Local Similarity Search on Geolocated Time Series Using Hybrid Indexing

Georgios Chatzigeorgakidis
Dept. of Inf. & Telecommunications
University of Peloponnese, Greece
chgeorgakidis@uop.gr

Dimitrios Skoutas
IMSI, Athena R.C., Greece
dskoutas@imis.athena-innovation.gr

Kostas Patroumpas
IMSI, Athena R.C., Greece
kpatro@imis.athena-innovation.gr

Themis Palpanas
LIPADE, Paris Descartes University, France
themis@mi.parisdescartes.fr

Spiros Athanasiou
IMSI, Athena R.C., Greece
spathan@imis.athena-innovation.gr

Spiros Skiadopoulos
Dept. of Inf. & Telecommunications
University of Peloponnese, Greece
spiros@uop.gr

ABSTRACT

Geolocated time series, i.e., time series associated with certain locations, abound in many modern applications. In this paper, we consider hybrid queries for retrieving geolocated time series based on filters that combine spatial distance and time series similarity. For the latter, unlike existing work, we allow filtering based on local similarity, which is computed based on subsequences rather than the entire length of each series, thus allowing the discovery of more fine-grained trends and patterns. To efficiently support such queries, we first leverage the state-of-the-art BTSR-tree index, which utilizes bounds over both the locations and the shapes of time series to prune the search space. Moreover, we propose optimizations that check at specific timestamps to identify candidate time series that may exceed the required local similarity threshold. To further increase pruning power, we introduce the $\mathcal{S}$BTSR-tree index, an extension to BTSR-tree, which additionally segments the time series temporally, allowing the construction of tighter bounds. Our experimental results on several real-world datasets demonstrate that $\mathcal{S}$BTSR-tree can provide answers much faster for all examined query types.

CCS CONCEPTS

• **Information systems → Spatial-temporal systems**.

KEYWORDS

local similarity, geolocated time series, hybrid indexing

ACM Reference Format:
Georgios Chatzigeorgakidis, Dimitrios Skoutas, Kostas Patroumpas, Themis Palpanas, Spiros Athanasiou, and Spiros Skiadopoulos. 2019. Local Similarity Search on Geolocated Time Series Using Hybrid Indexing. In *27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '19), November 5–8, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3347146.3359349

SIGSPATIAL '19, November 5–8, 2019, Chicago, IL, USA

ACM ISBN 978-1-4503-6909-1/19/11…$15.00
https://doi.org/10.1145/3347146.3359349

1 INTRODUCTION

A *time series* is a time-ordered sequence of data points. Time series are ubiquitous in many application domains. They can represent various types of measurements, such as user check-ins at various Points of Interest, energy consumption in smart buildings, PM2.5 particle concentration measured by air pollution sensors, etc. Analyzing and mining time series data is highly important for discovering trends and patterns in such phenomena, and has attracted extensive research interest over the last years [7, 12, 19].

However, what is usually overlooked is that the phenomena represented by time series are often also associated with geographic locations, e.g., time series generated by sensors installed at fixed positions. In such cases, spatial distance also plays an important role in the analysis, since discovery of trends and patterns may depend not only on time series similarity but also on geographic proximity. Motivated by this observation, in previous work [5, 6] we introduced the concept of *geolocated* time series and we proposed hybrid indexing techniques that efficiently support the retrieval of time series based on both spatial distance and time series similarity.

In particular, we introduced the BTSR-tree [6], a *hybrid index* that first builds an R-tree over the locations of the time series data. It then enhances each node with appropriate upper- and lower-bounding time series (MBTS) that enclose the subset of time series represented by it. Combining MBTSs and MBRs, the query evaluation algorithm can simultaneously prune the search space based on time series similarity and spatial distance while traversing the index. To further increase its pruning power, the BTSR-tree groups together similar time series within each node to derive tighter bounds.

This existing approach for hybrid search over geolocated time series using the BTSR-tree supports only *global* time series similarity, i.e., similarity measured across the entire length of time series. Specifically, as in other works in this area [2, 3, 7, 10], the distance between two time series is measured by aggregating the pairwise Euclidean distance of their respective values across the entire sequences. However, in many cases, more fine-grained trends and patterns may exist, which are missed under this global similarity measure. For example, consider two time series representing the hourly energy consumption of two nearby buildings over a week, and assume that the two buildings exhibit a similar consumption pattern during working days but a different one in weekends. A query imposing a similarity threshold over the entire week would

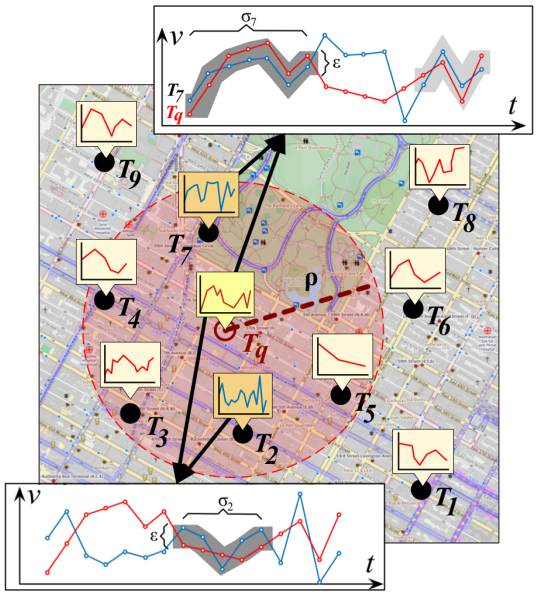


Figure 1: Example hybrid query.

fail to identify these two geolocated time series as similar. However, it may be useful to discover that there is a period of up to 5 days during which these two time series are actually similar.

Motivated by this observation, in this work we extend our previous approach on hybrid queries over geolocated time series to support *local similarity* of time series, thus allowing more flexible and fine-grained queries and analyses. The *local similarity score* between two time series T_i and T_j is defined as the maximum number of consecutive timestamps during which the respective values of T_i and T_j do not differ by more than a user-specified threshold ϵ. Notice that, compared to global similarity, this condition is more relaxed, in the sense that it is applied to subsequences of length lower than T_i and T_j, but at the same time stricter, in the sense that the threshold ϵ is required to be satisfied at each individual timestamp during the selected period rather than on the aggregate distance over all timestamps.

Combining this local similarity constraint with a filter on *spatial distance* leads to a novel set of hybrid queries. Figure 1 shows an example with a query time series T_q searching over a set of time series $T_1, \ldots, T_9$ for those within radius ρ from its location and also locally similar to T_q. In particular, with respect to a given ϵ, results should also be locally similar to T_q for at least 5 consecutive timestamps. Qualifying results include T_2 with local similarity score $\sigma_2 = 5$ (bottom chart), and T_7 with $\sigma_7 = 7$ (top chart).

It turns out that such hybrid queries involving local similarity can still be evaluated using the BTSR-tree index. We first present a baseline method employing a sweep-line algorithm to check for local similarity, and then describe how this can be optimized by using appropriately placed *checkpoints*, based on the local similarity score threshold specified by the query, in order to skip unnecessary comparisons. Despite the fact that this saves some computations, the resulting time savings are relatively small, since the number of index nodes that need to be probed is not essentially reduced. To overcome this problem, we introduce an improvement to the BTSR-tree index, which is based on temporally segmenting the time series bounds within each node and deriving tighter bounds per segment. Once the time series bounds in each node become more fine-grained, pruning the search space for local similarity queries proves much more effective.

