Spatio-Temporal Keyword Queries for Moving Objects

Paras Mehta
Databases and Information Systems Group
Freie Universität Berlin
paras.mehta@fu-berlin.de

Dimitrios Skoutas
Institute for the Management of Information Systems
R.C. ATHENA
dskoutas@imis.athena-innovation.gr

Agnès Voisard
Databases and Information Systems Group
Freie Universität Berlin and Fraunhofer FOKUS
agnes.voisard@fu-berlin.de

ABSTRACT

Many applications involve queries that combine spatial, temporal and textual filters. In this paper, we address the problem of efficient evaluation of queries that perform spatial, temporal and keyword-based filtering on historical movement data of objects which are additionally associated with textual information in the form of keywords. Our work combines and builds upon concepts and techniques for spatio-temporal and spatio-textual queries, proposing two hybrid indexes for this purpose. An experimental evaluation of the proposed approaches is presented, using real-world datasets from two different types of sources.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Experimentation

Keywords

moving objects, spatio-textual search, spatio-temporal search

1. INTRODUCTION

With the increasingly widespread use of GPS and mobile devices it has become possible to track the movement of various types of objects, ranging from ships, airplanes and vehicles to animals and people. Storing, querying and analyzing such movement data is becoming increasingly interesting and important for many applications, including location-based services, fleet management, emergency response and others. Moreover, users in social networks generate large amounts of content that encompasses spatial, temporal and textual information. A typical example is a traveler who uploads geotagged photos on Flickr or posts geotagged tweets while moving around in a city. This results in a "trail" of photos or tweets, each one being characterized by a location, a timestamp and a set of tags or keywords.

Although there exists an extensive amount of work both on spatio-temporal queries [7, 9] and spatio-textual queries [2], the problem of efficiently evaluating queries that combine all the three dimensions, *spatial*, *temporal* and *textual*, remains largely unexplored [5, 8, 4]. In this paper, we address this problem, focusing on historical movement data of objects which are additionally associated with textual information in the form of keywords, potentially changing at each timestamp and location.

In particular, we propose two hybrid indexes that combine and build upon concepts and techniques for spatio-temporal and spatio-textual search. The first one, denoted as `GKR` (Grid and KR*-tree), is based on a spatio-temporal index (`SETI` [1]) used for indexing trajectories of moving objects, and enhances it to incorporate the keyword information associated with the trajectory segments. The second, denoted as `IFST` (Inverted File with Spatio-Temporal order), is based on a spatio-textual index (`SFC-QUAD` [3]), enhancing it to incorporate the temporal dimension.

These indexes allow to (a) extend spatio-temporal queries to moving objects that are associated with textual information, and (b) extend spatio-textual queries to moving instead of static objects. To compare the performance of the two approaches, we conduct an experimental evaluation using two different real-world datasets, including yacht movement tracking data and geotagged images from Flickr.

2. PRELIMINARIES

Spatio-temporal indexes. A comprehensive survey of spatio-temporal access methods can be found in [9]. Existing indexes are categorized according to whether they index past, current or future positions of moving objects (or combining all three). In our work, we focus on the first category, namely, indexing the past positions of moving objects. For instance, SETI [1], which forms the basis of the `GKR` index proposed here, employs a two-level index structure to handle the spatial and the temporal dimensions. The spatial dimension is partitioned into static, non-overlapping partitions. Then, for each partition, a sparse index is built on the temporal dimension. Furthermore, an in-memory structure is used to speed up insertions. Queries are evaluated by first performing spatial filtering and then temporal filtering. Query execution concludes with a refinement step, to filter out candidates, and a duplicate elimination step, to filter out segments that belong to the same trajectory.

SIGSPATIAL'15 November 03-06, 2015, Bellevue, WA, USA

DOI: http://dx.doi.org/10.1145/2820783.2820845.

Spatio-textual indexes. A comprehensive survey and comparison of proposed indexes for spatio-textual queries can be found in [2]. These indexes are usually hybrid structures comprising a spatial indexing part and a text indexing part. The former is typically based on an R-tree, grid or space filling curve, while the latter can be an inverted file or a bitmap. Further, two main query processing strategies can be distinguished, textual-first and spatial-first [3]. For instance, in the former case, the top-level index can be an inverted file with each postings list indexed by an R-tree, while in the latter case, the top-level index can be an R-tree with inverted files attached to each leaf node. SFC-QUAD, on which our second hybrid index is based, employs a space filling curve and a quad-tree structure [3].

