Interactive Exploration of Correlated Time Series

Daniel Petrov, Rakan Alseghayer, Mohamed Sharaf*, Panos K. Chrysanthis, Alexandros Labrinidis
Dept. of Computer Science, University of Pittsburgh, Pitsburgh, PA, USA
*University of Queensland, Brisbane, Australia
dpetrov,ralseghayer,panos,labrinid@cs.pitt.edu,*m.sharaf@uq.edu.au

ABSTRACT

The rapid growth of monitoring applications has led to unprecedented amounts of generated time series data. Data analysts typically explore such large volumes of time series data looking for valuable insights. One such insight is finding pairs of time series, in which subsequences of values exhibit certain levels of correlation. However, since exploratory queries tend to be initially vague and imprecise, an analyst will typically use the results of one query as a springboard to formulating a new one, in which the correlation specifications are further refined. As such, it is essential to provide analysts with quick initial results to their exploratory queries, which allows for speeding up the refinement process. This goal is challenging when exploring the correlation in a large search space that consists of a big number of long time series. In this work we propose search algorithms that address precisely that challenge. The main idea underlying our work is to design priority-based search algorithms that efficiently navigate the rather large space to quickly find the initial results of an exploratory query. Our experimental results show that our algorithms outperform existing ones and enable high degree of interactivity in exploring large time series data.

CCS CONCEPTS

•Information systems →Information retrieval; Users and interactive retrieval; *Personalization;*

KEYWORDS

time series, data exploration, search, subsequence

ACM Reference format:
Daniel Petrov, Rakan Alseghayer, Mohamed Sharaf*, Panos K. Chrysanthis, Alexandros Labrinidis. 2017. Interactive Exploration of Correlated Time Series. In *Proceedings of ExploreDB'17, Chicago, IL, USA, May 14-19, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3077331.3077335

1 INTRODUCTION

In recent years, more and more individuals and companies are collecting data on natural phenomena, social and socio-technological processes over time in order to quantify them and also to study how they change over time. Applications, which produce and deal with such data are medical data [9], location-based services [2], wireless sensor networks, [1, 3, 15, 16] and finances [6] – to name a few. A common method to get a better understanding of the observed behavior conveyed in these datasets is to find correlations in the time series data [5]. The correlation can be also used as a source to manipulate the data further like finding similarity measures faster [11], running threshold queries, [17] or reducing the size of the data, yet preserving some of its characteristics [8].

Finding correlations in time series (TS) data is a challenging task. Traversing the data and calculating the correlation is computationally expensive and introduces significant delay in the production of results. One way to address the challenge is among the lines of navigating the search space [5]. An alternative approach is to index the time series data [4, 5, 7, 18]. Predominantly the users look for pairs of highly correlated TS, and a high number of TS implies an even bigger number of pairs to be compared – precisely $\frac{n*(n-1)}{2}$ pairs for n TS. Often TS data consists of a big number of very long TS, that would not all fit into memory.

The big number of long TS further complicates data exploration where an analyst is interested in using the results of one query as a springboard to formulating new queries, in which the correlation specifications are further refined. In order to support data exploration, it is essential to provide analysts with quick initial results to their exploratory queries, which allows for speeding up the refinement process. In other words, there is a need for algorithms that (1) quickly identify subsequences of highly correlated time series data and (2) provide results within an interactive time frame. Existing work has primarily addressed the former [12, 13], which was mainly focused on correlating semi-finite TS. These current approaches for identifying pairs of correlated time series are designed to produce a full set of results before presenting it to the user. Thus, all of them fall short of supporting interactive exploration, which imposes an upper bound on the time needed to present the first results and returns results incrementally. These works are in the time domain, i.e., working with the raw time-ordered data.

In this paper, we address the latter, an interactive framework in the time domain, which continuously and incrementally provides results to the user. To the best of our knowledge, there is no such system, which identifies correlated TS and provides incremental results within an interactive time frame with accuracy of 100% while exploring the data space. Our solution is based on the use of the Pearson Correlation Coefficient (PCC) as a metric of correlation of two subsequences of time series data and on the hypothesis that interactivity can be achieved by integrating caching and scheduling principles.