Summarizing, our main contributions are as follows:

- We extend our previous work on hybrid queries for geolocated time series to support local time series similarity. We consider both range and top-k queries, including combined criteria for spatial distance and local time series distance.
- We present how such queries can be answered efficiently exploiting the previously introduced BTSR-tree index.
- To achieve greater savings in execution time by further reducing node accesses, we propose an enhanced variant of BTSR-tree, called $\mathcal{S}$BTSR-tree, which additionally employs temporal segmentation in each node to derive tighter, more fine-grained time series bounds.
- We experimentally evaluate our methods using real-world datasets from different application domains, showing that BTSR-tree can efficiently handle hybrid queries under local similarity search, while $\mathcal{S}$BTSR-tree achieves even higher performance due to the additional temporal segmentation.

The remainder of the paper is structured as follows. Section 2 reviews related work, while Section 3 formally defines the problem. Section 4 presents how query evaluation under local time series similarity can be executed using the BTSR-tree. Then, Section 5 presents the enhanced $\mathcal{S}$BTSR-tree. Section 6 reports our experimental results and Section 7 concludes the paper.

2 RELATED WORK

Similarity search over time series has provided a wide range of algorithmic approaches; a detailed survey with experimental evaluation is available in [7]. Initially, the focus was mostly on wavelet-based methods [4] to reduce the dimensionality of time series and generate an index based on the transformed sequences. In contrast, state-of-the-art approaches for time series indexing are based on the *Symbolic Aggregate Approximation* (SAX) representation [10]. The first index in this family was *i*SAX[16], offering multi-resolution representations for time series. Further extensions like *i*SAX 2.0 [2], *i*SAX2+ [3], ADS+ [20], Coconut [9], DPiSAX [17], and ParIS [13] provided a wide range of advanced capabilities. These indices support *global* similarity search, i.e., the similarity score is computed over the entire length of the compared time series, as opposed to *local* similarity, which allows to consider similar subsequences. The most recent addition to this *SAX*-based family is *ULISSE* [11], which can answer similarity search queries of *varying* length. However, this still differs from our setting, since in *ULISSE* the goal is to build an index that supports similarity search for queries of any length within a given range $[\ell_{min}, \ell_{max}]$. Furthermore, none of the aforementioned approaches supports geolocated time series, and thus cannot efficiently process hybrid queries combining conditions on spatial distance and time series similarity.

The problem of *subsequence matching* over time series is to identify matches of a (relatively short) query subsequence across one or more (relatively long) time series. The UCR suite [14] offers

a framework comprising various optimizations regarding subsequence similarity search. Matrix Profile [18] includes methods for detecting, for each subsequence of a time series, its *nearest neighbor* subsequence, by keeping track of Euclidean distances among candidate pairs. Applying such approaches in our setting is not straightforward. First, they involve Euclidean or DTW distances, which are different from our definition of local similarity score, hence the pruning heuristics do not hold in our case. Second, they do not consider geolocated time series, thus spatial filtering has to be carried out independently, which reduces pruning opportunities.

To the best of our knowledge, the only index that supports searching over geolocated time series is the BTSR-tree [5, 6]. It is a spatial-first index based on the R-tree that can additionally compute bounds on similarity of time series instead of a textual similarity between documents. Apart from an MBR, each node also stores bounds over the time series indexed in its subtree. Thus, it offers increased pruning capabilities for range and top-k queries involving both time series similarity and spatial proximity. In the current work, we show how BTSR-tree can be used for another family of hybrid queries involving *local similarity* of time series. Furthermore, we introduce a variant structure, called $\mathcal{S}$BTSR-tree, which constructs tighter bounds over temporally segmented time series to offer stronger pruning power.

3 LOCAL SIMILARITY SEARCH ON GEOLOCATED TIME SERIES

Next, we briefly present some background on geolocated time series and the BTSR-tree index, and then formally define the problem.

3.1 Preliminaries

Geolocated Time Series. A *time series* is a time-ordered sequence of values $T = \{T^1, T^2 \ldots, T^n\}$, where T^i is the value at the i-th timestamp and n is the length of the series. A geolocated time series is additionally characterized by a *location*, denoted by $T.loc$. The *spatial distance* d between two geolocated time series is the Euclidean distance of their respective locations.

The BTSR-tree Index. In [6], we have introduced the BTSR-tree index, which is based on the notion of *Minimum Bounding Time Series* (MBTS). In a similar manner that an MBR encloses a set of geometries, an MBTS encloses *a set of time series* $\mathcal{T}$ using a pair of bounds that fully contain all of them. Figure 2 depicts an example of two MBTSs for two disjoint sets of time series. Formally, given a set of time series $\mathcal{T}$, its MBTS consists of an *upper bounding time series* $B^{\sqcap}$ and a *lower bounding time series* $B^{\sqcup}$, constructed by respectively selecting the maximum and minimum of values at each timestamp $i \in \{1, \ldots, n\}$ among all time series in set $\mathcal{T}$ as follows:

$$B^{\sqcap} = \{\max_{T \in \mathcal{T}} T^1, \ldots, \max_{T \in \mathcal{T}} T^n\}$$
$$B^{\sqcup} = \{\min_{T \in \mathcal{T}} T^1, \ldots, \min_{T \in \mathcal{T}} T^n\} \quad (1)$$

A BTSR-tree index is initialized as an R-tree [8] built on the spatial attributes of the given geolocated time series dataset, as depicted in the example of Figure 3. Besides MBRs, each node is enhanced to also store MBTSs, shown as colored strips per node in Figure 3c. This enables efficient pruning of the search space when evaluating hybrid queries combining time series similarity

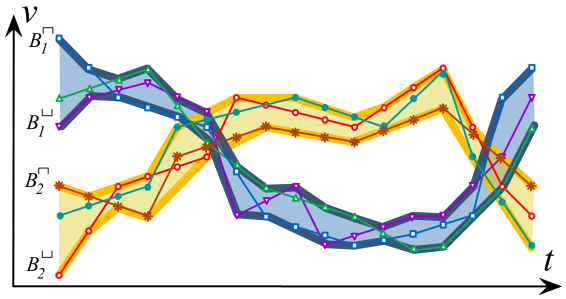
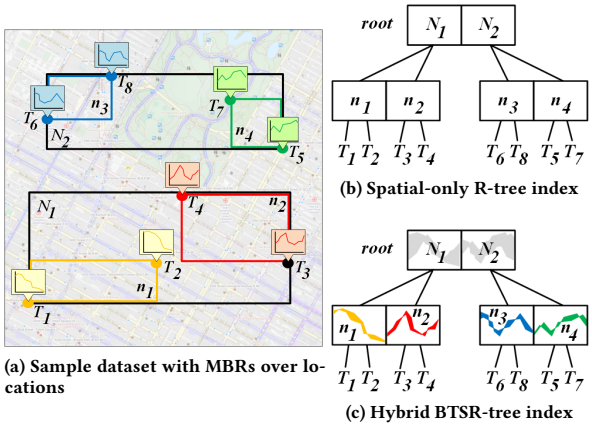


Figure 2: MBTS constructed for two sets of time series.



