the inverted index from the granularity of $q$-gram to a finer level of granularity called *spatial q-gram*.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



**Definition 9** (Spatial (Positional) $q$-gram). *A spatial $q$-gram $\zeta = (hc, tk)$ contains a Hilbert code $hc$ and a $q$-gram $tk$. A spatial positional $q$-gram $\xi = (hc, tk, p)$ consists of a Hilbert code $hc$ and a positional $q$-gram $(tk, p)$.*

The spatial $q$-gram links the Hilbert code and the $q$-gram to capture the spatial and textual information of an object. By extending the concept of *matching $q$-gram* in Definition 3, the matching spatial $q$-gram is defined and will be used to prune objects that cannot satisfy the textual constraint (see Section 4.1).

**Definition 10** (Matching Spatial $q$-gram). *Spatial $q$-grams $\zeta_1$ and $\zeta_2$ match if $\zeta_1.hc = \zeta_2.hc$ and $\zeta_1.tk = \zeta_2.tk$. Spatial positional $q$-grams $\xi_1$ and $\xi_2$ match w.r.t. a threshold $\tau$, if $\xi_1.hc = \xi_2.hc$, $\xi_1.tk = \xi_2.tk$ and $|\xi_1.p - \xi_2.p| \leq \tau$.*

TABLE 3: Two-level Inverted Index

(a) Object-level Inverted Index

| Spatial 3-gram | IDs |
|---|---|
| (0.2, "sta") | $O_2$ |
| (0.2, "tar") | $O_2$ |
| (2.0.3.0, "sta") | $O_5$, $O_6$ |
| (2.0.3.0, "tar") | $O_5$, $O_6$ |
| ..... | ..... |

| Spatial positional 2-gram | IDs |
|---|---|
| (0.2, "ar", 7) | $O_2$ |
| (0.2, "st", 5) | $O_2$, $O_3$ |
| (0.2, "ta", 6) | $O_2$ |
| (2.0.3.0, "ar", 2) | $O_5$, $O_6$ |
| (2.0.3.0, "st", 0) | $O_5$, $O_6$ |
| (2.0.3.0, "ta", 1) | $O_5$, $O_6$ |
| ..... | ..... |

(b) Node-level Inverted Index

| 3-gram | (Hilbert code, count) list |
|---|---|
| "sta" | (0.1, 1), (0.2, 1), (0.3, 1), (2.0.1, 1), (2.0.3.0, 2),(2.0.3.2, 1), (2.0.3.3, 1) |
| "tar" | (0.1, 1), (0.2, 1), (0.3, 1), (2.0.1, 1),(2.0.3.0, 2) |
| ..... | ..... |

| positional 2-gram | (Hilbert code, count) list |
|---|---|
| ("ar", 2) | (0.1, 1), (0.3, 1), (2.0.1, 1), (2.0.3.0, 2) |
| ("ar", 7) | (0.2, 1) |
| ("st", 0) | (0.1, 1), (0.3, 1), (2.0.1, 1), (2.0.3.0, 2), (2.0.3.2, 1), (2.0.3.3, 1) |
| ("st", 5) | (0.2, 2) |
| ("st", 7) | (2.0.3.2, 1) |
| ("ta", 1) | (0.1, 1), (0.3, 1), (2.0.1, 1), (2.0.3.0, 2), (2.0.3.3, 1) |
| ("ta", 6) | (0.2, 1) |
| ..... | ..... |

### 3.2 Object-level & Node-level Inverted Index

Once the *Hilbert-encoded Quadtree* is constructed, we build a two-level inverted index. To obtain the spatial and textual information of an object, we build an object-level inverted index on spatial $q$-grams by setting a spatial $q$-gram as the key and the objects containing the spatial $q$-gram as the value.

Besides the object-level inverted index, we need to know the number of objects having a particular $q$-gram in a node to estimate the selectivity to optimize the query processing. Thus, we attach a count to each leaf node of the Quadtree to record the number of objects containing a particular $q$-gram in a node. By setting a $q$-gram as the key and a list of (Hilbert-code, count) pairs as the value, we build a coarse-grained index, called node-level inverted index. We will present the selectivity estimation later in Section 5.1.

**Example 8.** *Table 3a illustrates the object-level inverted index of the database in Figure 4. The first entry, (0.2,"sta") $\rightarrow \{O_2\}$, indicates that $O_2$ is located in the leaf node with Hilbert code "0.2" and has a 3-gram "sta". Table 3b shows the node-level inverted index. From the first entry "sta" $\rightarrow$ {(0.1, 1), . . . , (2.0.3.3, 1)}, we know that there is one object located in node "0.1" and containing a 3-gram "sta", etc.*

B+ trees are built on the keys in the inverted index for quick access to index entries. The keys of the inverted index are sorted. For example, the spatial positional $q$-gram is sorted by then Hilbert code, then by the alphabetical order of the $q$-gram and the $q$-gram position. The values of the node-level and object-level inverted indexes are sorted by the Hilbert code of a node and the object ID respectively to support the filters in Section 4.1. Due to space limitations, the index construction and maintenance are presented in the technical report [14].

### 3.3 Combining $q$-gram with positional $q$-grams

Positional $q$-gram is proposed to accelerate the processing of SP, SPR and SAP queries at the expense of large storage space [8], [15], [16]. However, as reported in [17], the average number of words per query is small (2.35) and the number of words per query is smaller than three in most cases (87.4%). Thus, it is not worthwhile to store all positional $q$-grams. Instead, we only store the positional $q$-grams with position less than a certain value, referred to as *reserved position* $p^\top$. The setting of $p^\top$ will be presented in the experiment.

A compromise between storage and efficiency is made for storing less positional $q$-grams. However, it leads to a problem that SS and SAS queries cannot be supported. As compensation, we also store $q$-grams (NOT positional). Compared to the large storage required for positional $q$-grams, the storage of $q$-grams is much smaller. Therefore, we can store $q$-grams with larger $q$ value to achieve better performance. For ease of illustration, we use ($q_p$) $q_c$ to denote the $q$ value for (positional) $q$-grams . As a result, inverted indexes based on positional $q_p$-grams and $q_c$-grams are built as shown in Table 3.

## 4 Query Processing

In INSPIRE, the SP query is first processed. If its result size is smaller than the result size threshold $\theta$, different types of relaxed queries are applied incrementally. In this section, we present how a single query is processed. For processing an appending query, we will present it later in Section 6.

**Algorithm 1** Answering $Q_\omega$

```
1: Intr(Q_ω) ← getIntrNodeSnippet(Q_ω)
2: CanNode ← nodeLevelFilter(Intr(Q_ω), Q_ω)
3: for all n ∈ CanNode do
4:     CanObj ← objectLevelFilter(Obj(n, Q_ω), Q_ω, n)
5:     for all o ∈ CanObj do
6:         if verifyObject(o) then
7:             Result ← Result ∪ o
8: return Result
```

As shown in Algorithm 1, processing a query $Q_\omega$ has two key steps: (1) get the candidate nodes (lines 1-2), and (2) get the results for each candidate node (lines 4-7).

First, we present how to get the candidate nodes. We get the leaf nodes of the Quadtree that intersect with $Q_\omega.rng$, denoted as $Intr(Q_\omega)$ (line 1). Then the *node-level filter* takes $Intr(Q_\omega)$ as the input and removes the nodes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



that cannot satisfy the string constraint (line 2). For each node $n \in Intr(Q_\omega)$, its snippet is retrieved.