In this paper we make the following contributions:

- We formulate the problem of pairwise correlation of time series data, which produces results in interactive amount of time (Sec. 2).

ExploreDB'17, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/3077331.3077335

- We propose two primary algorithms. Our baseline one (*iBRAID*) explores the pairs in round robin fashion. Our flagship algorithm (*PriCe*) utilizes a priority function based on historical success rate, cost, and PCC (Sec. 3).
- We present an experimental evaluation framework, which implements *iBRAID* and eight different variations of *PriCe*. Our experimental results using a real dataset show a speedup of up to 1500% of *PriCe* compared to *iBRAID* (Sec. 4).

2 PROBLEM DEFINITION

Without loss of generality, we consider the complete dataset to be available upfront, data to be numeric, all dimensions to be synchronized – i.e., the next timestamp of all of them is received by the system at the same point in time, all time series start at the same point in time, there are no missing timestamps, and all time series are of the same length.

We adopt the widely used Pearson correlation coefficient (PCC) as a measure of correlation between a pair of time series. Given two numeric time series x and y of equal length m, the PCC is calculated with the following formula:

$$corr(x,y) = \sum_{i=1}^{m} \frac{(x_i - \mu_x)(y_i - \mu_y)}{\sigma_x \sigma_y} \quad (1)$$

where μ_x is the average (or mean) of the values of x, μ_y is the mean of the values of y, σ_x and σ_y are the standard deviations of the values of x and y, respectively.

PROBLEM (Interactive Pairwise Correlation (IPC)): Let T be a multidimensional time series of data. Let x and y be dimensions in T. IPC finds all pairs of subsequences of time series x and y, which start at the same point in time and have equal length s, for which $corr(x,y) > \tau$ and delivers every matched pair as it is detected.

GOAL: The objective of IPC is to minimize the time to deliver the first r% of the results, where r is a relatively small number. The equivalent SQL IPC query is:

```
SELECT x, y
FROM T
WHERE corr(x,y) > τ AND SubsequenceLength = s
FETCH FIRST r% ROWS ONLY;
```

3 ALGORITHMS

In this section, we address the interactivity shortcoming of the current approaches, propose *iBRAID*, which is a baseline algorithm, and introduce *PriCe* for interactive pairwise correlation of subsequences of time series data.

3.1 Naive Approach

The most obvious approach to find and report correlated subsequences in a TS dataset is to start with a single pair of time series, pick subsequences of them, which start at the same point in time and calculate the Pearson Correlation Coefficient (PCC) for them. Subsequently, continue the PCC calculations for other subsequences of the same pair of time series, or move to calculate the PCC for the corresponding subsequences of other pairs of time series until all possible pairs of subsequences in TS dataset are examined. Finally, return all pairs of subsequences whose PCC meets a threshold τ.

The major cost of the naive approach is in computing the PCC for each pair of subsequences, which requires at least two traversals of the values in these subsequences (Eq. 1). The first one is to calculate the average (or mean) of each subsequence. The second one is to calculate the standard deviation and the inner cross product of the two subsequences. The total number of traversals for each subsequence is in the orders of magnitude of the length of the subsequence, multiplied by the number of pairs it is part of. The naive approach illustrates the crux of the IPC problem, which is the high number of calculations required to explore the dataset.

3.2 iBRAID

The *BRAID* technique [13] partially solved the problem of the naive approach by efficiently calculating the PCC based on five basic and computationally cheap statistics: the *sum of the elements in each subsequence*, the *sum of the squares of the elements of each subsequence*, and the *inner crossproduct of the elements of the two sub sequences* for which the correlation is calculated.

