Mining Correlations on Massive Bursty Time Series Collections

Tomasz Kusmierczyk and Kjetil Nørvåg

Norwegian University of Science and Technology (NTNU), Trondheim, Norway
{tomaszku, noervaag}@idi.ntnu.no

Abstract. Existing methods for finding correlations between bursty time series are limited to collections consisting of a small number of time series. In this paper, we present a novel approach for mining correlation in collections consisting of a large number of time series. In our approach, we use bursts co-occurring in different streams as the measure of their relatedness. By exploiting the pruning properties of our measure we develop new indexing structures and algorithms that allow for efficient mining of related pairs from millions of streams. An experimental study performed on a large time series collection demonstrates the efficiency and scalability of the proposed approach.

1 Introduction

Finding correlations between time series has been an important research area for a long time [10]. Previously, the focus has mostly been on a single or few time series, however recently many new application areas have emerged where there is a need for analyzing a large number of long time series. Examples of domains where there is a need for detecting correlation between time series include financial data, data from road traffic monitoring sensors, smart grid (electricity consumption meters), and web page view counts.

Bursts are intervals of unexpectedly high values in time series and high frequencies of events (page views, edits, clicks etc.) in streams [6]. In contrast to the raw time series, bursts reduce the information to the most interesting by filtering out the low intensity background so that only information about regions with the highest values are kept (i.e., what would be the most visible on plots; see Fig. 1). In this paper, we introduce the problem of identifying streams of bursts that behave similarly, i.e., are correlated. We propose to use bursts as indicators of characteristics shared between streams. If there are many correlated bursts from two streams, it means that these streams are probably correlated too, i.e., respond to the same underlying events. However, different streams may vary in sensitivity, may be delayed or there might be problems in the bursts extraction process. As an example, consider Fig. 1 that shows two streams representing page views of two related Wikipedia articles; the first representing the TV show *The Big Bang Theory*, the second one represents one of the main characters from the show. Although the plots differ in intensity and bursts vary in heights, bursts locations match. Consequently, a standard way of correlating bursts is overlap relation, overlap operator or overlap measure [11, 12]. In that sense, bursts are assumed to be related if they overlap and the measure of stream similarity proposed in this paper is based on this relation.

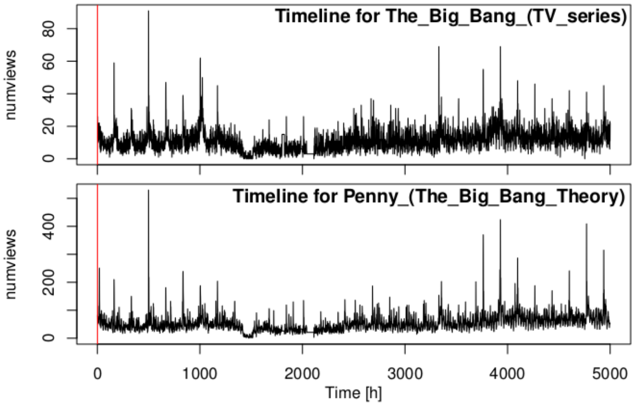

Fig. 1: Comparison of page views of two related Wikipedia articles.

The main challenge of correlation analysis of many streams is the computational complexity of all-pairs comparison. Approaches proposed for general time series are not appropriate for streams of bursts, since binary intervals have different nature than real-value, continuous signals. Therefore, we propose novel indexing structures and algorithms devoted particularly to efficient mining of correlated bursty streams. The core of the approach is a Jaccard-like measure based on the overlap relation and using its pruning properties to limit the number of pairs of streams that must be compared. Additionally, the bursts are indexed in a hybrid index that provides for efficient matching of streams, further reducing the computation cost of the correlation analysis.

We provide an extensive evaluation of our approach where we study the effect of different similarity thresholds and collection sizes. Although our approach is generic and can be applied in any domain, in this paper we perform the experimental evaluation on burst streams extracted from time series representing the number of page views per hour for Wikipedia articles. This collection contains a large number of time series and is freely available, thus facilitating reproducibility of our experiments.

To summarize, the main contributions of the paper include:

- A measure and framework for correlating streams using bursts.
- Indexes and algorithms for efficient mining of correlated bursty streams.
- An experimental evaluation on a large collection of time series studying the efficiency and scalability of the approach.

The rest of this paper is organized as follows. Sect. 2 gives an overview of related work. In Sect. 3 we formulate the main task and introduce needed notions. In Sect. 4 we discuss issues related to measures of similarity. Sect. 5 describes the indexes and algorithms used in our approach. Our experimental results are presented in Sect. 6. Finally, in Sect. 7 we conclude the paper.

2 Related Work

There is a large amount of related work on time series and correlation. For example, Gehrke et al. [5] designed a single-pass algorithm that approximately calculates correlated aggregates of two or more streams over both landmark and sliding windows. In [4] correlations were used for measuring semantic relatedness in search engine queries. In [15] it is studied how to select leading series in context of lagged correlations in sliding windows. However, the main challenge of correlation analysis is computational complexity of all-pairs comparison. Zhu and Shasha [17] addressed the problem of monitoring thousands of data streams. Mueen et al. [9] considered computing correlations between all-pairs over tens of thousands of time series. In both papers, they used the largest coefficients of the *discrete Fourier transform* as the approximate signal representation. In other works, researchers were mining and correlating large numbers of time series (millions of streams) using symbolic representation [2, 3]. However, binary intervals (bursts) cannot be effectively compressed (not loosing much information and without introducing artifacts) and indexed in that way. Similarly, we rejected wavelets as not matching bursts characteristics and efficient processing requirements.