**Definition 11** (Node Snippet)**.** *The node snippet of a node $n$ with respect to a query $Q_\omega$, denoted as $NS(n, Q_\omega)$, contains:*
*(1) $n.hc$: the Hilbert code of node $n$.*
*(2) $Obj(n)$: the objects located in node $n$.*
*(3) $Obj(n, Q_\omega)$: the objects located in node $n$ and $Q_\omega.rng$. $Obj(n)$ and $Obj(n, Q_\omega)$ are used to estimate the selectivity of a query, which will be illustrated in Section 5.1.*

Then we show how to get the result objects for each candidate node. For a candidate node $n$, $Obj(n, Q_\omega)$ is retrieved when getting the snippet of $n$. The *object-level filter* of node $n$ takes $Obj(n, Q_\omega)$ as the input and prunes the objects that cannot meet the string constraint (line 4). Finally, the candidate objects are verified whether they truly satisfy the textual constraint (lines 5-7).

### 4.1 Node-level and Object-level Filters

Filters are used at two levels: *nodeLevelFilter*() and *objectLevelFilter*() to filter nodes and objects that cannot satisfy the string constraint respectively. Recall Definition 4, string filters have been proposed in the literature for approximate queries. To cater for the processing of query $Q_\omega$ based on our two-level inverted index, we propose variants of them at the node level and object level respectively. Given a query text $s$ and an edit distance threshold $\tau$, the node-level and object-level filters are:

**Definition 12** (Node-level Filters)**.**
***CF**$_n$: a node $n$ must have at least $LB_{s,\tau,q_c}$ = (|s|-$q_c$+1)-$q_c$* $\tau$ matching $q_c$-grams.*
***PoF**$_n$: a node $n$ must have at least $LB_{s,\tau,q_p}$ matching positional $q_p$-grams.*
***PrF**$_n$: a node $n$ must have at least one matching positional $q_p$-gram in the first ($q_p$* $\tau$+1) positional $q_p$-grams of $s$.*

**Definition 13** (Object-level Filters in Node $n$)**.**
***CF**$_o$: an object $o$ must have at least $LB_{s,\tau,q_c}$ matching spatial $q_c$-grams in node $n$.*
***PoF**$_o$: an object $o$ must have at least $LB_{s,\tau,q_p}$ matching spatial positional $q_p$-grams in node $n$.*
***PrF**$_o$: an object $o$ must have at least one matching spatial positional $q_p$-gram in the first ($q_p$* $\tau$+1) spatial positional $q_p$-grams of $s$ in node $n$.*

Table 4 illustrates the filters used for each type of query. Column 1 is the query type, columns 3 and 2 specify the filters adopted and the condition to trigger such filters, while column 4 specifies the minimum match threshold for a qualified candidate to pass the filter.

In particular, PoF can only be applied when the query length is not greater than $UB_{p^\top,\tau,q_p}$, where $UB_{p^\top,\tau,q_p}$ = ($p^\top$ - $\tau$ + $q_p$ - 1). Similarly, PrF can be applied when $\tau$ is not greater than ($p^\top/q$) in SAP queries. Otherwise, we cannot get all the matching positional $q_p$-grams from the proposed index because we only store the positional $q_p$-grams with position less than $p^\top$ (recall Section 3.3).

The essence of the node-level and object-level filters is to obtain the candidate nodes and objects that appear at least a certain number of times on the given inverted lists. It is formalized as the *list merging problem* in [18], [10]. The minimum number of appearances corresponds to the min match threshold in the $4^{th}$ column of Table 4.

For the node-level filters, the lists are retrieved from the node-level inverted index using the matching (positional) $q$-grams. For the object-level filters, the lists are retrieved from the object-level inverted index using the matching spatial (positional) $q$-grams. Please refer to the technical report [14] for the analysis of query processing.

TABLE 4: Filters used in a particular query

| Query | Condition | Filter | min match |
|---|---|---|---|
| SP & SPR | $\|s\| \leq UB_{p^\top,0,q_p}$ | PoF | $LB_{s,0,q_p}$ |
| | $\|s\| > UB_{p^\top,0,q_p}$ | CF | $LB_{s,0,q_c}$ |
| SS | | CF | $LB_{s,0,q_c}$ |
| SAP | $\tau \leq p^\top/q_p$ | PrF | 1 |
| | $\|s\| \leq UB_{p^\top,\tau,q_p}$ | PoF | $LB_{s,\tau,q_p}$ |
| | $\|s\| > UB_{p^\top,\tau,q_p}$ | CF | $LB_{s,\tau,q_c}$ |
| SAS | | CF | $LB_{s,\tau,q_c}$ |

Here, we present how a query is processed using an example. We show how to answer an SAP query, $Q_{ap}$ = ($r$, "star", 1), where $\tau = 1$ and $Q_{ap}.r$ is shown in Fig. 4.

**Example 9.** *First, we retrieve the intersecting leaves $Intr(Q_{ap})$: {0.2, 1, 2.0.0, 2.0.3.0, 2.0.3.1, 2.0.3.2, 2.0.3.3}.*

*Second, we use the node-level filters to prune the nodes that cannot satisfy the string constraint. Here, we only describe PoF$_n$. PoF$_n$ with minimum match $LB_{s,\tau,q_p} = 1$ is used. The matching positional 2-grams for $Q_{ap}.str$="star" are ("st", 0), ("st", 1), ("ta", 0), ("ta", 1), ("ta", 2), ("ar", 1), ("ar", 2), ("ar", 3). The inverted lists for the matching positional 2-grams are retrieved and the intersection operation is performed between each inverted list and $Intr(Q_{ap})$. The nodes appearing at least $LB_{s,\tau,q_p} = 1$ time in the inverted entries are candidates. By employing the list merging algorithms [10], we get the candidate nodes, CanNode: {2.0.3.0, 2.0.3.2, 2.0.3.3}.*

*Third, we apply the object-level filters to prune objects in each candidate node. The procedure is similar to that of node-level filters. The major difference is that we retrieve objects from the object-level inverted index using matching spatial positional q-grams. Take node 2.0.3.0 as an example. The Hilbert code (2.0.3.0) and the matching positional 2-grams form the matching spatial positional 2-grams (recall Definition 10). By employing the object-level filters in node 2.0.3.0, we obtain the candidate objects: {$O_5$, $O_6$}.*

*Finally, $O_5$ and $O_6$ are checked by the query constraints. They are returned as answers.*

## 5 Intra-Query Optimization

Recall the framework in Section 2.3, since we adopt an incremental relaxation, it will be beneficial if we could skip processing a certain type of relaxation, whose result size does not exceed the result size threshold. It motivates our first optimization: estimating the selectivity of a query before its processing is triggered. In addition, we can use the dependencies between the relaxation at

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



different levels to optimize our query processing through reuse of intermediate results. These two form intra-query optimization. In this section, we present how to exploit such intra-query optimization to accelerate the processing of the incremental relaxation.

### 5.1 Selectivity Estimation

Given a spatial query $Q_\omega$=($rng$,$str$,$\tau$), we compute its selectivity $S(Q_\omega)$ in a bottom-up manner. We first retrieve the set of leaf nodes that intersect with $Q_\omega.rng$, denoted as $Intr(Q_\omega)$. For each node $n \in Intr(Q_\omega)$, we estimate the selectivity for string $Q_\omega.str$ of query type $\omega$ in node $n$, denoted as $S_\omega(n, Q_\omega.str)$. As we calculate $S(Q_\omega)$, we need to multiply $S_\omega(n, Q_\omega.str)$ by the proportion of node $n$ inside $Q_\omega.rng$. Similar to [19], we assume that objects are uniformly distributed inside each node. Thus this proportion is calculated as $|Obj(n, Q_\omega)|/|Obj(n)|$. Finally, we sum up the calculated values to obtain $S(Q_\omega)$:

$$S(Q_\omega) = \sum_{n \in Intr(Q_\omega)} \frac{|Obj(n, Q_\omega)|}{|Obj(n)|} S_\omega(n, Q_\omega.str), \quad (1)$$

where $Intr(Q_\omega)$, $|Obj(n, Q_\omega)|$ and $|Obj(n)|$ can be retrieved from *getIntrNodeSnippet*($Q_\omega$) (line 1 in Algorithm 1). Next we show how to compute $S_\omega(n, Q_\omega.str)$.