(a) Sample dataset with MBRs over locations

(b) Spatial-only R-tree index

(c) Hybrid BTSR-tree index

Figure 3: The BTSR-tree index.

with spatial proximity. For each child, a node stores a pre-specified number of MBTSs. Each MBTS is calculated according to Eq. 1. Construction and maintenance of the BTSR-tree follow the procedures of the R-tree for data insertion, deletion and node splitting. Objects (i.e., geolocated time series) are inserted into leaf nodes and any resulting changes are propagated upwards. Once the nodes have been populated, the MBTS of each node are calculated bottom-up, relying on *k-means clustering* according to their Euclidean distance in the time series domain. The example in Figure 2 depicts the $k = 2$ MBTSs (as two bands with a thick outline) obtained for a set of time series (shown as thin polylines). In a BTSR-tree, each parent node receives all the MBTSs of its children and computes its own k MBTSs. The process continues upwards, until reaching the root.

3.2 Problem Definition

We first define the local similarity between time series, and then present the query variants we consider in this paper.

Definition 1 (Local Time Series Similarity). *The* local similarity score σ *between two time series* T *and* T' *is the maximum count of consecutive timestamps during which the respective values of* T *and* T' *do not differ by more than a given margin* ϵ*, i.e.,* $\sigma(T, T', \epsilon) = |I_{max}|$*, where* I_{max} *is the longest consecutive time interval* I *such that* $\forall i \in I, |T^i - T'^i| \leq \epsilon$.

In this work, our goal is to efficiently support hybrid queries on geolocated time series that retrieve the results based both on

spatial proximity and local similarity. Specifically, we focus on the following types of queries (hereafter referred to as *LS-queries*):

- $Q_{rr}(T_q, \rho, \epsilon, \delta)$: Given a geolocated time series T_q, retrieve every geolocated time series T such that T is located within range ρ from T_q, i.e., $d(T_q, T) \leq \rho$ and has local similarity to T_q at least δ, i.e., $\sigma(T_q, T) \geq \delta$.
- $Q_{kr}(T_q, k, \epsilon, \delta)$: Given a geolocated time series T_q, retrieve the spatial k-nearest neighbors to T_q that also have local similarity to T_q at least δ.
- $Q_{rk}(T_q, \rho, \epsilon, k)$: Given a geolocated time series T_q, retrieve the top-k geolocated time series that have the highest local similarity to T_q with respect to ϵ and are located within range ρ from T_q.

Example 1. *Figure 1 depicts an example of the $Q_{rr}(T_q, \rho, \epsilon, \delta)$ query. Given the geolocated time series T_q as query, we seek the spatially close ones (i.e., within a circle of radius ρ) that are also locally similar within margin ϵ for at least δ timestamps. In this example, despite five geolocated time series being within range, only T_2 and T_7 qualify for the final result, since these are the ones that are also locally similar for at least one time interval of length at least δ.*

4 LS-QUERIES USING THE BTSR-TREE

A straightforward approach for answering LS-queries would be to use a spatial index to first filter by spatial distance and then perform a sequential scan across each result to filter out those having local similarity score below the given threshold. This suffers from generating an unnecessarily large number of intermediate results which are then discarded. Instead, we propose to process LS-queries by leveraging the BTSR-tree index [6], which can prune the search space simultaneously according to both criteria.

While traversing the BTSR-tree, *spatial filtering* is performed at each node N by computing the *bounding distance* $mindist_{sp}$ between the location of T_q and the MBR of N, as in R-Trees [15].

For *time series similarity*, we exploit the MBTS stored within each node. Considering an MBTS at a node N, we calculate its distance $mindist^i_{ts}$ from T_q at each timestamp i as:

$$mindist^i_{ts}(T_q, MBTS_N) = \begin{cases} T^i_q - B^{\sqcap_i}_N, & \text{if } T^i_q > B^{\sqcap_i}_N \\ B^{\sqcup_i}_N - T^i_q, & \text{if } T^i_q < B^{\sqcup_i}_N \\ 0, & \text{if } B^{\sqcap_i}_N \leq T^i_q \leq B^{\sqcup_i}_N \end{cases} \quad (2)$$

where $B^{\sqcap_i}_N$ and $B^{\sqcup_i}_N$ are the upper and lower values of the MBTS at timestamp i. By definition of MBTS, no time series indexed under N can differ from T_q by less than $mindist^i_{ts}$ at timestamp i. Hence, only at those timestamps that $mindist^i_{ts} \leq \epsilon$, it is possible that a time series indexed under N is locally similar to T_q. Subsequently, we can compute a *local similarity bound* σ_B:

$$\sigma_B(T_q, MBTS_N, \epsilon) = max\{|I|; \forall i \in I, mindist^i_{ts}(T_q, MBTS_N) \leq \epsilon\}. \quad (3)$$

that reflects the *maximum* interval I of consecutive timestamps where the distance computed by Eq. 2 does not exceed margin ϵ. This value is an upper bound of the local similarity scores of T_q with any time series enclosed in this MBTS. Figure 4 shows that T_q deviates from the given MBTS by no more than ϵ during two intervals: one consisting of $|I_1| = 5$ consecutive timestamps and a smaller one with only $|I_2| = 2$ timestamps (shown as square points). So, the local similarity bound for this MBTS is $\sigma_B = 5$.

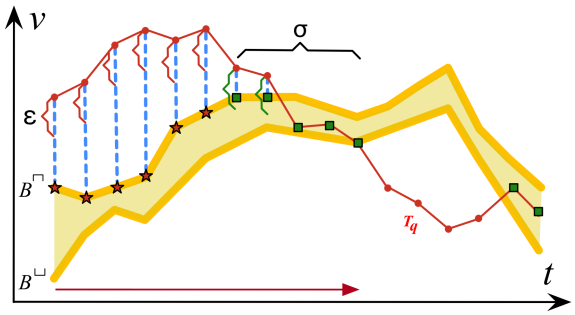


Figure 4: Local similarity check against an MBTS.

By construction, the MBTSs of a child node N' get tighter bounds compared to those of its parent N as we descend the BTSR-tree. It is easy to verify that

$$\sigma_B(T_q, MBTS_N, \epsilon) \geq \sigma_B(T_q, MBTS_{N'}, \epsilon) \quad (4)$$

hence local similarity bounds can only diminish when descending the index. This bound provides a useful pruning condition during search with a cutoff threshold δ. Any node where all its MBTSs have local similarity bound σ_B below δ can be safely pruned.

Next, we describe a baseline approach that employs a sequential scan over MBTSs, and then we present an optimization that prioritizes selected *checkpoints* to avoid many point-wise comparisons.

4.1 Sweep Line Approach

We explain how the BTSR-tree can be used, in conjunction with a simple sweep-line algorithm, to answer each of the three LS-queries, taking advantage of the two types of bounds, $mindist_{sp}$ and $mindist_{ts}$, described above.

$Q_{rr}(T_q, \rho, \epsilon, \delta)$: We traverse the BTSR-tree starting from its root. At each inner node N, we first check whether $mindist_{sp}(T_q, MBR_N) \leq \rho$. If so, we employ a sweep line across the time axis to compute the local similarity bound $\sigma_B(T_q, MBTS_N, \epsilon)$ for every MBTS included in N. If *all* resulting bounds σ_B are below δ, the subtree under N is pruned. Otherwise, the search continues at the children. Upon reaching a leaf node, we fetch the geolocated time series contained therein, and verify the query constraints against each one. Each T such that $d(T_q.loc, T.loc) \leq \rho$ and $\sigma(T_q, T, \epsilon) \geq \delta$ is added to the results.