Our task has similarities to clustering (a survey on clustering of related time series can be found in [8]). Those techniques are either not scalable, require embedding into Euclidean space, or provide only approximate results. Related works can be also found in the area of mining correlated sets (e.g. [1]) and also *sequential patterns*. However, bursts (with overlap relation) cannot be translated into set elements or items without additional assumptions or introducing transformation artifacts.

There are several papers exploiting bursts in a way similar to ours. Vlachos et al. [12] focused on indexing and matching single bursts. This task is different from our since bursts are considered independently within streams. Vlachos et al. adapted *containment encoded intervals* [16]. Although the proposed index is efficient in answering queries composed of sets of bursts, is not able to handle whole streams. An interesting approach to discover correlated bursty patterns containing bursts from different streams, can be found in [13, 14]. The basic idea is to introduce a latent cause variable that models underlying events. A similar approach was applied in [7] where they used *hidden Markov models* with *Poisson* emission distributions instead. However, all these approaches are based on matching probabilistic models and are not scalable, the authors assume not more than some hundreds of streams.

3 Preliminaries

We assume a set of N raw streams of basic events (or time series). Time intervals of untypically high frequencies (or values) are called *bursty intervals* (bursts).

Definition 1. ***Burst*** b *is time interval* $[start(b), end(b)]$ *of high frequency of basic events, where* $start(b)$ *denotes starting time point of the burst and* $end(b)$ *stands for ending time point.*

As mentioned above, we do not consider bursts height or width but only the fact of occurrence. The bursts are extracted from the original streams either in on-line or offline manner, for example using the algorithm of Kleinberg [6]. Similar to [11, 12], for the purpose of measuring similarity we use overlapping bursts. We define the overlap relation as follows:

Definition 2. ***Overlap relation*** *between two bursts b and b': $b \circ b' \iff (start(b) \leq start(b') \wedge end(b) \geq start(b')) \vee (start(b') \leq start(b) \wedge end(b') \geq start(b))$. The overlap relation is reflexive and symmetric but not transitive.*

The burst extraction process results in a set of bursty streams: $D = \{E^1, E^2, ..., E^N\}$ where $N = |D|$.

Definition 3. ***Streams*** *of bursty intervals are defined as $E^i = (b^i_1, b^i_2, ...)$ where $b^i_j \circ b^i_k \iff j = k$ and $start(b^i_j) > start(b^i_k) \iff j > k$.*

We define overlapping between streams as follows:

Definition 4. ***Set of bursts*** *of E^i* ***overlapping*** *with E^j: $O^i_j = \{b : b \in E^i \wedge \exists_{b' \in E^j} b \circ b'\}$.* ***Set of non-overlapping bursts*** *of E^i when compared to E^j: $N^i_j = E^i \setminus O^i_j$. We denote $e^i = |E^i|$, $o^i_j = |O^i_j|$, $n^i_j = |N^i_j|$.*

The main problem we address in this paper is *how to efficiently mine interesting pairs of bursty streams.*

Definition 5. ***Interesting correlated streams*** *are defined as pairs of streams, which for some measure of similarity J have similarity no smaller than a threshold J_T.*

Definition 6. *We define a set S^n containing all* ***streams with exactly*** *n* ***bursts****: $S^n = \{E^i : E^i \in D \wedge e^i = n\}$.*

4 Correlation of Bursty Streams

We aim at mining streams having correlated (overlapping) bursts. Neither set oriented measures such as the *Jaccard index*, contingency measures such as the *phi coefficient* nor general time series measures such as *dynamic time-warping* or *longest common subsequence* are directly applicable. Because of overlap relation properties, streams of bursts cannot be mapped to sets. Bursts could be grouped in equivalence classes according to underlying events, but such mapping is not available. Also, interpreting intervals (bursts) as continuous, real signals implies the need of adjustments which at the end introduce undesirable effects. For example, scaling time to maximize stream matching violates the assumption about simultaneous occurrence of related bursts. Consequently, we decided to focus on the overlap relation e.g., our measure should rely on o^j_i,o^i_j,n^j_i, and n^i_j. Below, we will a discuss possible solution and desired properties.

We are interested in measures that can be efficiently tested against many pairs of streams. We would like to be able to prune pairs that have similarity below some threshold in advance. For that purpose, we introduce pruning property.

Definition 7. *Similarity measure s has* ***pruning property*** *if $s \geq s_T \implies |e^i - e^j| \leq f(e^i, e^j, s_T)$ where s_T is some constant value of similarity (threshold) and f is some function.*

For measures having this property, the number of pairs of streams to be considered can be reduced as only pairs having similar (difference is limited by f) number of bursts can be scored above the threshold s_T. Naturally, we would like f to be as small as possible. In practice, streams have limited burst counts. Then, only f obtained values below that limit are interesting and allow for pruning.

We adapted one of the measures used widely in areas such as information retrieval and data mining, the *Jaccard index*. Unfortunately, bursts are are objects with properties different from set elements. In the general case there is no simple mapping, so the measure needs to be defined for the new domain as shown below. If there is a one-to-one assignment between overlapping elements from both streams, our measure reduces to standard Jaccard index. In that case, overlapping pairs of bursts from both streams are treated as common elements of sets. This situation is obviously a special case, in general one interval can overlap with two or more intervals. Because of that, our measure also does not preserve triangle inequality.