#### 5.1.1 Baseline method: Markov Estimator

The Markov Estimator ($ME$) is used to estimate the substring selectivity in relational databases [20], [21]. It is based on the Markov chain assumption, which states that "the probability of observing a $q$-gram in a sequence depends only on the $q$-gram immediately preceding it, and is independent of all other preceding $q$-grams." We apply the Markov Estimator to estimate the *substring selectivity* of a node. For example, the *substring selectivity* of "star" in node $n$ is:

$$\begin{aligned} ME_s(n, \text{“}star\text{”}) &= P(\text{“}star\text{”}|\text{“}sta\text{”}) * P(\text{“}sta\text{”}) * |n| \\ &\approx P(\text{“}tar\text{”}|\text{“}ta\text{”}) * P(\text{“}sta\text{”}) * |n| \\ &= C_n(\text{“}sta\text{”}) * C_n(\text{“}tar\text{”})/C_n(\text{“}ta\text{”}), \end{aligned} \quad (2)$$

where $C_n(tk)$ is the number of objects containing the $q$-gram $tk$ in node $n$, and can be retrieved from the node-level inverted index using $tk$ as the key. The same example can be used to present the *prefix selectivity* estimation of a node.

$$\begin{aligned} ME_p(n, \text{“}star\text{”}) &= P((\text{“}star\text{”}, 0)|\text{“}star\text{”}) * P(\text{“}star\text{”}) * |n| \\ &\approx (C_n(\text{“}st\text{”}, 0)/C_n(\text{“}st\text{”})) * ME_s(n, \text{“}star\text{”}), \end{aligned} \quad (3)$$

where $C_n(tk, p)$ is the number of objects containing the positional $q$-gram $(tk, p)$ in $n$, and can be retrieved from the node-level inverted index using $(tk, p)$ as the key. The term $(C_n(tk, p)\ /C_n(tk))$ is denoted as the prefix ratio $\Re$.

However, Chaudhuri et al. [21] discovered that the Markov Estimator underestimates the true selectivity when a query contains a short identifying substring.

**Definition 14** (Short Identifying Substring [21]). *A substring $t'$ of $t$ is a short identifying substring if*
*(1) the selectivity of $t'$ is close to the selectivity of $t$, and*
*(2) $t'$ is much shorter than $t$.*

#### 5.1.2 Scarce q-gram method

As shown in [21], if string $t$ contains a short identifying string $t'$, the estimated selectivity of $t'$ is much closer to the true selectivity of $t$. It motivates us to define *scarce q-gram* at the node level, which is an analogy of $t'$, and use the selectivity of the *scarce q-gram* to estimate the selectivity of the query text $t$ in the node.

**Definition 15** (Scarce $q$-gram in a Node). *A $q$-gram $tk$ in node $n$ is scarce if the proportion of objects containing $tk$ in node $n$ is less than a threshold $\epsilon$.*

A scarce $q$-gram is actually a short identifying substring, because: (1) the selectivity of the scarce $q$-gram in a node is close to the selectivity of the original query in the same node because its value is small and is an upper bound of the true selectivity, and (2) $q$-grams are short.

For each $q$-gram of string $t$, the number of objects containing the $q$-gram in node $n$ can be retrieved from the node-level inverted index. Among all the retrieved values in node $n$, we use $C_n^\perp(t)$ to denote the minimum value for the $q$-grams of $t$. Similarly, we use $C_n^{\perp'}(t)$ to denote the minimum value for the positional $q$-grams of $t$ in node $n$.

As stated in Equation 1, we use $S_s(n, t)$ to denote the *substring selectivity* of string $t$ in node $n$. If $(C_n^\perp(t)/|Obj(n)|) < \epsilon$, there exist scarce $q$-grams in node $n$ for $t$. Then we set the value of $S_s(n, t)$ to $C_n^\perp(t)$. Otherwise, we set the value of $S_s(n, t)$ to $ME_s(n, t)$. Therefore, we get:

$$S_s(n, t) = \begin{cases} C_n^\perp(t) & \text{if } (C_n^\perp(t)/|Obj(n)|) < \epsilon \\ ME_s(n, t) & \text{otherwise} \end{cases}$$

Similarly, by integrating the scarce $q$-gram with the Markov Estimator, the *prefix selectivity* of a string $t$ in node $n$ is:

$$S_p(n, t) = \begin{cases} \min(C_n^\perp(t), C_n^{\perp'}(t)) & \text{if } ((C_n^{\perp'}(t)/|Obj(n)| < \epsilon) \\ & \&(C_n^\perp(t)/|Obj(n)| < \epsilon)) \\ C_n^{\perp'}(t) & \text{else if } C_n^{\perp'}(t)/|Obj(n)| < \epsilon \\ C_n^\perp(t) * \Re & \text{else if } C_n^\perp(t)/|Obj(n)| < \epsilon \\ ME_p(n, t) & \text{otherwise} \end{cases}$$

The selectivity is estimated for SPR and SS queries only, but not for SAP and SAS queries because additional information is required [22], which will bring more storage cost.

### 5.2 Reuse of Query Results

Besides the selectivity estimation, our incremental processing provides another optimization opportunity: reusing the result of the relaxation in earlier phases to accelerate the processing of a relaxed query.

**Rule I. Reusing final results of the relaxation in earlier phases.** The *inter-relaxation relationships* (Observation 1) indicate the inclusion relationships between relaxation at different levels. Therefore, we can reuse the less relaxed query's results as a part of the more relaxed query.

**Rule II. Reusing common intermediate results.** Recall Section 4, all types of relaxation are processed over our

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



one-size-fits-all index, many intermediate results among queries can be reused as well, including:

*a. Reusing the snippets (Definition 11) of intersecting nodes*. When the spatial region is relaxed, the SPR query is applied. Suppose that the snippet of node $n$ is $NS(n, Q_p)$ = ($n.hc$, $Obj(n)$, $Obj(n, Q_p)$) for a SP query $Q_p$, it becomes $NS(n, Q_p^r)$ = ($n.hc$, $Obj(n)$, $Obj(n, Q_p^r)$) for the relaxed SPR query $Q_p^r$. When processing node $n$ in $Q_p^r$, we only need to verify the objects in ($Obj(n, Q_p^r) \setminus Obj(n, Q_p)$).

When the prefix constraint is relaxed, the SS, SAP and SAS queries are applied. Since they share the same spatial region, the intersecting nodes remain the same. In addition, the snippet of each intersecting node remains unchanged. Thus, this information can be reused.

*b. Reusing filtering results*. As shown in Table 4, we notice that (1) SP and SPR queries share the same PoF and CF filters; (2) SP and SS queries share the CF filter when the query length is greater than $UB_{p^\top,0,q_p}$; (3) SAP and SAS queries share the CF filter when the query length is greater than $UB_{p^\top,0,q_p}$. These filtering results can be reused.

*c. Reusing the result of an SP query in the selectivity estimation*. We can use the number of results of an SP query as the lower bound when estimating the selectivity of the SPR and SS queries (if needed), instead of computing the selectivity from scratch for these types of relaxation.

*d. Reusing SPR's selectivity to estimate SS's selectivity*. Based on Equation 3, $ME_s(n,t)$ can be computed using $ME_s(n,t) = ME_p(n,t)/\Re$. Since $ME_p(n,t)$ is already computed in estimating SPR's selectivity, we can reuse $ME_p(n,t)$ to compute the selectivity of the later SS query.