$Q_{kr}(T_q, k, \epsilon, \delta)$: We maintain a priority queue P containing both inner nodes (sorted by ascending $mindist_{sp}$) and geolocated time series (sorted by ascending spatial distance to T_q). We start by adding to P the root of BTSR-tree. In each iteration, we retrieve the top element from P. If it is an inner node, we visit its children to calculate local similarity bounds σ_B according to Eq. 3. For any child N that σ_B of one of its MBTSs satisfies threshold δ, we search the subtree of N. Then, we calculate the corresponding spatial distance ($mindist_{sp}$ for a node N or Euclidean distance for a geolocated time series T) and insert it back to P. Once we encounter a geolocated time series T at the top of P, we add it to the results. The process terminates once k geolocated time series have been obtained.

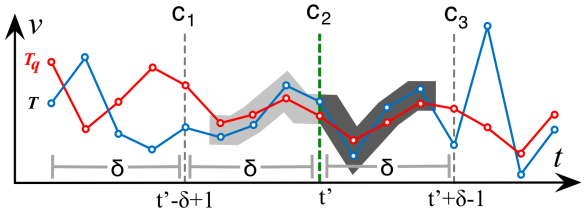


Figure 5: Local time series similarity via checkpoints.

$\boldsymbol{Q_{rk}(T_q, \rho, \epsilon, k)}$: This query is evaluated similarly to the previous one, with two differences. The first difference is that the priority queue P is now sorted based on local similarity bounds in descending order, instead of spatial distance bounds in ascending order. The second is that before inserting an item (node or time series) to P, its spatial distance ($mindist_{sp}$ or exact) is calculated, and if it is higher than ρ the item is skipped. The traversal starts again from the root, and terminates once k time series have been retrieved from the top of P. These are the top-k results with respect to local similarity (if another time series T had higher local similarity, it would have been retrieved from P first), and they are located within range ρ from T_q (otherwise, they would not have been admitted to P).

4.2 Checkpoint Approach

The drawback of the sweep-line approach is that it needs to perform a comparison for each individual timestamp to eventually determine the exact or maximum local similarity of a given time series or node, respectively. In the following, we explain how we can use *checkpoints* along the time axis to avoid this exhaustive search. These checkpoints prioritize specific timestamps when checking for candidate matches to eagerly filter out non-qualifying items.

Assume a query with local similarity threshold δ. We can place checkpoints at every δ timestamps, and only apply the local similarity filter (i.e., $|T_q^i - T^i| \leq \epsilon$) at those. If no checkpoint satisfies the condition, this item can be safely pruned since it cannot have local similarity to T_q at least δ (as this would require the condition to be true for at least δ consecutive timestamps, thus crossing at least one checkpoint).

Figure 5 shows an example with checkpoints placed along the time axis every $\delta = 5$ timestamps. For clarity, we consider a single time series T. Assume a checkpoint at timestamp t' and a minimal duration δ starting at timestamp $t' - \delta + 1$ for asserting local similarity with query T_q, as shown with the light grey strip in the figure. This interval cannot have smaller duration, as it would not satisfy the δ constraint. Thus, the local similarity condition will be true at checkpoint t'. Similarly, if such an interval ends at timestamp $t' + \delta - 1$ (darker shaded grey strip in Figure 5), it will be detected at the checkpoint at t'. Thus, it suffices to check for local similarity only at checkpoints, i.e., every δ timestamps. We denote the set of checkpoints as C, determined at query time. If a checkpoint satisfies the condition, then we need to scan both forward and backward from it to determine the actual local similarity score, i.e., to find the exact extent of the time interval for which the condition holds.

Figure 6 exemplifies the use of checkpoints for comparing T_q to an MBTS of a node for $\delta = 5$ timestamps. Instead of sequentially performing 11 comparisons until verifying that local similarity score σ is at least δ (i.e., we stop the verification at $t = 11$, once

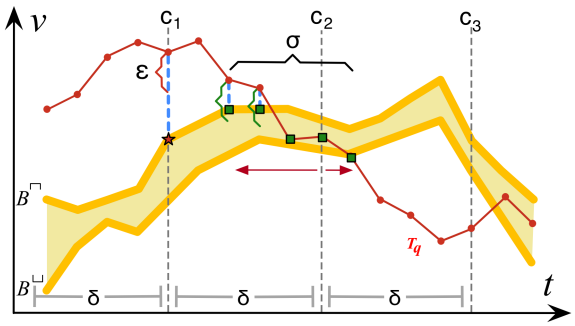


Figure 6: Local similarity with a MBTS using checkpoints.

$\sigma = 5$), we check only around the checkpoints. At the leftmost checkpoint c_1, no local similarity is found (T_q is farther than ϵ from the MBTS), so we skip directly to checkpoint c_2. Since T_q differs by less than ϵ at c_2, we need to compare values backward and forward, up to the previous and next checkpoint, respectively. This requires only 6 comparisons instead of 11 to decide that this node may contain candidates. Next, we describe how probing with checkpoints is applied during evaluation of LS-queries.

$\boldsymbol{Q_{rr}(T_q, \rho, \epsilon, \delta)}$: Algorithm 1 outlines the procedure. Initially, we obtain the children of the root node in a list and place the checkpoints every δ timestamps (Lines 1-2). We iterate over each item N in this list. If N is an inner node, we have to examine whether both constraints with respect to ρ and δ are met for each of its children. Verification of MBTS against query T_q will be discussed shortly. If this is the case, we traverse the sub-tree of each child in the same manner, by adding it to the list (Lines 6-10), thus descending the tree. If the examined node is a leaf (Line 11), we iterate over each contained time series T to check the constraints ρ and δ. If T qualifies, it is added to the results (Lines 12-14). Note that now the calculation of local similarity scores for geolocated time series is based on checkpoints (Line 13), as discussed above.

Verification of MBTS against the local similarity constraints ϵ, δ is applied using checkpoints (Lines 16-37). This verification concerns each MBTS in a given node N'. At each checkpoint c, we first verify whether its $mindist_{ts}^{c}$ to query T_q is at most ϵ (Line 19). If so, we first scan backward to inspect whether there are at least δ consecutive timestamps where T_q deviates by at most ϵ from this MBTS (Lines 21-28). Similarly, we probe forward from checkpoint c (Lines 29-36). In either case, once local similarity no longer holds at a timestamp, probing skips to the next checkpoint. If the check fails for all checkpoints of all MBTSs, then this node cannot contain any results (Line 37).