Definition 8. *For the purpose of measuring similarity of bursty streams we define an adapted* ***Jaccard index*** *as:*

$$J(E^i, E^j) = \frac{\min(o_i^j, o_j^i)}{e^j + e^i - \min(o_i^j, o_j^i)} \in [0, 1]$$

Lemma 1. *J* ***has pruning property*** *with $f(e^i, e^j, J_T) = \max(e^i, e^j) - \lceil \max(e^i, e^j) \cdot J_T \rceil$*

The maximum value of J for a pair of two streams E^i, E^j is obtained when $\min(o_i^j, o_j^i) = \min(e^j, e^i)$. Then the measure reduces to: $J_{max} = \frac{\min(e^j, e^i)}{\max(e^j, e^i)} \rightarrow J \leq \frac{\min(e^j, e^i)}{\max(e^j, e^i)}$. Without loss of generality, we assume that $e^i \geq e^j$. This implies $J \leq \frac{e^j}{e^i}$ and for fixed J_T: $J_T \cdot e^i \leq e^j \leq e^i$. Consequently, to obtain related pairs, sets S^n, need to be compared only with streams in S^m where $m \in [\lceil n \cdot J_T \rceil, n]$.

Definition 9. *We define* ***connected counts*** *$connected(n)$ as the set of such values m for some burst count n that $m \in [n - f(n, m, s_T), n]$. We denote n as the* ***base count****.*

5 Indexing and Mining

Mining related pairs is a computationally expensive task. The cost of similarity measure calculation for a single pair E^i, E^j is $O(\max(e^i, e^j))$. For all pairs, it sums up to $O(|D|^2 e)$, where e stands for average number of bursts per stream. However, thanks to the pruning property of the measure, we do not need to consider all pairs: we can prune those differing much in number of bursts. Unfortunately, what is left can be also expensive to compute. Therefore further optimizations are needed.

A high level description of our approach is presented on Fig. 2. The workflow can be used to describe both an offline setting where source streams are stored in some local storage, and an on-line setting where streams of events are produced in a continuous manner. In the latter case we assume that the burst detection algorithm is applied immediately, and therefore the amount of data is significantly reduced. Although initially

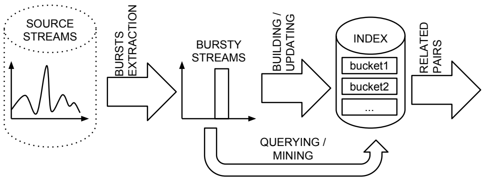

Fig. 2: Overview of data workflow.

there can be millions of raw streams, in real-life contexts we expect no more than some of tens of bursty intervals per stream. Each interval can be stored using only two numbers (start and end time point), and then we are able to store in main memory all bursts and all streams from the interesting period of time.

Bursty streams are indexed in a high level index composed of buckets. Buckets are responsible for keeping track of subsets of streams (or pairs of streams) and can contain lower-level indexes. The partitioning into buckets is based on number of bursts per stream. Mining of correlated pairs is done by comparing buckets between themselves and/or against stored streams. However, details vary according to approaches described below.

5.1 Naive Approach

As baseline, which we call *Naive*, we compare all pairs that are not pruned. As described above, streams are partitioned to buckets according to number of bursts. I.e., set S^n is placed in bucket n, and there is no further lower-level indexes within the buckets. To obtain all related pairs, for each base count n we simply check all connected counts m and compute the correlation J between all possible pairs of streams from $S^n \times S^m$ (n-th bucket vs. each of m-th buckets).

5.2 Interval Boxes Index

The naive baseline can be improved by speeding up the matching within buckets. Each bucket n is assigned a lower-level index responsible for keeping track of S^n. During the mining, the bucket n index is queried with all streams having connected counts m, i.e., queried with streams from $\bigcup_{m \in connected(n)} S^m$. One approach that could be used is to apply some of the existing indexes designed for efficient retrieval of (single) overlapping bursts, e.g., *containment encoded intervals* [16] for selection of candidates. Then candidate streams returned by the index, already having at least one overlapping burst with the query stream, are validated using the similarity measure and those under the threshold are rejected. Unfortunately this does not scale well, because candidate sets increase proportionally to number of streams in data set. In order to overcome this, we propose to consider k-element, ordered subsets of bursts. Each k-subset is placed in a k-dimensional space as shown in Fig. 3. The first burst from the subset determines interval in the first dimension, second burst determines interval in the second dimension, and so on. A k-element subset determines a k-dimensional box. For example, in Fig. 3,

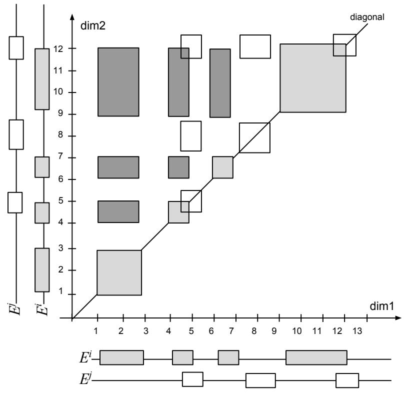

Fig. 3: Example of *IB* index: indexed stream E^i and querying stream E^j $(k = 2)$.

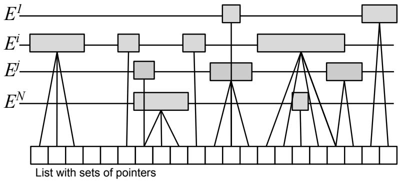

Fig. 4: Structure of a single *LS* index.

we have all possible 2-dimensional boxes representing all possible ordered 2-subsets of streams E^i and E^j.