3. PROBLEM DEFINITION

Let $\mathcal{O}$ be a set of moving objects associated with a set $\mathcal{T}$ of trajectories. Each trajectory $T \in \mathcal{T}$ belongs to an object $o \in \mathcal{O}$ and comprises a series of line segments, i.e. $T = (o, \langle \ell_1, \ell_2, \ldots, \ell_n \rangle)$. Each line segment is defined by a tuple $\ell = (p_s, p_e, \psi)$, where $p_s = (x_s, y_s, t_s)$ and $p_e = (x_e, y_e, t_e)$ are its start and end points, respectively, specified by a location and a timestamp, and $\psi = \{k_1, k_2, \ldots, k_m\}$ is a set containing zero or more keywords associated with this part of the trajectory. We use the notation $\ell.loc$, $\ell.\tau$ and $\ell.\psi$ to refer, respectively, to the location, the timespan and the set of keywords of the trajectory segment ℓ.

We define the Spatio-Temporal Keyword (`STK`) query as a boolean range query that comprises a spatial, a temporal and a keyword filter, i.e. $Q = (R, T, \Psi)$, where $R = [(x_s, y_s), (x_e, y_e)]$ specifies a spatial range, $T = [t_s, t_e]$ a time interval, and $\Psi = \{k_1, k_2, \ldots, k_n\}$ a set of keywords.

DEFINITION 1 (STK QUERY). *Given a set of objects $\mathcal{O}$ and their trajectories $\mathcal{T}$, the STK query $Q = (R, T, \Psi)$ returns a set of objects $O \subseteq \mathcal{O}$ such that each $o \in O$ contains a set of trajectory segments $T_{o,q} \subseteq T_o$ that satisfy all of the following conditions: (a) $\forall \ell \in T_{o,q} : \ell.loc \cap R \neq \emptyset$, (b) $\forall \ell \in T_{o,q} : \ell.\tau \cap T \neq \emptyset$, and (c) $\bigcup_{\ell \in T_{o,q}} \ell.\psi \supseteq \Psi$.*

4. THE GKR INDEX

`GKR` is a hybrid structure that combines concepts from the `SETI` index [1], for indexing trajectories of moving objects, and the `KR*-tree` [6], for indexing spatio-textual objects.

`GKR` uses a grid to partition the space into a number of equally sized disjoint cells. Each cell indexes the trajectory segments that lie within it. Segments that cross multiple cells are split, so that each new segment is fully contained within a single cell. These segments are then stored in one or more disk pages, such that each disk page only contains segments belonging to the same cell. This part is similar to `SETI`, which also partitions the space into disjoint cells and stores their contents in separate disk pages. However, since `SETI` only deals with spatiotemporal data, each created disk page is only associated with a timespan, which is the union of the timespans of the segments stored in it. Then, the timespans of all pages belonging to the same cell are organized in a one-dimensional R*-tree.

Instead, in our case, each segment contained in a cell is characterized by both its timespan and the list of keywords associated with it. To deal with both dimensions, we organize the corresponding disk pages of a cell using a `KR*-tree`. Now, besides its timespan, each disk page is associated with a set of keywords, which is the union of the sets of keywords associated with its segments. The `KR*-tree` is an augmented R*-tree that additionally associates nodes with keywords. Thus, the disk pages are again organized in an R*-tree according to their timespans, but in addition a structure is maintained associating tree nodes with keywords contained in the corresponding disk pages.

Since the `GKR` index combines and adapts parts from `SETI` and `KR*-tree`, the insert and update procedures also follow steps similar to the corresponding ones for those indexes. Specifically, inserting a new trajectory segment ℓ is performed following the steps described below.

1. Identify the grid cells that ℓ crosses. If there are more than one such cells, split ℓ into multiple segments.
2. Identify the disk page(s) associated with that particular cell. In these pages, the segments are ordered chronologically, according to the timestamp of their endpoint. Thus, traverse the `KR*-tree` associated with the cell to find the page in which the new segment should be inserted.
3. If such a page exists and is not full, insert the new segment. Otherwise, shift the contents of the subsequent pages or create a new page.
4. Update the timespan and the keyword set of the affected page(s), as well as the `KR*-tree`.