$\boldsymbol{Q_{kr}(T_q, k, \epsilon, \delta)}$: We follow a similar procedure to the one in Section 4.1 for query Q_{kr}, employing the same verification process over MBTSs and time series as in Algorithm 1. Algorithm 2 describes the procedure. We start by adding the root node to a priority queue P based on spatial distance (Line 1). After determining the checkpoints using the given δ (Line 2), we iteratively retrieve elements from P (Line 4). Then, three cases may occur:

(i) If this element is a time series (Lines 5-8), it is guaranteed to be a result, given that P is sorted based on spatial distance from T_q. Indeed, any subsequent element must be located

Algorithm 1: $Q_{rr}(T_q, \rho, \epsilon, \delta)$

```
1  R ← ∅, List ← Root.entries
2  C ← determineCheckpoints(δ)
3  while List ≠ ∅ do
4      N ← List.getNext()
5      if N is not leaf then
6          foreach N' ∈ N.getChildren() do
7              if mindist_sp(T_q, MBR_N') ≤ ρ then
8                  count ← ∅
9                  if VerifyMBTS(T_q, N', C, ϵ, δ) then
10                     List ← List ∪ {N'.getChildren()}
11     else
12         foreach T ∈ N.getObjects() do
13             if d(T_q, T) ≤ ρ ∧ σ^C(T_q, T, ϵ) ≥ δ then
14                 R ← R ∪ {T}
15         return R
16 Procedure VerifyMBTS(T_q, N', C, ϵ, δ)
17     foreach MBTS ∈ N' do
18         foreach c ∈ C do
19             if mindist^c_ts(T_q, MBTS) ≤ ϵ then
20                 count + +, c' ← c
21                 while True do
22                     c' − −
23                     if mindist^c'_ts(T_q, MBTS) ≤ ϵ then
24                         count + +
25                         if count ≥ δ then
26                             return True
27                     else
28                         break
29                 while True do
30                     c + +
31                     if mindist^c_ts(T_q, MBTS) ≤ ϵ then
32                         count + +
33                         if count ≥ δ then
34                             return True
35                     else
36                         break
37     return False
```

farther than the current. When list R obtains the required number k of results, the search terminates.

(ii) The element is a leaf node (Lines 9-13): In this case, we obtain each time series T contained in this leaf, and verify the local similarity score of T against δ. If the condition is met, we calculate the spatial distance of candidate T from query T_q and push T into the priority list along with its spatial distance.

(iii) If the element is an inner node, we iterate over its children and only push back to the queue the ones whose MBTSs are verified against ϵ and δ using checkpoints (Lines 13-18).

$\boldsymbol{Q_{rk}(T_q, \rho, \epsilon, k)}$: The procedure for this query is listed in Algorithm 3. Notice that for employing checkpoints, we need a local similarity threshold δ, so as to determine their placement, but this query does not specify a fixed δ. To be able to obtain one during search, we now maintain two priority queues: P holds inner nodes sorted by local similarity bounds (Eq. 3), while R keeps up to k geolocated time series sorted by local similarity scores (as in Def. 1).

Algorithm 2: $Q_{kr}(T_q, k, \epsilon, \delta)$

```
1  R ← ∅, P.push(Root)
2  C ← determineCheckpoints(δ)
3  while P is not empty do
4      N ← P.poll()
5      if N is raw then
6          R ← R ∪ {N}
7          if |R| = k then
8              break
9      else if N is leaf then
10         foreach T ∈ N.getObjects() do
11             if σ^C(T_q, T, ϵ) ≥ δ then
12                 T.dist ← d(T_q, T)
13                 P.push(T, T.dist)
14     else
15         foreach N' ∈ N.getChildren() do
16             if VerifyMBTS(T_q, N', C, ϵ, δ) then
17                 N'.dist ← mindist_sp(T_q, MBR_N')
18                 P.push(N', N'.dist)
19 return R
```

We initially set $\delta = 1$, so checkpoints are trivially placed at every timestamp. This implies that computation of local similarity scores with $\delta = 1$ is equivalent to the sweep line approach. However, δ increases with the detection of qualifying results, hence checkpoints will progressively get placed more sparsely. The search starts by adding the BTSR-tree root in P (Line 1). We iteratively poll the top element from P, and there are two possible cases:

(i) The top element is a leaf node. Then, we iterate over the contained time series and add the ones that satisfy the spatial condition (ρ) to R, along with their corresponding local similarity score σ if it exceeds the current value of δ (Lines 7-11). Once R exceeds capacity k, its last element is evicted to make room for the newly inserted one and δ is updated according to the local similarity score σ_k of the k-th element in R. In this case, the placement of checkpoints is re-adjusted according to the increased δ value (Lines 12-15).

(ii) The top element is an inner node. In this case, we iterate over each child N' and check if $mindist_{sp}(T_q, MBR'_N) \leq \rho$. If N' qualifies, we calculate the local similarity bound σ_B of all its MBTSs using checkpoints. If the maximum among these bounds $max(\sigma_B) \geq \delta$, then N' is inserted to P with this maximum score (Lines 16-24).

The process terminates once the top element in P has local similarity less than δ (Lines 5-6). The result is the contents of R.

5 THE $\mathcal{S}$BTSR-TREE INDEX

5.1 Index Structure

The BTSR-tree index uses k-means clustering to cluster the time series under each node and then stores the MBTSs of those clusters. However, clustering entire time series typically generates many overlapping MBTSs, incurring much dead space. This has a negative impact on the pruning power of the index, especially when considering local similarities. Figure 7a depicts such a case of six time series indexed in a node. A k-means clustering with $k = 3$ will form the depicted MBTSs denoted with shaded colors. As a

Algorithm 3: $Q_{rk}(T_q, k, \rho)$

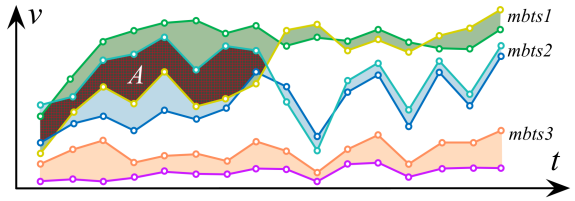
```
1  R ← ∅, P.push(Root)
2  δ ← 1
3  C ← determineCheckpoints(δ)
4  while P is not empty do
5      if P.peekFirst.σ_B < δ then
6          break
7      if N is leaf then
8          foreach T ∈ N.getObjects() do
9              if d(T_q, T) ≤ ρ then
10                 if σ^C(T_q, T, ε) ≥ δ then
11                     R.push(T, σ^C(T_q, T, ε))
12             if R.size > k then
13                 R.pollLast
14                 δ ← R.peekLast.σ
15                 C ← determineCheckpoints(δ)
16     else
17         foreach N' ∈ N.getChildren() do
18             if mindist_SP(T_q, MBR_N') ≤ ρ then
19                 σ_B ← 0
20                 foreach MBTS ∈ N' do
21                     if σ_B^C(T_q, MBTS, ε) ≥ σ_B then
22                         σ_B ← σ_B^C(T_q, MBTS, ε)
23                 if σ_B ≥ δ then
24                     P.push(N', σ_B)
25 return R
```

result, the dark area A represents the overlap between $mbts.1$ and $mbts.2$ and actually makes those bounds less tight. Hence, such MBTSs inflate estimates for local similarity bounds, and thus lead to unnecessarily descending further down the index.

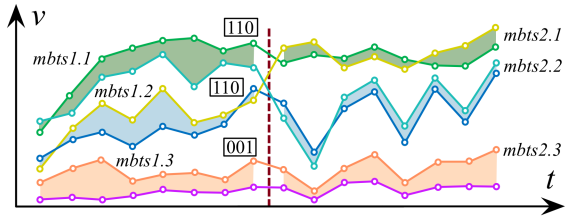
To reduce the amount of overlap within the MBTSs of nodes, we introduce an extended version of the BTSR-tree, named $\mathcal{S}$BTSR-tree. $\mathcal{S}$BTSR-tree attempts to eliminate as much overlap as possible, through segmentation of time series. Figure 7b depicts the intuition. If we segment the time series before applying k-means, the resulting MBTSs for each segment tend to be tighter, eliminating the excessive overlap A from Figure 7a. The $\mathcal{S}$BTSR-tree is built similarly to BTSR-tree. The only difference is that the MBTSs of each node are calculated *per segment*. In this method, we assume a pre-defined number s of segments, but segmentation is orthogonal to our problem and can be carried out by applying existing methods like [1]. Ultimately, $\mathcal{S}$BTSR-tree allows for more aggressive pruning when traversing the index.