The following notation is adopted for the rest of the paper. The sum of the elements of a subsequence of length m of time series x is denoted

$$sumx = \sum_{i=1}^{m} x_i$$

The same way, the sum of the square of the elements is denoted

$$sumxx = \sum_{i=1}^{m} x_i^2$$

The inner product will be denoted

$$sumprodxy = \sum_{i=1}^{m} x_i y_i$$

The covariance of the two time series x and y is

$$cov = sumprodxy - \frac{sumx \times sumy}{m}$$

The variance of the subsequence can be calculated according to the following formula

$$varx = sumxx - \frac{(sumx)^2}{m}$$

Similarly, the variance for time series y will be denoted $vary$. Then the PCC can be calculated byapplying the following formula

$$corr(x,y) = \frac{cov}{\sqrt{varx \times vary}}$$

The essential statistics can be computed either at once or incrementally, each time a pair of subsequences gets explored. In the case of incremental calculation, the sums that are stored in memory are incremented by the new values added and decremented by the values that are not part of the subsequences anymore. The same operations are done for the sums of the squares and the inner cross products using the respective values.

As we discussed earlier, the naive approach is to arbitrarily select two time series, calculate the correlation for a pair of subsequences, explore it by one value, and recalculate the correlation until the end of the time series is reached. The next step is to arbitrarily pick different pairs of time series and run the same algorithm. This should be done until all pairs are explored. This algorithm does

not have any prior knowledge about the data and does not use any results as a decision making input to speed up the production of results. Additionally, each value in each time series is touched as many times as are the different pairs of time series in the dataset. This makes the computation cost of the algorithm prohibitively expensive.

We propose a computationally cheaper algorithm, which produces the same results as the naive one but with fewer computations and returns the results as soon as they are produced. We explore the dataset sequentially, starting from the first value for all time series. We calculate the essential statistics for each subsequence of each time series and the inner cross product of pairs of time series. The next step is to calculate the correlations for all pairs of subsequences, starting from the first value. Once this is done, the subsequences are explored further by one value, the essential statistics are updated incrementally - the first value is expired / subtracted from them and the new value is added. The Pearson Correlation is calculated again for all pairs. The rest of the steps are to keep exploring all time series by a single value, augment the essential statistics incrementally, and recalculate the correlation. These steps are repeated until the whole dataset is explored.

We named the algorithm "iBRAID", paying tribute to the authors of BRAID [13] and emphasizing its focus on producing results in interactive amount of time. The algorithm has a number of advantages: it is accurate, easy to implement, and does not cause "starvation" among the pairs. In other words, all pairs are considered (i.e., new values are included and old values are excluded) at each step. Additionally, it reduces the amount of computations by half due to the usage of the five essential statistics. This algorithm is expected to perform well for datasets whose data is uniformly distributed. On the other hand, it might underperform on skewed datasets. This hypothesis has motivated our second technique, which is discussed next.

3.3 PriCe

By reusing partial PCC computations, *iBRAID* captures one part of our hypothesis, that interactivity can be achieved by utilizing caching and scheduling principles. In this section, we propose a novel algorithm that captures both.

PriCe is a more informed searching algorithm; it uses a priority function to explore the pairs of subsequences while reusing partial PCC computations as *iBRAID*. The idea of *PriCe* is to explore the most promising pair first, which is the one with the highest priority function value. We designed the following priority function:

$$PCC * (M/totalExp)/C \quad (2)$$

where *PCC* is the most recent calculated correlation for a pair of subsequences that belong to the same pair of time series, M is the number of produced results that match the query (i.e., pair of subsequences) by a pair of time series so far, $totalExp$ is the total number of explored pair of subsequences, and C is the cost of exploring a pair of subsequences.

PriCe gives the highest priority to the pair of time series that have a history of high number of results produced so far and high recent calculated PCC. This way, we capture the idea of space locality along with temporal locality: where *space locality* is captured by the PCC, and *temporal locality* is captured by the ratio of the results to the total number of explored pairs of subsequences. Moreover, we include the cost to explore a pair of subsequences into the function.