**Example 10.** *Suppose we have answered an SP query $Q_p$= ($r$,"star") in Example 6, we need to answer its relaxed SPR query $Q_p^r$=($r'$, "star"). By Observation 1, we obtain $Res(Q_p) \subseteq Res(Q_p^r)$ and $Res(Q_p) = \{O_5, O_6\}$. Thus, $Res(Q_p^r)$ must contain $\{O_5, O_6\}$. This shows the application of Rule I.*

*According to the framework shown in Figure 5, before the processing of $Q_p.r$ is triggered, we estimate $S(Q_p^r)$ first. By Rule II.c, we only need to estimate the selectivity for the nodes that are not contained in $Q_p.r$. We are not required to estimate the selectivity for the nodes that are contained in $Q_p.r$, such as node 2.0.3.0, because we have obtained their results.*

*By Rules I and II.a, we do not revisit the objects in $Q_p.r$. We only need to visit object $O_2$, node 0.1 and 2.0.1. By Rule II.b, SP and SPR queries share the same PoF filter. We notice that node 0.2 is removed by $PoF_n$ previously. Consequently, we do not visit node 0.2 when we process $Q_p^r$.*

## 6 Inter-query Optimization

Recall the framework in Section 2.3, we point out the relationship between a query and its appending query, and highlight the relationships between their results in Observation 2. Next, we present how to exploit the inter-query relationship to accelerate the processing of appending queries.

### 6.1 Processing appending queries

Before discussing how to process appending queries, we need to prove Observation 2, because it is a foundation of our methods. For a string query and its appending query, we observe the following properties.

**Observation 3.** *Given strings $s$,$t$,$u$, where $s$ is a prefix of $t$, and an edit distance threshold $\tau$, these properties hold:*
*(1) If $s$ is not a prefix of $u$, $t$ is not a prefix of $u$.*
*(2) If $s$ is not a substring of $u$, $t$ is not a substring of $u$.*
*(3) If $s$ is not an approximate prefix of $u$ under $\tau$, $t$ is not an approximate prefix of $u$ under $\tau$.*
*(4) If $s$ is not an approximate substring of $u$ under $\tau$, $t$ is not an approximate substring of $u$ under $\tau$.*

*Proof:* Properties (1), (2) and (3) can be easily derived from property (4). Here we prove property (4) by contrapositive. We use D(*,*) to denote the edit distance between two strings, which is calculated by:

$$D(a[0,i], b[0,j]) = min \begin{cases} D(a[0,i-1], b[0,j-1]) + d(a_i, b_j) \\ D(a[0,i-1], b[0,j]) + 1 \\ D(a[0,i], b[0,j-1]) + 1 \end{cases},$$

where $d(a_i, b_j)$=0 if $a_i$=$b_j$, and $d(a_i, b_j)$=1 if $a_i$≠$b_j$. When we match $a_i$ to $b_j$, one substitution is required if $a_i$≠$b_j$. The substitution is not needed if we do not match $a_i$ to $b_j$. Therefore, we get D($a$[0,$i$],$b$[0,$j$])≥D($a$[0,$i$-1],$b$[0,$j$-1]).

By the definition of approximate substring, we suppose D($t$,$u$[$i$, $j$])=$\tau$. We get D($t$[0, |$t$|-1], $u$[$i$, $j$]) ≥ D($t$[0, |$t$|-2], $u$[$i$, $j$-1]) ≥ … ≥ D($t$[0, |$s$|-1],$u$[$i$, $j$-(|$t$|-|$s$|)]) = D($s$, $u$[$i$, $j$-(|$t$|-|$s$|)]).

As a result, we get a substring of $u$, whose edit distance to $s$ is within $\tau$. In other words, $s$ is an approximate substring of $u$ under the edit distance threshold $\tau$. □

By Definition 8, the difference between a spatial query $Q_\omega$ and its appending query $Q'_\omega$ is: $Q_\omega.str$ is a prefix of $Q'_\omega.str$. Thus, we can easily derive the *inter-query relationship* (Observation 2) from Observation 3.

Based on the *inter-query relationship*, we process a query in the following way, as shown in Figure 5. Whenever a query $Q'_\omega$ is coming, we first check whether it is an appending query of the previous query $Q_\omega$. If $Q'_\omega$ is an appending query of $Q_\omega$, we process $Q'_\omega$ from where $Q_\omega$ stops. In addition, we use the results of $Q_\omega$ to form the candidate results of $Q'_\omega$. Otherwise, $Q'_p$ is a non-appending query. We process it as a fresh SP query.

Such *inter-query relationship* plus our incremental paradigm bring two immediate benefits: (1) When processing $Q'_\omega$, instead of starting from scratch, we could start from the type of relaxation where $Q_\omega$ stops. Therefore, processing relaxation at previous levels can be skipped. (2) The result of $Q_\omega$, i.e. $Res(Q_\omega)$, can be output to $Q'_\omega$ as candidates in the corresponding relaxation, and we only need to check whether each object in $Res(Q_\omega)$ satisfies the string condition for $Q'_\omega$, as $Q_\omega.rng$=$Q'_\omega.rng$.

### 6.2 Merging inverted lists for appending queries

When the above reuse does not bring enough number of results for $Q'_\omega$, we need to further relax $Q'_\omega$ and process

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



it by adopting Algorithm 1. The most time consuming part is to get candidates by applying the filters shown in Section 4.1, i.e. to get the objects that appear at least $m$ times in the given inverted lists. It is formalized as the *list merging problem* in [18], [10]. Note that the value of $m$ corresponds to the minimum match threshold, which is shown in Table 4.

Now, our problem becomes: given a set of lists $L_{old}$, we have found the candidate set $C_{old}$, of which an id appears at least $m$ times in $L_{old}$. When a new set of inverted lists $L_{new}$ comes for the appending query (brought by the new $q$-grams), we need to find a set of candidates $C_{new}$, of which an id appears at least $m'$ times in the set $(L_{old} \bigcup L_{new})$.

We propose a novel algorithm to address this problem, where our idea is to *reuse* the results computed for $Q_w$ to efficiently merge the lists when processing its appending query $Q'_w$. When processing $Q_w$, we find these properties:

**Property 1.** *For each id in $C_{old}$, we have its number of occurrences in $L_{old}$, which is at least $m$.*

**Property 2.** *For all the ids not in $C_{old}$, they appear at most ($m$-1) times in $L_{old}$.*

Based on the above two properties, we process the ids in and outside $C_{old}$ separately. There are three cases.

**Case (1)**: $m' \geq (m+|L_{new}|)$. For each id in $C_{old}$, by Property 1, we do not need to visit $L_{old}$ again and can directly check whether it has enough number of appearances in $L_{new}$. For all the ids not in $C_{old}$, by Property 2, they appear at most ($m$-1+$|L_{new}|$) in $(L_{old} \bigcup L_{new})$, which is less than $m'$. Thus, they cannot be in $C_{new}$.

**Case (2)**: $m \leq m' < (m+|L_{new}|)$. For each id in $C_{old}$, the same method is applied as Case (1). For the ids not in $C_{old}$, we present how to process them in Section 6.2.1.

**Case (3)**: $m' < m$. We need to restart merging the inverted lists in $(L_{old} \bigcup L_{new})$ to get $C_{new}$.

### 6.2.1 Processing the ids not in $C_{old}$ of Case (2)

We continue to show how to process the ids not in $C_{old}$ of Case (2). Based on Property 2, we find:

**Property 3.** *For the ids not in $C_{old}$, if a minimum of $m'$ occurrences is required in $(L_{old} \bigcup L_{new})$, they must appear at least ($m'$-$m$+1) times in $L_{new}$.*

The MergeSkip algorithm [10] is proposed to efficiently merge lists using a heap structure. The heap is used to record the frontiers of the inverted lists. If a minimum of $m$ occurrences is required, the heap returns the $(m)^{th}$ smallest record in each round, whose value is $t$. The records smaller than $t$ are skipped in the inverted index because they cannot appear more than $m$ times.