We assume that in practice the `GKR` index is constructed in bulk mode, inserting trajectory segments in chronological order. Hence, each new segment is appended at the end of the last disk page of the corresponding cell (or in a new disk page, if the last one is full).

Next, we describe the steps for evaluating an `STK` query $Q = (R, T, \Psi)$ using the `GKR` index.

1. Select all candidate grid cells that overlap with R.
2. For each candidate cell, traverse the corresponding `KR*-tree` to identify those nodes that: (a) have a timespan that overlaps with T, and (b) have a keyword that is contained in Ψ.
3. From the leaf nodes reached, retrieve the set of candidate disk pages. These pages provide a set of candidate trajectory segments that potentially satisfy predicates R and T and contain one or more keywords from Ψ.
4. Apply a first filtering step as a refinement w.r.t. the spatial and temporal dimensions: discard segments that are false positives, i.e. are located outside R or have their timespans outside T. This results in a set of candidate objects, which are the objects to which the remaining segments belong.
5. Apply a second filtering step to discard those objects whose trajectory segments from Step 4 do not fully cover the set of query keywords Ψ.

5. THE IFST INDEX

`IFST` comprises two main structures. The first is a global inverted file, containing, for each keyword, an inverted list with the ids of the trajectory segments that contain it. When a query is evaluated, only segments appearing in the inverted lists associated with keywords contained in the query need to be examined. Since these lists can still be quite long, the key issue for efficiency is to restrict the portions of each list that may contain segments satisfying the spatial and

temporal predicates of the query. This is achieved by assigning ids to segments in a spatio-temporal order. For this purpose, a second, hybrid structure is maintained, which comprises in turn two parts. The first is a quadtree that indexes trajectory segments according to the spatial dimension. This allows for ordering cells, and their corresponding trajectory segments, according to a Z curve, so that segments that are spatially close together will also have similar ids. Furthermore, segments belonging to the same cell are assigned ids chronologically, according to the end timestamp of each segment. Then, for each leaf node of the quadtree, an R*-tree is built to index the timespans of the contained segments. Lastly, the inverted lists are themselves split into blocks (with size that is typically a multiple of 128 bytes) and are compressed using a block compression algorithm before being stored on disk.

The following steps describe how the `IFST` index is constructed.

1. Construct a quadtree to partition the space and assign ids to cells according to the Z-order. Each trajectory segment is assigned to the corresponding cell; if it spans more than one cells, it is split.
2. Assign ids to segments according to the position of their parent cell in the Z-ordered quadtree and their position in the chronological order of segments within the cell.
3. For each cell, construct an R*-tree to index the timespans of the contained segments.
4. Construct an inverted file to index segments according to their keywords, using the ids assigned previously based on the spatial and temporal ordering.

The steps for evaluating an `STK` query $Q = (R, T, \Psi)$ using the `IFST` index are described below.

1. The quadtree is traversed to find the leaf nodes that overlap with the spatial predicate R.
2. For each leaf node, the traversal continues using the associated R*-tree to identify subsets of the contained segments that also have a timespan overlapping with T. The result is a list of segment ids, which are merged into a smaller set of k disk sweeps.
3. The inverted index is used to identify the posting lists for the keywords contained in Ψ. The compressed blocks corresponding to the segment ids are read from disk in k disk sweeps and are decompressed. The result is a set of candidate trajectory segments that potentially overlap with R and T and contain at least one of the keywords in Ψ. Then, as with `GKR`, two filtering steps are applied to obtain the result.
4. In the first filtering step, using document-at-a-time (DAAT) processing on the set of segment ids from the R*-trees and the set from the inverted index, the segments that are located outside R or that have their timespan outside T are discarded. This results in a set of candidate objects.
5. In the second filtering step, it is checked for each remaining object whether its segments from Step 4 fully cover the set of query keywords Ψ.