The idea of the *Interval Boxes (IB)* index is to have a k-dimensional box assigned to stream E^i in the index, that will match (overlap) some k-dimensional box assigned to E^j that the index will be queried with. Assuming that E^j, E^i have similarity above some similarity threshold J_T, there are o_i^j bursts of E^j overlapping with o_j^i bursts of E^i. Consequently, there are two boxes: μ-dimensional, where $\mu = \min(o_i^j, o_j^i)$, of E^j and μ-dimensional of E^j that overlap. What is more, all k-dimensional projections of these boxes also overlap. Now instead of single burst, we are matching k bursts at once. Because we are matching k bursts at the same time, the probability of spurious candidates (that will be later rejected as having less than the similarity threshold) is low. The higher k we choose, the lower that probability and the additional cost of validating.

In practice, boxes can be stored in any spatial index that supports overlap queries. In our approach R-trees were used. Because bursts are ordered, only the ”top” half of the k-dimensional space is filled. Ordering of bursts is important because of complexity issues. It significantly reduces number of subsets to be considered and inserted into the index. However, the guarantee that no potentially matching streams will be missed still holds. If some burst b from stream E^i overlaps with some b' from stream E^j it means that those later than b cannot overlap with these being earlier than b'.

One should also notice boxes on the diagonal. Pure subsets (without replacement) do not cover situations where one burst overlaps with many. In such cases, some burst must be repeated in several consecutive dimensions (bursts are ordered). For $k = 2$ each burst can be used once, twice or not used at all in the box. For $k = 3$ each burst can be repeated once, twice, three times or not at all as long as no more than 3 dimensions are used in total. Higher k implies many more combinations to be considered. This can significantly increase the number of boxes and decrease the efficiency. On the other hand multiple overlapping is not very probable. To prevent inserting and querying indexes with unnecessarily many boxes, we introduced an additional parameter ρ that

Algorithm 1 Generation of k-dimensional boxes

```
1: function BOXES($E$, $k$, $r$)
2:     $C \leftarrow$ COMBINATIONS($E$, $k$)
3:     $C' \leftarrow \emptyset$
4:     for all $r' = 1..\min(r, \rho - 1)$ do
5:         $k' \leftarrow k - r'$
6:         for all $c' \in$ COMBINATIONS($E$, $k'$) do
7:             for all $I \in$ COMBINATIONSREP($1..|c'|$, $r'$) do
8:                 $c' \leftarrow c'$ with bursts of indices $I$ repeated
9:                 $C' \leftarrow C' \cup c'$
10:    return $C \cup C'$
```

COMBINATIONS(S, k) - k-element ordered combinations of S
COMBINATIONSREP(S, k) - combinations with replacement

limits how many times, i.e, in how many dimensions, each burst can be repeated. As a result, some pairs, i.e. relying on multiple overlaps between bursts, may be missing but the mining speed increases significantly.

Mining correlated pairs of streams is done by querying all the indexes with streams having connected counts. Index in bucket n, where n is the base count, is queried with all streams from S^m, for all possible connected counts m. The index itself is queried with all possible k-dimensional boxes generated from query stream. For each query box all overlapping boxes from the index are retrieved. For each of them candidate pair (query stream and matching stream from the index) is extracted. In the final step, candidate pairs are validated against similarity measure J and only these having no less than threshold value J_T are kept.

Algorithm 1 shows how k-dimensional boxes are generated. It is composed of two parts. In line 2, boxes without repetitions are generated. In lines, 3-9 boxes having up to r dimensions (given as the parameter) being repetitions of previous dimensions are computed (recall that bursts and dimensions are ordered). An important line is 4, where we additionally limit number of dimensions being repetitions with ρ. For $\rho = 1$, the set of combinations with repetitions C' remains empty.

The number of k-subsets (and consequently k-boxes) without replacement ($\rho = 1$) of some stream E^i is equal to $\binom{e^i}{k}$. For higher values of ρ it is even more. To keep the number of generated boxes (both in the index and in queries) relatively small, k should be either very small or close to e^i. Consequently, we introduce two types of indexes: *IBLD* (low dimensional, for small k-s) and *IBHD* (high dimensional, for big k-s) that have very different properties.

IBLD Index *IBLD* (low dimensional) indexes are ordinary k-dimensional (e.g., $k = 2$) R-trees. The dimensionality k is constant for indexes in all buckets. Consequently, insertion, deletion and querying require generation of all possible k-dimensional boxes.

What is important, *IBLD* cannot be used for streams having very small number of bursts (e.g. $\sim k$) and for very low values of threshold. The index does not work when the similarity for output pair (by output pair we mean the pair that has similarity above

Algorithm 2 Querying the *IBHD* index

```
function QUERY($E^j$, $J_T$)
    $n \leftarrow Index.baselevel$
    $k \leftarrow Index.dimensionality$
    $m \leftarrow |E^j|$                                  ▷ We assume $\lceil n \cdot J'_T \rceil \leq m \leq n$
    $r \leftarrow m - \lceil J_T \cdot (n+m)/(1+J_T) \rceil$
    CANDIDATES $\leftarrow \emptyset$
    for all $B \in$ BOXES($E^j, k, r$) do
        MATCHING $\leftarrow Index$.getOverlapping($B$)
        STREAMS $\leftarrow B'.stream$ for each $B' \in$ MATCHING
        CANDIDATES $\leftarrow$ CANDIDATES $\cup$ STREAMS
    OUTPUT $\leftarrow \emptyset$          ▷ Subset of interesting pairs with $E^j$ on the first position
    for all $E^i \in$ CANDIDATES do
        if $J(E^j, E^i) \geq J_T$ then
            OUTPUT $\leftarrow$ OUTPUT $\cup$ $(E^j, E^i)$                ▷ Output pair found
    return OUTPUT
```

the threshold) of streams is obtained for number of overlapping bursts (measured with o^i_j, o^j_i) lower than k.