**Example 11.** *Figure 6(a) shows an example of the MergeSkip algorithm. Records which appear at least $m$ = 3 in the inverted lists $L_{old}$ are returned. The MergeSkip algorithm uses a heap to record the frontier of each inverted list. In round one, the heap returns the $3^{th}$ smallest record, 23. The records smaller than 23 are skipped, as the red doted arrows on the left show.*

Similar to the MergeSkip algorithm, we maintain a heap $H_{all}$ for frontiers of the lists in $(L_{old} \bigcup L_{new})$. In addition, we maintain another heap $H_{new}$ for frontiers of the lists in $L_{new}$. In each round, $H_{all}$ returns the $(m')^{th}$ smallest record, whose value is $t_{all}$, while $H_{new}$ returns the $(m'$-$m$+1$)^{th}$ smallest record whose value is $t_{new}$. By Property 3, the records smaller than max($t_{all}$,$t_{new}$) are skipped because they cannot appear more than $m'$ times.

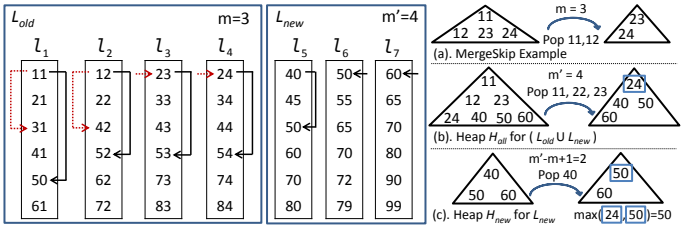



Fig. 6: Merging Lists of Case (2)

**Example 12.** *Figure 6 shows an example of merging lists for the ids not in $C_{old}$. Given $m$ = 3, we need to get the candidates with minimum $m'$ = 4 occurrences in $(L_{old} \bigcup L_{new})$. We use $H_{all}$ to store the frontiers of all the lists in Figure 6(b) and $H_{new}$ to store the frontiers of the lists in $L_{new}$ in Figure 6(c). In round one, $H_{all}$ outputs the $4^{th}$ smallest number, which is 24. $H_{new}$ outputs the $2^{nd}$ smallest number, which is 50. Since* max*(50,24) = 50, the records smaller than 50 are skipped, as the black arrows on the right show. In contrast, only the records smaller than 24 are skipped in MergeSkip algorithm.*

When the new inverted lists are added for the appending query, the MergeSkip algorithm merges the lists from scratch, while our algorithm optimizes this process by reusing the intermediate results computed from the previously-formed query. Therefore, we do not need to merge the lists from scratch for Cases (1) and (2).

## 7 EXPERIMENT

In this section, we aim to study four issues: (1) how the intra-query optimization boosts the performance of INSPIRE for *non-appending* queries, i.e. query reuse and selectivity estimation; (2) how the inter-query optimization accelerates the processing of the *appending queries* in INSPIRE; (3) compare INSPIRE against existing works for each specified type of relaxation to show that there is a need for a single, unifying framework to support spatial prefix query and its different types of relaxation; (4) compare INSPIRE against existing map services to show the effectiveness of our approach in the context of local search. An online demo [23] can be found here[1].

### 7.1 Experiment Setting

**Data sets:** We report results over three real POI data sets in the United States: *OSM*, *GNIS* and *SGP*. *OSM* is downloaded from the OpenStreetMap project[2] which contains

1. http://dbgpucluster-2.d1.comp.nus.edu.sg:8080/MESA/
2. http://www.openstreetmap.org/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



around 1.6 million POIs. *GNIS* is extracted from the Geographic Names Information System[3] and contains the geographic name usage in the U.S. Government, including about two million records. *SGP* is obtained from the SimpleGeo's Places[4] and has thirteen million records. The statistics are shown in Table 5. We observed that the results have similar patterns in our experiment on these data sets. Due to space limitations, we mainly plot the results on the SGP data set. Please refer to our technical report [14] for the complete experiment results.

TABLE 5: Statistics of the used data sets

| Property | *OSM* | *GNIS* | *SGP* |
|---|---|---|---|
| Number of records | 1,616,295 | 2,078,921 | 12,918,933 |
| Size (MB) | 103.7 | 136.7 | 867.2 |
| Max text length (characters) | 173 | 104 | 200 |
| Average text length (characters) | 18 | 19 | 19 |

**Query set:** To ensure that queries have nonempty results, we generate queries in the following way: we randomly select 1000 objects with the length of the textual content larger than 5. Unless otherwise specified, we use the first keyword as the query text to make it compatible with some existing works that can only process queries at the word level. The result size threshold $\theta$ is set to 10. The edit distance threshold $\tau$ is set to 20% of the query length for SAP and SAS queries.

We normalize the latitude and longitude into [0, 1]. Thus, the query range is denoted by the percentage of coverage in the geographic space. Six spatial ranges with the selected object as the center are generated: $0.005^2$, $0.01^2$, $0.02^2$, $0.04^2$, $0.06^2$ and $0.08^2$. The first (last) range is at a town (state/province) level. By default, we use the $0.01 \times 0.01$ range at a city level and expand the query region by doubling the area in the SPR query.

**Comparisons:** `TAS` [1] and `FEH` [2] are proposed to provide type-ahead search in spatial databases. Besides, `LBAK` [4] and `MHR` [3] are proposed to allow approximate keyword matching in spatial databases. We compare with `TAS` for SP queries, `FEH` for SP and SS queries, `LBAK` and `MHR` for SAP and SAS queries. In particular, `TAS` and `LBAK` are memory resident while `FEH` and `MHR` are disk resident. The parameters of the baseline algorithms are set to the default values in the original papers. Our methods are implemented in JAVA. All experiments are run on a Quad-Core AMD Opteron Processor 8356 @ 2.29 GHz with 128 GB main memory. Queries are run under a heap with a maximum size of 4 GB.

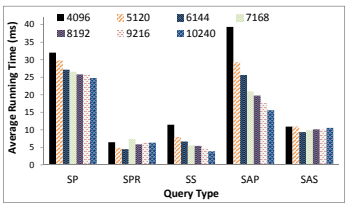



Fig. 7: Impact of the max capacity on *SGP*

**Parameter Choice:** According to [24], the performance of a Quadtree depends on the tiling level, which in turn depends on the max capacity of a Quadtree node. So we study the performance of each type of relaxation w.r.t. different choices of the max capacity. We test values from 512 to 10240 for all the three data sets. The average running time for *SGP* is shown in Figure 7. The time for the values from 512 to 3072 is omitted as it is much longer than that of the rest.

We take all types of queries into consideration when choosing an appropriate max capacity. For each type of query, the max capacity with the shortest running time is assigned a score of 1 while the scores for the rest are computed as the proportion of the running time to the shortest one. The score of the overall performance for each max capacity is the summation of its scores for all the five types of queries. We choose the parameter with the lowest score, which indicates the best overall performance. As a result, we set the max capacity to 3072, 3072 and 10240 for *OSM*, *GNIS* and *SGP* respectively.

The length of $q_c$-grams and positional $q_c$-grams is set to 3 and 2 respectively, which is the same as the setting in [25]. The reserved position $p^\top$ is set to 9. By Definitions 12 and 13, up to four edit operations can be supported by PrF. If there is an error in every five characters, PrF can be applied when the query length is less than 20.

TABLE 6: Index Construction Time and Index Size

| Index | *OSM* | | *GNIS* | | *SGP* | |
|---|---|---|---|---|---|---|
| | Time(mins) | Size(MB) | Time | Size | Time | Size |
| `INSPIRE` | 2.7 | 346 | 3.6 | 448 | 21.5 | 3857 |
| `TAS` [1] | 2.0 | 150 | 3.4 | 194 | 20.1 | 1445 |
| `FEH` [2] | 5.1 | 306 | 7.3 | 405 | 33.6 | 2704 |
| `MHR` [3] | 4.2 | 663 | 7.4 | 857 | 37.0 | 5332 |
| `LBAK` [4] | 1.0 | 253 | 1.4 | 291 | 9.5 | 2541 |

**Index Construction:** We build our index based on the above parameters. Table 6 shows the construction time and the size of our proposed index for the chosen parameters and the compared indexes. Overall, `LBAK` builds the index in the shortest time, while `TAS` has the smallest index size. Our index is around the medium in the term of construction time and the index size.