6. EXPERIMENTAL EVALUATION

Datasets. We performed an experimental evaluation using two different types of datasets. The first comprises yacht movement tracking data collected over a period of four weeks from an online yacht tracking service[1], where yacht owners can register their vessels and submit GPS traces along with other information and messages. The second comprises photos from the Flickr Creative Commons dataset provided by Yahoo[2]. For our experiments, we filtered out photos that do not contain coordinates, timestamp or tags, and then we selected 1 million photos within a bounding box around Europe and dates between 2000 and 2010. Table 1 summarizes the characteristics of the two datasets.

Dataset	Size	Objects	Points	Keywords
Yachts	26 MB	1,496	215,937	51,542
Flickr	195 MB	46,016	1,000,000	482,561

Table 1: Datasets used in the experiments.

	Yachts	Flickr
$R(km^2)$	[50K, 250K, **500K**, 750K, 1M]	[1K, **5K**, 10K, 15K, 20K]
$T(hrs)$	[6, 9, **12**, 18, 24]	[2, 4, **6**, 9, 12]
$\|\Psi\|$	[1, 2, **3**, 4, 5]	[1, 2, **3**, 4, 5]

Table 2: Parameters used in the experiments.

Parameters and measures. The experimental evaluation focuses on the following aspects: (a) index construction time and size, and (b) query execution time. We also examine the effect of the following parameters: (a) size of query spatial range R, (b) length of query time interval T, and (c) number of query keywords Ψ. In each experiment, we varied the value of the selected parameter, while setting the rest to default values, as shown in Table 2.

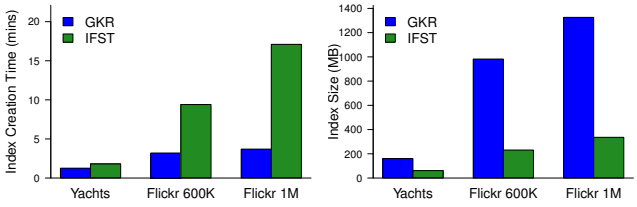
Index construction time and size. We first compare the construction time and size of the `GKR` and `IFST` indexes. The results are shown in Figures 1(a) and 1(b), respectively. The construction time of `IFST` is higher than that of `GKR`. This can be attributed to the overhead caused by the division of the inverted lists into blocks and the compression of the blocks before being written to disk. On the other hand, this extra time spent is compensated by the savings achieved by `IFST` in disk space compared to `GKR`.

Query execution time. Next, we vary the query parameters, i.e, the size of the region R, the length of the time interval T, and the number of keywords Ψ in the query, and we measure the query execution time.

(a) Size of query area. As shown in Figure 2, `GKR` shows superior results with Flickr, whereas `IFST` performs better with the Yachts dataset. This is because, in comparison to `GKR`, the query evaluation time of `IFST` is more affected by the size of the dataset. With more data, the `IFST` quadtree grows deeper as nodes split further to accommodate more objects. As a consequence, the number of leaf nodes that intersect the query region becomes higher, thereby generating the overhead of loading and querying more number of R*-trees. Moreover, the query execution time of both indexes tends to increase as the query area increases. This is due to the fact that the number of objects in the dataset that intersect the query also becomes higher. This increase is more noticeable between certain points, as a result of a

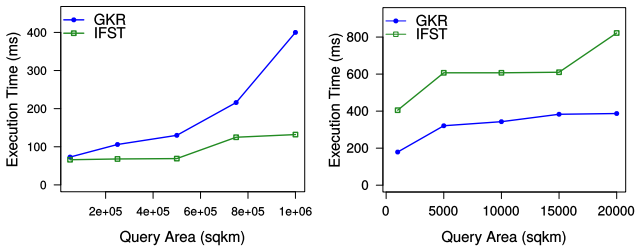
[1] `http://www.yachttrack.org/`

[2] `http://yahoolabs.tumblr.com/post/89783581601/one-hundred-million-creative-commons-flickr-images`



(a) Index creation time (b) Index size

Figure 1: (a) Index creation time and (b) index size for GKR and IFST.