IBHD Index *IBHD* (high dimensional) indexes require k to be as high as possible in order to reduce overall size. On the other hand, if $k > \mu$ we can miss matching between some pairs. From that we imply $k = \mu$. Unfortunately, μ depends on the measure threshold. Consequently, *IBHD* indexes are built for some threshold J'_T and cannot be used for finding pairs of similarity below this threshold. For index built for J'_T, only values $J_T \geq J'_T$ can be used. The border situation is when all bursts of stream having $m' = \lceil J'_T \cdot n \rceil$ (the lowest number of bursts that stream must have to be compared to streams having n bursts) overlap with bursts of stream having n bursts. It means $k = \lceil J'_T \cdot n \rceil$. To match streams having m bursts for $m > m'$, higher values of k would be better. Unfortunately, this would introduce additional costs both in computations and space needed. Therefore, we prefer to have a single index for whole range of connected counts $m \in [m', n]$ and for each n we choose $k = m'$. This value is the highest possible, guaranteeing not missing any pairs (holds when $\rho = \infty$; for $\rho < \infty$ some pairs may be missing but not because of selection of k and due to some boxes being skipped).

As mentioned earlier, one of the major issues influencing speed is the possibility of single burst overlapping with many. Fortunately, in *IBHD* indexes, only a limited number of dimensions needs to be covered with repeated (copied) bursts. For example, if $n = 10$ and $J_T = 0.7$, the lowest $m = 7$. It means that $min(o^j_i, o^i_j) = 7$ and at least 7 out of 10 bursts must be different. Only 3 dimensions can be repeated. In general, for base count n: $r = n - \lceil J_T \cdot n \rceil$. For connected counts m situation is slightly different (as $m \leq n$) and $r = m - \lceil J_T(n+m)/(1+J_T) \rceil$.

Algorithm 2 shows how above ideas can be implemented in practice. The index keeps track of streams having n bursts and can handle queries of streams having $m \in [m', n]$ bursts. For input stream E^j all possible k-dimensional boxes are generated. Each of these boxes is used to query internal index (e.g., R-Tree). The internal index returns boxes overlapping with query boxes. For each returned box relevant stream identifier is extracted. Then, streams are appended to the candidates set. Finally, all generated candidate streams E^i are compared to E^j using similarity measure J. If the value is above the threshold J_T the pair (E^j, E^i) is included in the result set.

5.3 List-based Index

For the *IB* approach, in bucket n we store the index responsible for keeping track of streams from S^n. An alternative is to use buckets indexed with two values: base count n and some connected count m. Each bucket contains a single *List-based* (*LS*) index covering both S^n and S^m. The number of connected counts to be considered depends on the predetermined threshold J'_T. Consequently, such an architecture is able to support mining for thresholds $J_T \geq J'_T$.

The structure of a single *LS* index is shown in Fig. 4. For *LS* we use the notion of discrete time where the timeline is divided with some granularity (e.g., hour, day, week; depending on data characteristics) into time windows. In such a setting bursts must be adjusted to window borders. For each window we store the information about all bursts overlapping with it. Consequently, the index is a list of sets of references pointing at bursts where a single set represents single time window.

For the *Naive* and *IB* approaches, mining was done by querying proper indexes with all of the input streams. Using *LS* index it is done directly. Algorithm 3 presents how values of o^i_j,o^j_i are computed. The main loop (lines 5-21) iterates over all time windows (sets of references) indexed with t. In each step (for each t), four sets are computed: *ACTIVE*, *NEW*, *OLD*, and *END*. *ACTIVE* is a set of bursts active in the current time window (taken directly from the index). *NEW* is a set of bursts that were not active in the previous $(t-1)$-th time window but are active in the current one, *OLD* contains those active both in the current and the previous window and *END* those active in the previous but not in the current one. A map (dictionary) *OVERLAPS* contains sets of streams overlapping with bursts active in the current time window. Keys are bursts and values are sets of streams. Maintenance is done in lines 11-12. When a burst ends, it is removed from the map. Pairs of overlapping bursts that were not seen previously (in the previous step of the main loop) are those from the set *NEW* × *NEW* ∪ *OLD* × *NEW*. For each of these pairs the map *OVERLAPS* and the map o are updated in lines 16-21. Using the map o, candidate pairs of streams can be validated against the threshold J_T. The final step of the mining (lines 22-26) is then validation of all pairs included in the map o. Only pairs having at least one pair of bursts overlapping are considered (included in the o). What is more, each pair is validated in constant time as o^i_j,o^j_i (and e^i, e^j) are already known.