### 7.2 Comparison between Variants of `INSPIRE`

In Sections 5 and 6, the intra-query optimization and inter-query optimization are proposed to accelerate the processing of non-appending and appending queries respectively. So we would like to study the effectiveness of these optimizations.

#### 7.2.1 Intra-query Optimization

We first study the impact of each of the two intra-query optimization techniques to accelerate the processing of the *non-appending queries*: result reuse and selectivity estimation (in Section 5). Note that, for each technique under test, the other one is turned on by default.

**Result Reuse:** We compare the running time of the relaxed queries with reuse to that without reuse. As shown in Figure 8, we find that the performance of each relaxed type boosts up by four times for *SGP* on average. The SP query takes more time for the reuse version as

3. http://geonames.usgs.gov/domestic/download_data.htm
4. http://s3.amazonaws.com/simplegeo-public/places_dump_20110628.zip

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



common intermediate results and processing are saved. Compared to the extra cost of the SP query, it is worth applying this optimization as the benefit that it brings is significant for the relaxed queries. For *OSM* and *GNIS*, the performance boosts up by three times on average.

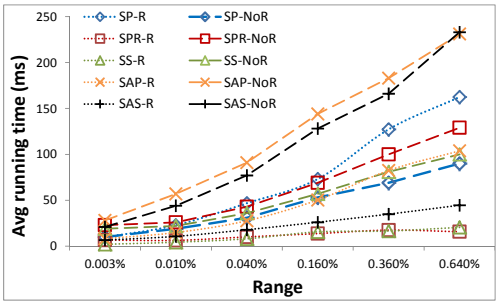
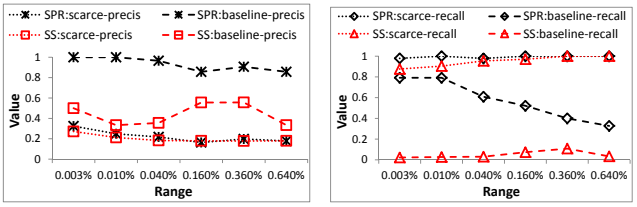



Fig. 8: Relaxation of the SP query for *SGP*

**Selectivity Estimation:** We compare our method using the scarce $q$-grams to the baseline method, which uses the Markov Estimator as described in Section 5.1.1.

Note that a query is triggered if its estimated selectivity is greater than the result size threshold $\theta$. Therefore, the terms positive and negative are defined as whether the estimated selectivity is greater than or less than $\theta$ respectively. Accordingly, when both the true selectivity and the estimated selectivity are greater than $\theta$, we refer to it as *true positive*. The scarce threshold $\epsilon$ is set to 0.01.




(a) Precision (b) Recall

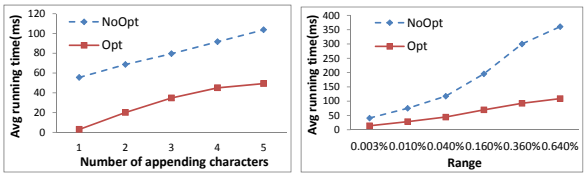
Fig. 9: Precision and Recall

Figure 9 shows the precision and recall for the SPR and SS queries in different query ranges for *SGP*. Overall, our approach has a much higher recall than the baseline method, and it is at least 0.8 across all query ranges for *SGP*. In contrast, the recall of the baseline is low for SS queries so that many queries that should be processed are skipped. The baseline approach has higher precision than ours, because it underestimates the selectivity when a query contains scarce $q$-grams. However, a potential problem of such high precision is to bring more time complexity in processing the relaxed query at the next level as they may lose the possibility to reuse the results got in earlier phases.

### 7.2.2 Inter-query Optimization

In this part, we would like to study the impact of the inter-query optimization (proposed in Section 6) to accelerate the processing of the *appending queries*. The non-optimized approach is the one treating an appending query as a fresh query.

We first investigate the impact of the number of appending characters, ranging from 1 to 5. From Figure 10a, We find: (1) The processing time increases as the number of appending characters increases, because more inverted lists need to be merged to get candidates. (2) On average our inter-query optimization brings around three times acceleration on the three data sets.




(a) Appending Characters (b) Spatial Range

Fig. 10: Impact of inter-query optimization

Next, we study how the performance of the appending query varies w.r.t. different query ranges, while the number of appending characters is randomly selected from one to five for each query to simulate real cases. From Figure 10b, we find that the method with optimization scales better than the one without optimization on *SGP*. This same finding exists on *OSM* and *GNIS* as well.

## 7.3 Comparison to Other Approaches

In this subsection, we compare `INSPIRE` (which is equipped with all the aforementioned optimizations) with existing works. However, all state-of-the-art works only study a certain type of relaxation for the spatial prefix query (see Section 8 for more details). As such, we have to compare `INSPIRE` with each individual type of relaxation even though it is not a fair comparison for a general framework. For each type of relaxation, we divide the query set into two parts: appending queries and non-appending queries.

### 7.3.1 SP & SS Queries

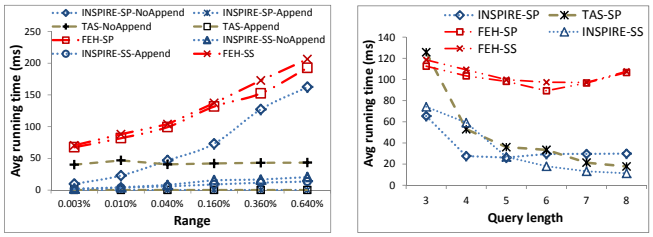
We compare with `TAS` [1] for SP queries, and `FEH` [2] for SP and SS queries. Figure 11a plots the average running time in different query ranges. We find: (1) `INSPIRE` outperforms `TAS` when the query range is small, while `TAS` has better performance when the query range increases. This is because `TAS` retrieves candidates on the textual dimension first and then verifies candidates on the spatial dimension. However, `INSPIRE` performs in the opposite way. (2) `INSPIRE` outperforms `FEH` on both SP and SS queries. (3) Moreover, the running time of appending queries is shorter than that of non-appending queries. In particular, for SP queries, the running time of appending queries is almost 0 because the results of previously-formed queries can be used directly.

Furthermore, we investigate the impact of query lengths. Figure 11b demonstrates the query performance for lengths ranging from 3 to 8. `INSPIRE` outperforms `TAS` when the query text is short. This is because `TAS` uses a trie-like index to retrieve candidate objects that satisfy the prefix condition. Then all the candidate objects are verified against the spatial condition one by one to get the final results. When the query text is short, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



candidate size could be huge. Thus the verification time is long. We also find that INSPIRE outperforms FEH, mainly because FEH needs to retrieve inverted entries at each level of its spatial index.



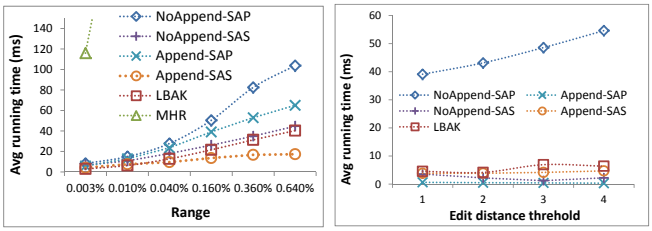

(a) Spatial Regions (b) Query Length

Fig. 11: SP & SS Queries

### 7.3.2 SAP & SAS Queries

For SAP and SAS queries, no existing work has addressed the search-as-you-type problem at the $q$-gram level, so no direct comparison can be made. Instead, we compare with two works (LBAK [4] and MHR [3]) at the word level. They require users to type at least one full word for a query. Accordingly, we form queries by extracting the first word from each query of our query set. Again, we note that it is not a fair comparison because INSPIRE works at the $q$-gram level.