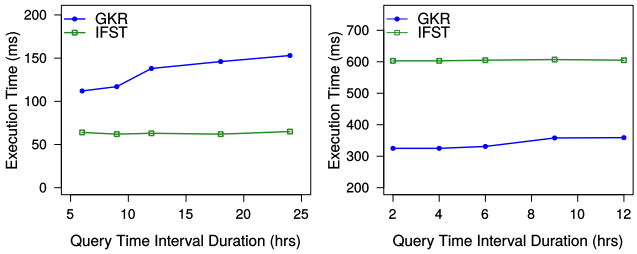
(a) Yachts (b) Flickr

Figure 2: Query execution time vs. query area.

sudden increase in the number of grid cells in GKR and the number of leaf nodes in IFST intersecting the query region. More number of cells and leaf nodes causes a jump in the range of segments that have to be considered and also increases the number of temporal (K)R*-trees that have to be loaded from disk and searched.

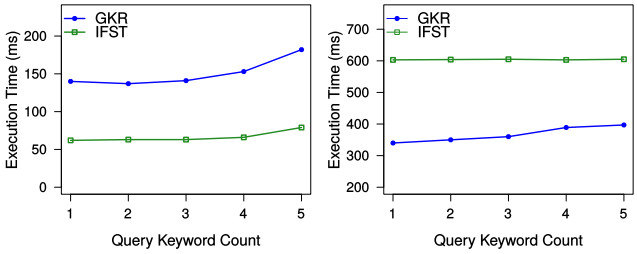
(b) Query time interval. As shown in Figure 3, an increase in the duration of the query time interval also leads to a general increase in the query execution time for both indexes. However, this increase is less pronounced than that produced by increasing the query area, because both indexes use spatial indexes to first filter out data which lies completely outside the query range. Again, in these experiments, GKR outperforms IFST on the Flickr dataset while the latter performs better with the Yachts dataset. It is also noteworthy that in comparison to GKR, the query execution time for IFST varies very little w.r.t. the time interval duration. This is because during the refinement step, IFST uses DAAT processing to find the intersection of the list of segments obtained by querying the temporal R*-trees and those containing at least one query keyword read from the inverted index. On the other hand, in GKR a longer time interval can produce more number of disk pages whose timespans intersect the query time interval. These disk pages then have to be loaded into memory to scan their segments iteratively.

(c) Number of query keywords. As seen in Figure 4, the execution time generally demonstrates an upward trend with an increase in the number of keywords, and the performance of GKR is better than that of IFST with the Flickr dataset, while being worse with the Yachts dataset. Also, varying the number of keywords in the query tends to impact IFST less than GKR. This is again due to DAAT processing in IFST which always chooses the shorter list to iterate over during refinement and perform lookups on the longer list. In case of GKR, while querying the cell KR*-trees, more keywords in the query can produce more disk pages to be read from disk, since all pages containing at least one query keyword and intersecting the query time interval have to be considered.



(a) Yachts (b) Flickr

Figure 3: Query execution time vs. query time interval.



(a) Yachts (b) Flickr

Figure 4: Query execution time vs. number of query keywords.

Acknowledgements

This work was partially supported by the EU H2020 Project City.Risks (H2020-FCT-10-2014-653747).

7. REFERENCES

[1] V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing large trajectory data sets with SETI. In *CIDR*, 2003.

[2] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.

[3] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. space: efficient geo-search query processing. In *CIKM*, pages 423–432, 2011.

[4] G. Cong, H. Lu, B. C. Ooi, D. Zhang, and M. Zhang. Efficient spatial keyword search in trajectory databases. *CoRR*, abs/1205.2880, 2012.

[5] Y. Han, L. Wang, Y. Zhang, W. Zhang, and X. Lin. Spatial keyword range search on trajectories. In *DASFAA*, pages 223–240, 2015.

[6] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *SSDBM*, page 16, 2007.

[7] M. F. Mokbel, T. M. Ghanem, and W. G. Aref. Spatio-temporal access methods. *IEEE Data Eng. Bull.*, 26(2):40–49, 2003.

[8] S. Nepomnyachiy, B. Gelley, W. Jiang, and T. Minkus. What, where, and when: keyword search with spatio-temporal ranges. In *GIR*, pages 2:1–2:8, 2014.

[9] L. Nguyen-Dinh, W. G. Aref, and M. F. Mokbel. Spatio-temporal access methods: Part 2 (2003 - 2010). *IEEE Data Eng. Bull.*, 33(2):46–55, 2010.

[10] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015.