5.2 Cross-Segment Continuity Via Bit-Vectors

A downside of the segmentation approach is the loss of the MBTS continuity across time, which results in MBTSs enclosing different time series in neighboring segments. For example, in Figure 7b, there are no MBTSs in the right segment containing the same time series as $mbts1.1$ and $mbts1.2$, a fact which hinders the calculation of local similarity on the segment boundaries (the vertical line). To overcome this, we introduce a *bit-vector* V along each MBTS of a segment, having one bit for each MBTS created. If in the current segment a bit in vector V of a given MBTS is set, this indicates that this MBTS encloses at least one common time series with another MBTS′ in the next segment. In the example shown in Figure 7b,



(a) Example of a node's MBTS.



(b) Segmenting can eliminate whitespace.

Figure 7: Segmenting time series yields tighter MBTS.

$V = 110$ for $mbts1.1$ indicates common time series with $mbts2.1$ and $mbts2.2$ in the next segment, while $V = 001$ for $mbts1.3$ signifies common time series with only $mbts2.3$. This way, to calculate local similarity, we can easily identify all the MBTSs that share common time series among two successive segments.

To evaluate LS-queries, traversal of the $\mathcal{S}$BTSR-tree index follows a similar rationale to the procedure in Section 4.2. For each checkpoint c, we first obtain the segment where it falls in, and we scan each MBTS leftward and rightward from c, as discussed in Section 4.2. If we cross the border to another segment, the available bit-vectors directly identify the MBTS that need be examined in this neighboring segment. This propagates until the local similarity constraints (ϵ and δ) are satisfied. Figure 8 illustrates an example of a node verification. Let us consider a predetermined number of three segments and the corresponding MBTS of each segment for that node. Suppose that there exists a checkpoint c on the second segment. To verify whether this node satisfies the local similarity constraints, we start from checkpoint c and we check leftwards whether $mindist^i_{ts} \leq \epsilon$ for each timestamp. If the currently examined timestamp falls in the first segment, we fetch the corresponding MBTS and bit-vectors and continue checking whether $mindist^i_{ts} \leq \epsilon$ in both MBTS (green shaded), as their bit-vectors both indicate common members with the first one in segment 2. A similar procedure is followed rightwards, where we only have to check the first MBTS, according to the bit-vectors.

5.3 Cost Analysis

Next, we analyze the cost of the Q_{rr} query (the other queries have similar costs). For index traversal, since the index is an augmented R-tree, the basic cost for searching over an R-tree applies here as well [8]. However, there is an extra cost which involves two parts. The first part concerns MBTS verification. Assume a query time series T_q of length n that is verified against the MBTS of a node N. For each checkpoint, the algorithm checks for each timestamp t

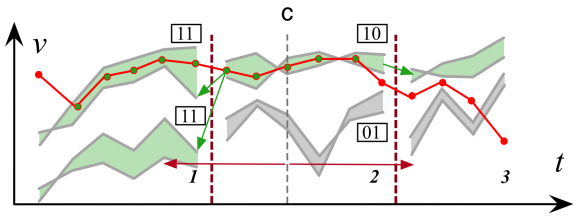


Figure 8: Example of verifying a $\mathcal{S}$BTSR-tree node.

Table 1: Datasets and parameters used in the experiments.

Dataset	Area (km^2)	Number of locations	Length of timeseries	Default query parameters ρ	ϵ	δ	k
Flickr	Earth	414,967	96	30%	7.5%	20	30
Crime	392,000	362,215	76	30%	7.5%	25	30
Taxi	2,500	417,960	168	30%	10%	20	30

among two segments, whether the mindist at t between T_q and the node's MBTS is less than ϵ (see Equation 4). This is repeated at each neighboring segment for each MBTS whose bit vector is 1, until threshold δ is satisfied, or rejected for all checkpoints. Thus, this extra cost is $O(c * b^2 * s * g)$ in the worst case, where c is the number of checkpoints, b is the number of MBTS, s the number of segments, and $g = n/s$ the number of timestamps between two segments. In practice, this will typically require much fewer comparisons, since the node is traversed only when a qualifying interval is found. The second part of extra cost concerns time series verification. To verify T_q against T, the algorithm needs to check for each timestamp t whether the value difference between T_q and T is less than ϵ, and keep the largest detected one; hence, this extra cost is $O(n)$.

6 EXPERIMENTAL EVALUATION

Next, we report results from a comprehensive evaluation of our methods against real-world datasets.

6.1 Experimental Setup

6.1.1 Datasets. We use three real-world datasets (Table 1) selected from different application domains, containing diverse types of geolocated time series, as detailed below:

UK historical crime data (Crime). Contains time series representing the temporal variation in the number of crime incidents reported across England and Wales over 76 months (December 2010– March 2017). We generated time series over a grid with cell size 200 meters applied on the original data[1]. For each month, we counted incidents having their location within each cell.

Flickr geotagged photos (Flickr). Contains time series data extracted from geolocated Flickr images between 2006 and 2013 over the entire planet[2]. To get meaningful geolocated time series, we partitioned the space by a uniform grid of 7200 × 3600 cells (each one spanning 0.05 decimal degrees in each dimension) and counted the number of photos contained in every cell each month. We excluded empty cells (e.g., in the oceans). Each time series conveys the visits pattern (in terms of number of photos taken per month) of that region over this period.

NYC taxi drop-offs (Taxi). Contains time series extracted from yellow taxi rides in New York City during 2015. The original data[3] provide pick-up and drop-off locations, as well as corresponding timestamps for each ride. For each month, we generated time series by applying a uniform spatial grid over the entire city (cell side was 200 meters) and counting all drop-offs therein for each day of the week at the time granularity of one hour. Thus, we obtained the number of drop-offs for 24×7 time intervals in every cell, which essentially captures the weekly fluctuation of taxi destinations there. Without loss of generality, the centroid of each cell is used as the geolocation of the corresponding time series.

Synthetic. To test scalability, we augmented the Flickr dataset by slightly moving each location in a random manner and altering each time series value by a random number between 1 and 10. We produced three additional synthetic datasets each containing ×2, ×3, ×4 the number of time series from the original dataset.

6.1.2 Index and Query Parameters. To evaluate the performance benefits observed in the experiments only based on pruning, we tuned the index parameters to fixed values. The minimum (m) and maximum (M) number of entries stored in each node are set to 40 and 100, respectively. For both BTSR-tree and $\mathcal{S}$BTSR-tree, the number of MBTS set to 10 and for $\mathcal{S}$BTSR-tree, the number of segments s is also set to 10. The query parameters involve the spatial distance and local similarity thresholds, i.e., ρ, ϵ, δ and k. The values of these parameters are set differently for each dataset, based on their characteristics; default values are shown in Table 1. The value of ρ is set relatively, by setting the covered area as a percentage of the total area. Similarly, ϵ is set as a percentage of the maximum difference between the observed values.