Figure 12a shows the average running time of different query ranges. LBAK achieves the best performance while INSPIRE gives intermediate performance. One reason is that INSPIRE is proposed in a more general setting, which handles approximate queries in the search-as-you-type paradigm[5]. Another reason is that LBAK is memory-based, thus disk access is not required. Although it is not a fair comparison, the performance of INSPIRE is at the same level as that of the state of the art.




(a) Spatial Range (b) Edit Distance

Fig. 12: SAP & SAS Queries

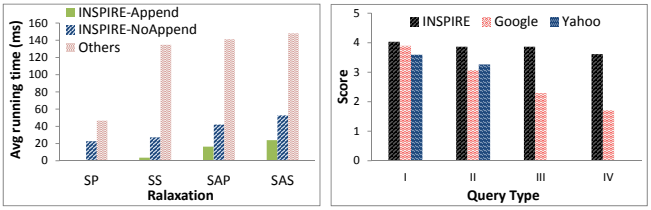
**Impact of edit distance threshold:** Next, we test the performance for different values of the edit distance threshold $\tau$. We randomly select objects with the length of the first keyword greater than 17 from both data sets. Then we add one, two, three, and four errors respectively to the query to evaluate the performance. The errors are randomly generated by skipping letter, doubling letters, reversing letters, replacing letter, or inserting letter.

Figure 12b shows the average running time for different $\tau$ values. For the non-appending queries, LBAK achieves the best result. For the appending queries, INSPIRE is the best because the results of the previously-formed queries can be reused. We do not plot the results of MHR because the running time for MHR is at least 280 milliseconds.

5. LBAK, for example, cannot handle both exact and approximate substring queries.

### 7.3.3 Overall Performance

At the framework level, we would like to see INSPIRE 's overall performance in a real usage scenario, where we do not know which type of relaxation is sufficient to obtain enough number of results. Since all of the existing works study a single type of relaxation, we construct a baseline framework as an integration of the state of the art for each type of relaxation, and compare with INSPIRE equipped with the proposed inter-query and intra-query optimizations. In particular, TAS is tailored for SP queries, while FEH is tailored for SS queries and LBAK is tailored for SAP and SAS queries.




(a) Efficiency (b) Effectiveness

Fig. 13: Framework Comparison

Figure 13a shows the accumulated time when different types of relaxation stop. INSPIRE outperforms such an integrated framework. Especially for appending queries, INSPIRE handles them almost for free due to the various result reuse algorithms that we have proposed. In contrast, a simple integration of existing methods does not have such benefit, because the existing methods answer one type of particular query using an optimized index structure. Although our method loses for some particular types of queries, our INSPIRE framework outperforms the existing methods as a system to provide relaxation for spatial prefix query. This finding confirms the need for a single, unifying framework to support the spatial prefix query and its different types of relaxation.

## 7.4 Effectiveness Study

We conduct an effectiveness study on local search to simulate the real-life search scenario. Queries are issued in a small spatial region such as a city. We test five online map services, including Google Maps, Bing, Yahoo!, Here and OpenStreetMap. Only Google and Yahoo! provide the instant search feature. We use the *SGP* data set in INSPIRE for this effectiveness study.

For the query set, we first find tourist attractions from the five cities with the largest population in the USA: New York (NY), Los Angeles (LA), Chicago (CH), Houston (HO) and Philadelphia (PH). Then we extract the texts from these tourist attractions and generate four types of queries:

Type I. the query is a prefix of the text;

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



Type II. the query is a substring of the text;
Type III. the query is a prefix of the text, with typos;
Type IV. the query is a substring of the text, with typos.
For each city, four types of queries are formed on the attractions in the city, and each type contains ten queries.

**Evaluation Method**: We invite twenty participants to take part in the task of scoring the top-5 results from the three systems. To conduct a fair evaluation, the source of each result is kept anonymous. The participants are asked to score the quality of each result by using the Cumulated Gain-based evaluation [26] metric (from 0 to 5 points, 5 means the best while 0 means the worst). Each participant grades the results of three cities. Therefore, for each result, there are twelve participants grading it.

**Results**: We first report the summarized result, which is shown in Figure 13b. For query Types I and II, the three systems perform equally well. For Types III and IV, INSPIRE outperforms Google Maps. The results of Yahoo! Maps are not shown because it does not support error-tolerant search. The reason why INSPIRE performs better is that INSPIRE in essence is able to answer error-tolerant instant search. Google Maps, though, probably adopts a global search strategy to serve the users from all over the world.

TABLE 7: Query Examples

<table>
<tr><td>Type I</td><td>$Q_1$:(CH, “navy pier”)</td><td>$Q_2$:(NY, “times square”)</td></tr>
<tr><td>Inspire</td><td rowspan="3">navy pier in CH</td><td rowspan="3">times square in NY</td></tr>
<tr><td>Google</td></tr>
<tr><td>Yahoo!</td></tr>
<tr><td>Type II</td><td colspan="2">$Q_3$:(HO, “race park”)</td></tr>
<tr><td>Inspire</td><td rowspan="3">sam houston race park in HO</td><td>ace park & ride in HO</td></tr>
<tr><td>Google</td><td>delaware park in Wilmington</td></tr>
<tr><td>Yahoo!</td><td>seven flags race park in Dickson</td></tr>
<tr><td>Type III</td><td>$Q_4$:(LA, “doger stadium”)</td><td>$Q_5$:(NY, “yangkee”)</td></tr>
<tr><td>Inspire</td><td rowspan="2">dodger stadium in LA</td><td>yankee stadium in NY</td></tr>
<tr><td>Google</td><td>yangkee logistics in Singapore</td></tr>
<tr><td>Type IV</td><td>$Q_6$:(HO, “otdoor theatre”)</td><td>$Q_7$:(LA, “librart”)</td></tr>
<tr><td>Inspire</td><td>miller outdoor theatre in HO</td><td>library tower in LA</td></tr>
<tr><td>Google</td><td></td><td>librarti sa in Switzerland</td></tr>
</table>

We then explain the results using some selected examples, which are listed in Table 7. For Types I and II, the three maps return correct results in most cases. Still, some results are not related to the issued query. We show two sample results for $Q_3$. From the column on the right side, INSPIRE returns a POI in Houston. The query text is an approximate prefix of the text of the POI. However, the POIs returned from the other two systems are spatially far away from the query region. For Types III and IV with typos, INSPIRE can correct the typos in the queries and return reasonable answers. In contrast, Google Maps does not always perform well. For $Q_5$ and $Q_7$, Google Maps again returns distant POIs from the query region, probably because their names match the queried text. This approach is a double-edged sword. On one hand, it suggests distant objects that possibly match the query terms, which is pretty good if the user is searching on a global scale. On the other hand, it dwarfs the approximate answers which are local. Depending on users' intention, this may or may not be helpful. However, in our context of local search, the local answers surely play a more important role.

From the above effectiveness study, we find: (1) the search strategy proposed in INSPIRE fulfills the need for context-aware local search; (2) INSPIRE is a good complement to online web services on context-aware local search. When a user conducts a local search, the user would continuously zoom in a particular spatial region in the viewport. Therefore, the local search pattern can be easily detected and the local search can be optimized using INSPIRE to improve the user experience.

## 8 RELATED WORK

Spatial keyword search has been receiving significant attention recently. Chen et al. [27] summarized three types of spatial keyword queries. They surveyed twelve state-of-the-art geo-textual indexes and compared them.

In order to ensure effective spatial keyword search, various extensions were done, including autocompletion [1], [2] and the allowance of errors in keyword matching [3], [4]. Autocompletion is widely used to predict what words users want to type without typing them completely [28]. In particular, prefix search is a method to achieve the autocompletion and is integrated into spatial databases [1], [2]. In addition, approximate string search is an error-tolerant technique to find strings that match a query approximately and is combined with spatial keyword search [3], [4].