The biggest disadvantage of the *LS* index is its memory use if there are many streams bursty in a particular index time window (bursts from many streams overlapping at once), i.e., when there are particularly many references in some list set. A solution to this problem is sharding, i.e., we use several indexes per each high-level bucket n, m. Each index covers only a subset of possible pairs of streams. Division of the pairs can be done in any way. Function QUERY guarantees that any pair of bursts, and any pair of streams having bursts overlapping, will be considered. We only need to make sure that

Algorithm 3 Candidates generation and validation in *LS* index

```
function QUERY($J_T$)
    $o \leftarrow \emptyset$    ▷ dictionary $\{(i,j) \rightarrow$ current value of $o_i^j$ $\}$
    $PREV \leftarrow \emptyset$    ▷ empty set of bursts
    $OVERLAPS \leftarrow \emptyset$    ▷ dictionary $\{$burst $\rightarrow$ set of overlapping streams$\}$
    for all $t = 1...Index.length$ do    ▷ Iterate over consecutive windows
        $ACTIVE \leftarrow Index.interval[t]$    ▷ Bursts in $t$
        $NEW \leftarrow ACTIVE \backslash PREV$    ▷ New bursts
        $OLD \leftarrow ACTIVE \backslash NEW$    ▷ Old bursts
        $END \leftarrow PREV \backslash ACTIVE$    ▷ Ending bursts
        $PREV \leftarrow ACTIVE$

        for all $b \in END$ do
            delete $OVERLAPS[b]$

        for all $b, b' \in NEW \times NEW \cup OLD \times NEW$ do
            $i \leftarrow b.streamindex$
            $j \leftarrow b'.streamindex$
            if $j \notin OVERLAPS[b]$ then
                $OVERLAPS[b] \leftarrow OVERLAPS[b] \cup \{j\}$
                $o_j^i = o_j^i + 1$    ▷ Increase count
            if $i \notin OVERLAPS[b']$ then
                $OVERLAPS[b'] \leftarrow OVERLAPS[b'] \cup \{i\}$
                $o_i^j = o_i^j + 1$    ▷ Increase count
    $OUTPUT \leftarrow \emptyset$    ▷ Subset of interesting pairs
    for all $(i,j) \in o$ do
        if $\frac{\min(o_j^i, o_i^j)}{e^i + e^j - \min(o_j^i, o_i^j)} \geq J_T$ then
            $OUTPUT \leftarrow OUTPUT \cup (E^j, E^i)$    ▷ Output pair found
    return $OUTPUT$
```

all possible pairs of streams (that are not pruned) are at some point placed together in the same *LS* index.

5.4 Hybrid Index

The *IB* and *LS* indexes have different properties and are appropriate for data of different characteristics. *IB* efficiency depends mostly on the number of k-dimensional boxes in the index. This increases fast with number of bursts per stream and when threshold J'_T is decreasing (this applies only to *IBHD*). On the other hand, *LS* efficiency does not depend directly on either number of bursts per stream or J'_T. This two factors influence only number of buckets to be considered, but not *LS* processing time. What affects mostly the efficiency here is the size of sets of references. The bigger sets are, the more pairs need to be considered at each step.

Consequently, we propose the *Hybrid* index that exploits good properties of both approaches. It uses *IBHD* for low base counts (and proper connected counts) and *LS* for high base counts. Switching count value depends mostly on data characteristics but some observations can be made. Number of boxes generated for *IBHD* index for each base count e depends mostly on index dimensionality k. Assuming $\rho = 1$, the number of boxes generated per stream can be approximated as $\sim e^{e-k}$. This can be seen from the number of distinct k-element subsets $\binom{e}{k} = \frac{e!}{(e-k)!k!}$, when k is close to e, then $(e-k)!$ is small and $\binom{e}{k} \sim e \cdot (e-1) \cdot .. \cdot (k+1) \sim \prod_{l=1..(e-k)} e \sim e^{e-k}$. The exponent $e - k$ changes stepwise. For example for $J'_T = 0.95$ it changes for $e = 20, 40, 60, ...$, and in ranges $e \in (0, 20)$, $[20, 40)$, $[40, 60)$, ... the efficiency of *IBHD* is more or less constant. Consequently, it is enough to check only one switching count per range, starting from lower values toward higher ones.

5.5 On-line Maintenance

The described approaches can be used both in the offline and on-line case. For offline mining, where we have information about number of bursts per stream available in advance, construction of the indexes is performed by assigning each stream to the proper bucket (or buckets for LS) and then by inserting it into the index (or indexes) within that bucket.

In the on-line case, bursts arrive and number of bursts per stream changes in a continuous way. When a new burst is extracted, the relevant stream is moved from the previous bucket (or buckets) to a new one matching the new number of bursts. In addition, indexes need to be updated accordingly. Although this introduces additional maintenance cost these operations are distributed over time. In contrast to offline mining, where the whole dataset must be processed at once, for the online case the cost is dispersed among burst arrival moments.

For the *IB* index, deletion and insertion require generation of all k-dimensional boxes. First, all boxes generated for the old version of the stream (without the new burst) are deleted from the proper spatial index. Then, all boxes generated for the stream with the new burst included are inserted into the index responsible for keeping track of streams with higher number of bursts.

For the *LS* index, insertion and removal are performed by adding and deleting relevant burst references. This is done over all indexes matching the stream burst count (recall that each stream can be indexed in several buckets). First, all sets of references assigned to time windows overlapping with old bursts are updated by deletion of proper references. Then, references to all bursts including the new one are inserted to sets into indexes matching the new number of bursts.

6 Experimental Evaluation

In this section, we present the results of the experimental evaluation. All our experiments were carried out on a machine with two Intel Xeon X5650 2.67GHz processors and 128GB RAM. However, experiments were limited to using one thread and 24GB of main memory.