6.1.3 Evaluation Setting. Each experiment is performed using a randomly selected workload of 100 queries for each dataset and we report the average response time. All indices are held in memory, while the leafs contain pointers to files with geolocated time series stored on disk. All methods were developed in Java. Tests were executed on a server with 4 CPUs, each containing 8 cores clocked at 2.13GHz, and 256 GB RAM running Debian Linux.

6.2 Query Performance

We compare the average per query execution time for all three queries using sweep line and checkpoint methods on BTSR-tree and the checkpoint method on $\mathcal{S}$BTSR-tree.

6.2.1 $Q_{rr}(T_q, \rho, \epsilon, \delta)$. Figure 9 illustrates the query performance for varying thresholds ρ and ϵ and the first column of Figure 10 for varying δ, on all three datasets. It is apparent that the $\mathcal{S}$BTSR-tree with the checkpoint approach outperforms the rest in all cases. Its superior pruning power is attributed to the segmentation, which yields tighter bounds within the nodes and consequently less disk accesses. The sweep line and checkpoint methods over BTSR-tree perform similarly in all cases. Both methods access the same nodes, but the checkpoint approach needs to examine significantly less

[1] https://data.police.uk/data/
[2] https://code.flickr.net/category/geo/
[3] http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

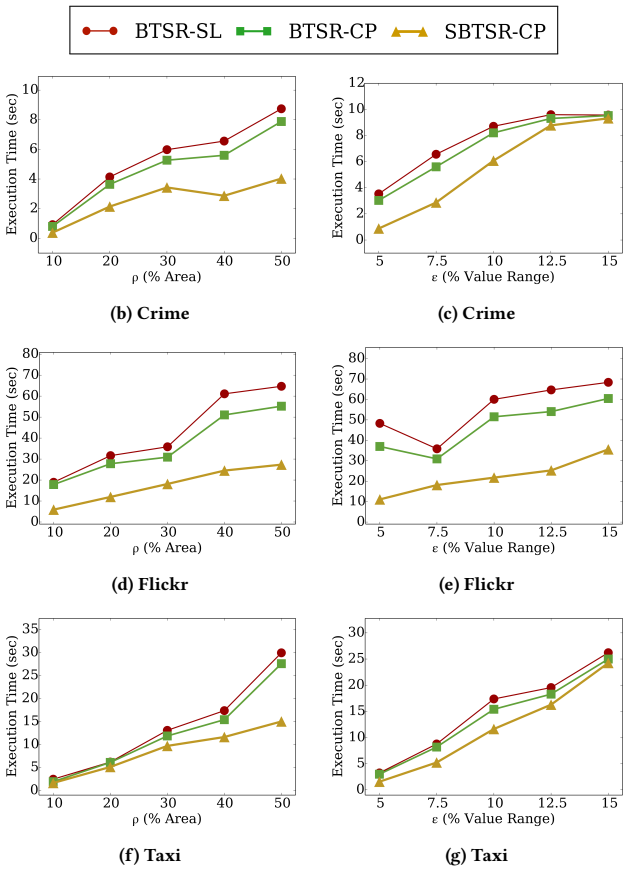


Figure 9: Query $Q_{rr}(T_q, \rho, \epsilon, \delta)$ for varying ρ and ϵ.

values across time to determine local similarities. However, since all local similarity calculations take place in-memory, computation cost does not make a big difference, compared to the less node accesses required with the $\mathcal{S}$BTSR-tree.

More specifically, for the *crime* dataset, relaxing ρ (Figure 9b) has a negative impact on all three methods as more nodes have to be accessed and pruning depends mostly on the ϵ value. $\mathcal{S}$BTSR-tree increasingly outperforms the rest as ρ increases, due to its more aggressive pruning on local similarity. For the case of increasing ϵ (Figure 9c), the result is the opposite, as this way the parameter is relaxed and more nodes get accessed. For very large ϵ values, pruning is solely based on spatial distance and all approaches perform similarly. Finally, increasing δ (Figure 10b) also increases the difference in performance among the three approaches, while it also reduces the average query response time. This is due to large numbers of subsequences qualifying for small δ values, resulting in more node accesses. As δ increases, pruning is more rapidly improved in the case of $\mathcal{S}$BTSR-tree due to its tighter bounds.

The results are similar but with larger differences for the *Flickr* dataset (Figures 9d, 9e and 10f). Intuitively, the less periodicity in a dataset, the more the benefit from segmentation; if the time series in the dataset exhibit periodicity, the bounds that will occur from applying k-means clustering on the whole sequences will be relatively tighter than otherwise. The Flickr dataset, due to its nature, is more random than the crime dataset, which justifies the larger differences. This explanation is also supported by the results for the *taxi* dataset, illustrated in Figures 9b, 9c and 10b. Despite a similar behavior in varying all thresholds, the differences in average query response time among the different approaches are smaller than in the crime and Flickr datasets, due to the high daily periodicity of taxi drop-offs.

Another observation is that the execution cost for queries against the Taxi dataset is lower than that against Flickr. Although these two datasets have a similar number of locations, their spatial distribution and extent differ substantially (Taxi data spans New York city, while Flickr data spans the entire planet), which may significantly affect pruning during search. To verify this, we ran a test with a random Q_{rr} query, $\rho = 30\%$ and the default parameters, and we measured the number of pruned nodes. For the query against the Taxi dataset, 3017 nodes were pruned in the tree as opposed to only 360 nodes in the tree built for the Flickr data. Since spatial filtering is much faster with our approach, this explains the difference in execution cost against these two datasets.

6.2.2 $Q_{kr}(T_q, k, \epsilon, \delta)$. Figures 10c, 10g and 10k depict the results for the $Q_{kr}(T_q, k, \epsilon, \delta)$ query for the three datasets. As k increases, more nodes have to be traversed in order to fetch the additional results, and the execution time increases for all methods. Nevertheless, $\mathcal{S}$BTSR-tree still clearly outperforms the other two algorithms.