However, the approximate search proposed in [3], [4] cannot support autocompletion. As stated in [11], real-time search engines should be error-tolerant for autocompletion. These approaches model the approximate search at the word level and require users to type at least one full word for a query. Therefore, they violate the concept of autocompletion. To the best of our knowledge, supporting the error-tolerant autocompletion has not yet been studied in the context of spatial databases. In contrast, INSPIRE supports error-tolerant autocompletion functions such as the approximate prefix query and the approximate substring query.

In addition to being used to query POIs within a spatial region, spatial keyword query is used to search travel routes or trajectories [29], [30], [31], [32], [33]. Generally, it returns optimal routes that cover or approximately cover a set of user-specified keywords as results.

## 9 CONCLUSION

In this paper, we proposed INSPIRE, a general framework, which adopts a unifying strategy for processing different variants of spatial keyword queries. We observed that relaxation can be done by expanding the spatial region or loosening the prefix matching constraint. To maximize query reusability, we adopted an incremental way to perform relaxation. Then we built a one-size-fits-all index to support all types of relaxation. To accelerate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2015.2391107, IEEE Transactions on Knowledge and Data Engineering



query processing, we proposed our intra-query optimization covering common result reuse and selectivity estimation. Moreover, we observed that the search-as-you-type paradigm allows appending characters after the initial query, so we proposed the inter-query optimization to reuse the results of the initial query in processing its appending query, instead of processing it from scratch. As a result, INSPIRE is able to decide the most appropriate relaxation for a spatial prefix query, and outperforms most existing works.

In the future work, we would like to investigate other relaxation strategies such as relaxing the edit distance threshold progressively so that similar results are returned first. Another possible improvement on spatial prefix search is taking semantics into account and organizing results in appropriate orders by ranking and diversifying results. In addition, we can extend the query relaxation to other applications such as searching trajectories or graph data.

## REFERENCES


[1] S. Basu Roy and K. Chakrabarti, "Location-aware type ahead search on spatial databases: semantics and efficiency," in *SIGMOD*, 2011.
[2] S. Ji and C. Li, "Location-based instant search," in *SSDBM*, 2011.
[3] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou, "Approximate string search in spatial databases," in *ICDE*, 2010.
[4] S. Alsubaiee, A. Behm, and C. Li, "Supporting location-based approximate-keyword queries," in *SIGSPATIAL GIS*, 2010.
[5] R. Finkel and J. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, no. 1, 1974.
[6] H. Sagan, *Space-filling curves*. Springer-Verlag, 1994, vol. 2.
[7] M. Bader, *Space-Filling Curves: An Introduction With Applications in Scientific Computing*. Springer, 2012, vol. 9.
[8] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate string joins in a database (almost) for free," in *VLDB*, 2001.
[9] C. Xiao, W. Wang, and X. Lin, "Ed-join: an efficient algorithm for similarity joins with edit distance constraints," *VLDB*, 2008.
[10] C. Li, J. Lu, and Y. Lu, "Efficient merging and filtering algorithms for approximate string searches," in *ICDE*, 2008.
[11] S. Chaudhuri and R. Kaushik, "Extending autocompletion to tolerate errors," in *SIGMOD*, 2009.
[12] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, L. Pietarinen, and D. Srivastava, "Using q-grams in a dbms for approximate string processing," *IEEE Data Eng. Bull.*, vol. 24, no. 4, 2001.
[13] G. Li, S. Ji, C. Li, and J. Feng, "Efficient fuzzy full-text type-ahead search," *The VLDB Journal*, vol. 20, no. 4, 2011.
[14] Y. Zheng, Z. Bao, L. Shou, and A. K. H. Tung, "Inspire: A framework for incremental spatial prefix query relaxation," *Technical Report*, 2014. [Online]. Available: https://www.dropbox.com/s/tjpzs2t08zxxfmj/inspire.pdf?dl=0
[15] C. Xiao, W. Wang, and X. Lin, "Ed-join: An efficient algorithm for similarity joins with edit distance constraints," *Proc. VLDB Endow.*, vol. 1, no. 1, 2008.
[16] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava, "Bed-tree: An all-purpose index structure for string similarity search based on edit distance," in *SIGMOD*, 2010.
[17] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz, "Analysis of a very large web search engine query log," *SIGIR Forum*, vol. 33, no. 1, 1999.
[18] S. Sarawagi and A. Kirpal, "Efficient set joins on similarity predicates," in *SIGMOD*, 2004.
[19] S. Acharya, V. Poosala, and S. Ramaswamy, "Selectivity estimation in spatial databases," in *SIGMOD*, 1999.
[20] H. V. Jagadish, O. Kapitskaia, R. T. Ng, and D. Srivastava, "One-dimensional and multi-dimensional substring selectivity estimation," *The VLDB Journal*, vol. 9, no. 3, 2000.
[21] S. Chaudhuri, V. Ganti, and L. Gravano, "Selectivity estimation for string predicates: overcoming the underestimation problem," in *ICDE*, 2004.
[22] H. Lee, R. T. Ng, and K. Shim, "Approximate substring selectivity estimation," in *EDBT*, 2009.
[23] Y. Zheng, Z. Bao, L. Shou, and A. K. H. Tung, "Mesa: A map service to support fuzzy type-ahead search over geo-textual data," *PVLDB*, vol. 7, no. 13, 2014.
[24] R. K. V. Kothuri, S. Ravada, and D. Abugov, "Quadtree and r-tree indexes in oracle spatial: a comparison using gis data," in *SIGMOD*, 2002.
[25] Y. Kim and K. Shim, "Efficient top-k algorithms for approximate substring matching," in *SIGMOD*, 2013.
[26] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, 2002.
[27] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *VLDB*, 2013.
[28] H. Bast and I. Weber, "Type less, find more: fast autocompletion search with a succinct index," in *SIGIR*, 2006.
[29] B. Yao, M. Tang, and F. Li, "Multi-approximate-keyword routing in gis data," ser. SIGSPATIAL GIS, 2011.
[30] X. Cao, L. Chen, G. Cong, and X. Xiao, "Keyword-aware optimal route search," *Proc. VLDB Endow.*, vol. 5, 2012.
[31] G. Cong, H. Lu, B. C. Ooi, D. Zhang, and M. Zhang, "Efficient spatial keyword search in trajectory databases," *CoRR*, 2012.
[32] K. Chen, W. Sun, C. Tu, C. Chen, and Y. Huang, "Aggregate keyword routing in spatial database," ser. SIGSPATIAL, 2012.
[33] X. Cao, L. Chen, G. Cong, J. Guan, N.-T. Phan, and X. Xiao, "Kors: Keyword-aware optimal route search system," in *ICDE*, 2013.


PLACE PHOTO HERE

**Yuxin Zheng** is currently working toward the PhD degree in the Department of Computer Science, School of Computing, National University of Singapore. His research interests include spatial database, high-dimensional data management and computer vision.

PLACE PHOTO HERE

**Zhifeng Bao** is an Assistant Professor in University of Tasmania, Australia. He received his PhD in computer science from the National University of Singapore in 2011. He was the winner of the Singapore IDA gold medal. His works got the best paper nomination in ICDE 2009, DASFAA 2012 and ASONAM 2013. His research focuses on how to make heterogeneous data usable via visualized, interactive and efficient keyword search, and enhance data & knowledge sharing over the social network.

PLACE PHOTO HERE

**Lidan Shou** received the PhD degree in computer science from the National University of Singapore. He is a professor with the College of Computer Science, Zhejiang University, China. Prior to joining the faculty, he had worked in the software industry for more than two years. His research interests include spatial database, data access methods, visual and multimedia databases, and web data mining. He is a member of the ACM.

PLACE PHOTO HERE

**Anthony K.H. Tung** received the BSc (second class honor) and MSc degrees in computer science from the National University of Singapore (NUS), in 1997 and 1998, respectively, and the PhD degree in computer sciences from Simon Fraser University in 2001. He is currently an associate professor in the Department of Computer Science (NUS). His research interests include various aspects of databases and data mining (KDD) including buffer management, frequent pattern discovery, spatial clustering, outlier detection, and classification analysis.