6.1 Experimental Setup

Dataset. For the experimental evaluation we used a dataset based on English Wikipedia page view statistics[1] ($\sim 4.5M$ articles) from the years 2011-2013. These statistics have a granularity equal to 1 hour, so that the time series for each page covers 26304 time points. Bursts extraction was performed using Kleinberg's off-line algorithm [6]. In post-processing, we reduced the hierarchical bursts that are produced by the algorithm to a flat list of non-overlapping bursts, and we also filtered out bursts that had average height (measured in original page views) lower than twice the average background level. After applying this bursts extraction procedure the dataset had $\sim 47M$ bursts in total.

Key features that influence efficiency of our approach are number of streams, number of bursts per stream, and length and distribution of bursts. Fig. 5 shows the distribution of number of bursts per article (steams) in our dataset. One can see that pages with low number of bursts dominate and that number of streams decreases fast when number of bursts per stream increases (notice the use of logarithmic scale). Streams having a low number of bursts have a higher probability of spurious correlations, therefore we filtered out streams having less than 5 bursts. After that, we are left with $\sim 2.1M$ streams, having in total $\sim 43M$ bursts.

Fig. 6 presents the distribution of length of bursts in the dataset after filtering. In the dataset short bursts dominate. The average length is equal to 28 hours but median is only 10 hours. Nevertheless, one should notice there is non-negligible number of long bursts (100-1000 hours) that significantly increases the sizes of candidate sets and computation time for indexes.

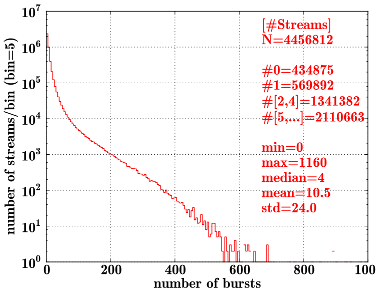

Fig. 5: Number of bursts per stream.

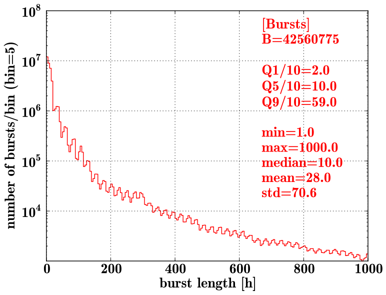

Fig. 6: Length of bursts.

Algorithms and measurements. In the experiments, we studied the *Naive*, *LS* , 2-dimensional *IBLD* (denoted as *IBLD2*), and *Hybrid* approaches. For *LS* the number of streams in bucket processed at once was limited to $50k$, and for the *Hybrid* approach *IBHD* was used for *burst count* < 40 and *LS* was used for *burst count* ≥ 40). In the experiments we set $\rho = 1$ and $J_T = J'_T$. Mining time is the actual run time for mining, i.e., excluding index building time (which was measured separately).

[1] https://dumps.wikimedia.org/other/pagecounts-raw/

6.2 Experimental Results

Fig. 7 presents the time for querying streams for each base count, i.e., for each stream having a particular base count (and stored in the index), the streams having related counts (wrt. to the particular base count) are used to query the stored streams. The efficiency of both the *Naive* and *LS* approaches increases with increasing burst count. This is caused by a decreasing number of streams having high number of bursts. *IBLD2* behaves worse than the *Naive* and *LS* approaches for almost any base count. The reason is that the cost of matching of all possible 2-dimensional boxes dominates the benefit of reduced number of pairs to be validated. Notice that the *IBHD* index approach has a stepped plot. Whenever the difference between index dimensionality and base count increases, the computation time also increases (about 10 times). The observations in this figure also gives a justification for the *Hybrid* approach, where *IBHD* is used for low burst counts and then *LS* for the higher ones. It also shows the threshold for switching strategy, i.e., with burst count of 40.

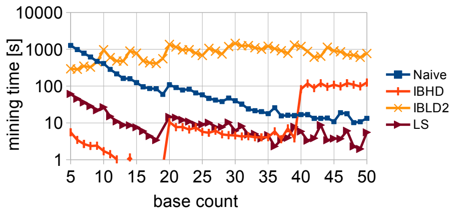

Fig. 7: Querying efficiency for different base counts ($N = 100k$, $J'_T = 0.95$).

Fig. 8 shows the cost for mining correlations for random subsets of varying cardinalities (the cost of *IBHD* is not shown since even for small datasets the number of boxes per stream can be extremely high). As can be seen, *IBLD2* is not scalable and even behaves worse than *Naive* in all cases. In contrast, both the *LS* and *Hybrid* approaches scale well. However, for large volumes *Hybrid* outperform all the other approaches, as it combined the benefits of *IBHD* and *LS*. Fig. 9 shows the cost of building the indexes. As can be seen, this cost is insignificant (less than 10%) compared to the cost of the correlation mining.

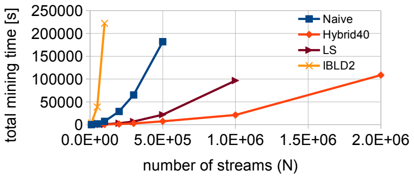

Fig. 8: Index mining time ($J'_T = 0.95$).

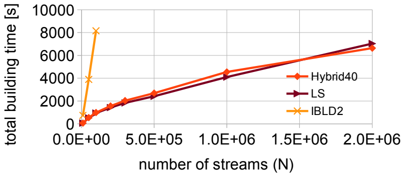

Fig. 9: Index building time ($J'_T = 0.95$).

Fig. 10 illustrates how the number of generated pairs increases with the size of the dataset. The output size increases quadratically with input size, up to $\sim 82k$ for the whole dataset.