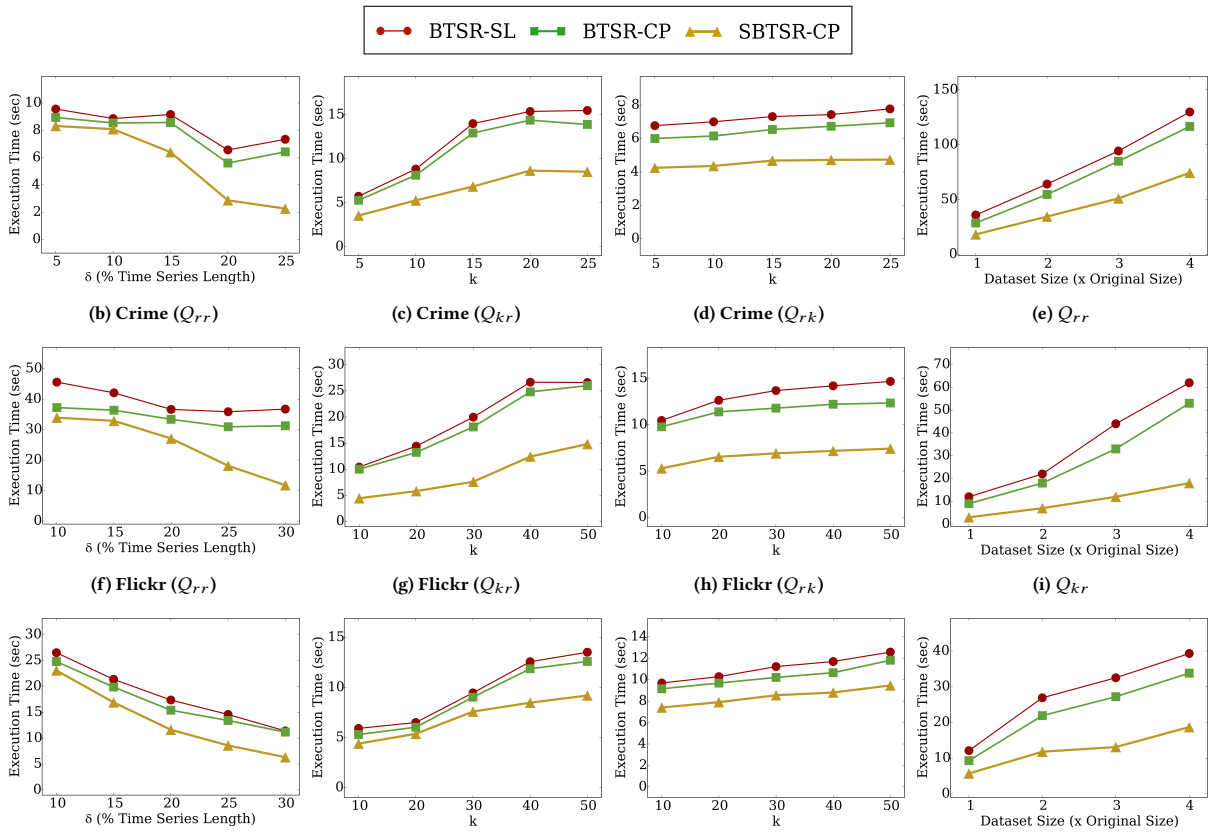
6.2.3 $Q_{rk}(T_q, k, \rho)$. Finally, Figures 10d, 10h and 10l depict the results for the $Q_{rk}(T_q, k, \rho)$ query. In this case, the performance deterioration as k increases is less abrupt, especially for the crime dataset, as usually the top-k results are spatially closely located and are retrieved quickly. Again, the largest and smallest differences are spotted on the Flickr and taxi datasets, respectively.

6.3 Scalability

We performed a scalability evaluation for all three queries using the Flickr-based synthetic datasets, again measuring the average query response time for the same query workload. The results for increasing dataset size (up to four times) are depicted in Figure 10. In all cases, the $\mathcal{S}$BTSR-tree-based approach scales better, especially in the top-k queries (Figures 10i and 10m), where the larger difference observed in Figures 10g and 10h is further augmented.

7 CONCLUSIONS

We have studied three variants of hybrid queries on geolocated time series, involving both range and top-k search, and combining spatial distance with local time series similarity. The latter allows to measure similarity of time series over subsequences instead of their entire length, and thus enables the identification of more fine-grained trends and patterns. The queries are evaluated by hybrid index structures, in order to allow for simultaneous pruning by both criteria. We first discuss query evaluation using the previously proposed BTSR-tree, and then we further extend it to derive the $\mathcal{S}$BTSR-tree which exhibits even better performance, by using temporal segmentation of time series to derive tighter bounds. Our evaluation against several real-world datasets has shown that $\mathcal{S}$BTSR-tree can compute results much faster for all query variants.



(b) Crime (Q_{rr}) **(c) Crime (Q_{kr})** **(d) Crime (Q_{rk})** **(e) Q_{rr}**

(f) Flickr (Q_{rr}) **(g) Flickr (Q_{kr})** **(h) Flickr (Q_{rk})** **(i) Q_{kr}**

(j) Taxi (Q_{rr}) **(k) Taxi (Q_{kr})** **(l) Taxi (Q_{rk})** **(m) Q_{rk}**

Figure 10: Per column: $Q_{rr}(T_q, \rho, \epsilon, \delta)$ for varying δ – $Q_{kr}(T_q, k, \epsilon, \delta)$ for varying k – $Q_{rk}(T_q, k, \rho)$ for varying k – Scalability.

Acknowledgements. This work was partially funded by the EU H2020 projects SLIPO (731581) and SmartDataLake (825041), and the NSRF 2014-2020 project HELIX (5002781).

REFERENCES

[1] Ella Bingham, Aristides Gionis, Niina Haiminen, Heli Hiisilä, Heikki Mannila, and Evimaria Terzi. 2006. Segmentation and dimensionality reduction. In *SIAM*. 372–383.

[2] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn J. Keogh. 2010. iSAX 2.0: Indexing and Mining One Billion Time Series. In *ICDM*. 58–67.

[3] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn J. Keogh. 2014. Beyond one billion time series: indexing and mining very large time series collections with i SAX2+. *Knowl. Inf. Syst.* 39, 1 (2014), 123–151.

[4] Kin-pong Chan and Ada Wai-Chee Fu. 1999. Efficient Time Series Matching by Wavelets. In *ICDE*. 126–133.

[5] Georgios Chatzigeorgakidis, Kostas Patroumpas, Dimitrios Skoutas, Spiros Athanasiou, and Spiros Skiadopoulos. 2018. Scalable hybrid similarity join over geolocated time series. In *SIGSPATIAL*. 119–128.

[6] Georgios Chatzigeorgakidis, Dimitrios Skoutas, Kostas Patroumpas, Spiros Athanasiou, and Spiros Skiadopoulos. 2017. Indexing Geolocated Time Series Data. In *SIGSPATIAL*. 19:1–19:10.

[7] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2018. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. *PVLDB* 12, 2 (2018), 112–127.

[8] Antonin Guttman. 1984. R-trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*. 47–57.

[9] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2018. Coconut: A Scalable Bottom-Up Approach for Building Data Series Indexes. *PVLDB* 11, 6 (2018), 677–690.

[10] Jessica Lin, Eamonn J. Keogh, Li Wei, and Stefano Lonardi. 2007. Experiencing SAX: a novel symbolic representation of time series. *Data Min. Knowl. Discov.* 15, 2 (2007), 107–144.

[11] Michele Linardi and Themis Palpanas. 2018. Scalable, variable-length similarity search in data series: The ULISSE approach. *PVLDB* 11, 13 (2018), 2236–2248.

[12] Michele Linardi, Yan Zhu, Themis Palpanas, and Eamonn J. Keogh. 2018. VALMOD: A Suite for Easy and Exact Detection of Variable Length Motifs in Data Series. In *SIGMOD*. 1757–1760.

[13] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2018. ParIS: The Next Destination for Fast Data Series Indexing and Query Answering. In *IEEE BigData*.

[14] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *SIGKDD*. 262–270.

[15] Nick Roussopoulos, Stephen Kelley, and Frédéic Vincent. 1995. Nearest Neighbor Queries. In *SIGMOD*. 71–79.

[16] Jin Shieh and Eamonn J. Keogh. 2008. *i*SAX: indexing and mining terabyte sized time series. In *SIGKDD*. 623–631.

[17] Djamel-Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. 2018. Massively Distributed Time Series Indexing and Querying. *TKDE (to appear)* (2018).

[18] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. 2016. Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In *ICDM*.

[19] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Zachary Zimmerman, Diego Furtado Silva, Abdullah Mueen, and Eamonn J. Keogh. 2018. Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile. *Data Min. Knowl. Discov.* 32, 1 (2018), 83–123.

[20] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2014. Indexing for interactive exploration of big data series. In *SIGMOD*. 1555–1566.