The *Naive* and *LS* approaches guarantee generation of all pairs above the selected threshold. This does not hold for *Hybrid*. However, in our experiments the number of missing output pairs was small e.g., for $N = 500k$ streams it was always less than 10%. Fig. 11 shows the influence of $\rho = 1$ on $N = 100k$ streams and for different thresholds. We can observe that even for low thresholds, e.g., $J'_T = 0.8$, the number of missing pairs is smaller than 15%. Furthermore, for higher thresholds the matching streams must have smaller differences in number of bursts and therefore the influence of ρ decreases.

Fig. 12 presents the behavior of the *Hybrid* approach for different thresholds J'_T. With higher threshold, the cost of mining reduces. There are two reasons for that. First, the number of counts connected to each base count is expressed by this value. Second, the *IBHD* dimensionality is also related to it. Consequently, for lower J'_T more pairs need to be considered and using lower dimension indexes.

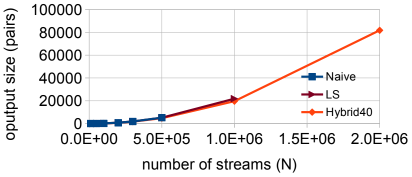

Fig. 10: Number of generated pairs ($J'_T = 0.95$).

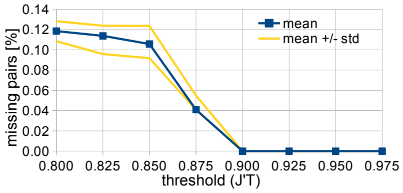

Fig. 11: Number of missing pairs for different thresholds ($N = 100k$, $\rho = 1$, *Hybrid*).

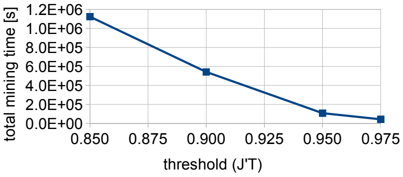

Fig. 12: *Hybrid* mining time ($N \sim 2.1M$).

As shown above, the *Hybrid* approach is scalable wrt. computation time. Regarding memory requirements, the indexes in this approach fit in main memory. The reason is that the *LS* index size can be easily expressed by the number of pointers plus some additional space for the main list. The number of pointers is equal to *total number of bursts* times *average burst length*. *IBHD* uses spatial index and memory usage depends mostly on the size of that index, and is proportional to *number of boxes inserted* times *insertion cost*. For the dataset used in this evaluation, the size of the indexes is in the order of a few GB.

7 Conclusions

With emerging applications creating very large numbers of streams that needs to be analyzed, there is a need for scalable techniques for mining correlations. In this paper, we have presented a novel approach based on using bursts co-occurring in different streams for the measurement of their relatedness. An important contribution of our approach is the introduction and study of a new Jaccard-like measure for correlation between streams of bursts, and exploiting the pruning properties of this measure for reducing the cost of correlation mining. Combined with a new hybrid indexing approach, the result is an approach that allows for efficient mining of related pairs from millions of streams.

In the future we plan to work further on improvements of our algorithms. One interesting issue is border cases that not necessarily follow the assumptions of the algorithms, i.e., when there is many streams bursting often and almost always in overlapping time moments. We also would like to investigate how to reduce the cost of online index maintenance.

References

1. F. Alvanaki and S. Michel. Tracking set correlations at large scale. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, 2014.
2. A. Camerra, T. Palpanas, J. Shieh, and E. J. Keogh. iSAX 2.0: Indexing and mining one billion time series. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, 2010.
3. A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. J. Keogh. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *Knowl. Inf. Syst.*, 39(1):123–151, 2014.
4. S. Chien and N. Immorlica. Semantic similarity between search engine queries using temporal correlation. In *Proceedings of the 14th International Conference on World Wide Web*, 2005.
5. J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. *SIGMOD Rec.*, 30(2):13–24, 2001.
6. J. Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.
7. A. Kotov, C. Zhai, and R. Sproat. Mining named entities with temporally correlated bursts from multilingual web news streams. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, 2011.
8. T. W. Liao. Clustering of time series data - a survey. *Pattern Recognition*, 38:1857–1874, 2005.
9. A. Mueen, S. Nath, and J. Liu. Fast approximate correlation for massive time-series data. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 2010.
10. C. Ratanamahatana, J. Lin, D. Gunopulos, E. Keogh, M. Vlachos, and G. Das. Mining time series data. In *Data Mining and Knowledge Discovery Handbook*. CRC Press, 2010.

11. M. Vlachos, C. Meek, Z. Vagena, and D. Gunopulos. Identifying similarities, periodicities and bursts for online search queries. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004.
12. M. Vlachos, K.-L. Wu, S.-K. Chen, and P. S. Yu. Correlating burst events on streaming stock market data. *Data Mining and Knowledge Discovery*, 16(1):109–133, 2008.
13. X. Wang, C. Zhai, X. Hu, and R. Sproat. Mining correlated bursty topic patterns from coordinated text streams. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.
14. X. Wang, K. Zhang, X. Jin, and D. Shen. Mining common topics from multiple asynchronous text streams. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, 2009.
15. D. Wu, Y. Ke, J. X. Yu, P. S. Yu, and L. Chen. Detecting leaders from correlated time series. In *Proceedings of the 15th International Conference on Database Systems for Advanced Applications*, 2010.
16. K.-L. Wu, S.-K. Chen, and P. S. Yu. Query indexing with containment-encoded intervals for efficient stream processing. *Knowl. Inf. Syst.*, 9(1):62–90, 2006.
17. Y. Zhu and D. Shasha. StatStream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.