The cost in the priority function is the number of operations needed to calculate the essential statistics for a pair of subsequences. For example, if a pair of time series shares one time series with another pair. Then, the more advanced one (i.e., the one that is at a higher timestamp) has already calculated the sums of the time series and the sums of the squares. This leaves the pair that is lagging behind with lower cost to advance, since the more advanced one has already computed some of the essential statistics for that shared time series.

PriCe has two modes of execution: *preemptive* and *non-preemptive*. In the non-preemptive mode, if the algorithm declares a pair of subsequences as a result after exploring it (i.e., the calculated PCC is above τ), it further continues exploring the next pair of subsequences of that pair of time series. By doing that, we try to leverage the space locality of a result. The algorithm ceases to explore that pair of time series when it encounters the very first pair of subsequences that does not count as a result (i.e., the calculated PCC is below or equal τ). After that, it reevaluates the priority function, and elects the next pair to explore accordingly. On the other hand, the preemptive mode would consult the priority function after each single exploration of any pair of subsequences, irrespective of the calculated PCC.

Table 1: PriCe and its variations

Abbreviation	Name
PriCe M Ratio [P | NP]*	Original PriCe
PriCe M [P | NP]	PriCe with only num. of results
PriCe C [P | NP]	PriCe with PCC/C
PriCe [P | NP]	PriCe with PCC only

*[P | NP] = [Preemptive | Non-preemptive]

4 EXPERIMENTS AND ANALYSIS

In this section we present initial results from the evaluation of our proposed algorithms. In addition to the comparison between *iBRAID* and *PriCe*, we study the significance of each parameter of the priority function of *PriCe* (Eq. 2).

4.1 Experimental Framework

Algorithms In addition to *iBRAID* and *PriCe*, we generated three variants of *PriCe* by setting some of the parameters of its priority function (Eq. 2) to 1. All variations are summarized in Table 1. The first variation of the priority function is PCC/C (i.e., $M/totalExp$= 1). This variation is denoted as *PriCe C*, which is less informed than the original *PriCe* (denoted as *PriCe M Ratio*). Similarly, we have come up with a second variation denoted as *Price M*, where the cost is only the number of results produced so far M (i.e., $totalExp$ =1 and C=1). *Price M* is also less informed compared to PriCe M Ratio, as it does not take into consideration the cost to explore the pair C nor the total number of explored subsequences so far $totalExp$. The third variation is the simplest, and it has the priority function *PCC*. This variation is denoted as *PriCe*. This means that the algorithm will elect the pair of time series that have the highest PCC for the most recently explored pair of subsequences. The downside of this

scheme is that it does not account for the cost to explore the pair of subsequences, which affects the interactivity of the algorithm. Also, it lacks foresight, as it fails to capture the number of results a pair of time series has produced so far.

Testbed We implemented all the discussed algorithms and their variations in both Java 1.8 and in C++ 11. We ran the experiments on a computer with 2 Intel CPUs, running at 2.66GHz, and 96GB of RAM memory. The operating system used was CentOS 6.5 and the compiler was GCC version 4.8.2.

Metrics We evaluated the performance of the algorithms in terms of both response time and memory footprint.

Response time: We measured the latency in both the *number of operations* performed to reach the first r% of the results (i.e., pairs of subsequences) and *wall clock time* needed to reach the first r% of the results. We used the number of operations as it provides the asymptotic efficiency of the algorithms compared to one another. This does not depend on factors such as the hardware characteristics and the operating system of the computer, which the experiments are run on, nor the efficiency of the compiler / virtual machine, which compiles and/or executes the code. At the same time, we use the wall clock time to measure the impact of such dependencies in the runtime environment mentioned above.

Memory footprint: We measured the number of values of the five essential statistics, stored in memory for each pair of subsequences.

Dataset *Yahoo Finance Historical Data* [6]: The dataset we have used in our experiments consists of 318 time series. Those reflect the trading of 56 companies on the NYSE for the last 28 years. This gives us a total of 50403 different pairs to query. The data granularity is a day, which includes the price of the stock of the company at opening, the price at the end of the day (closing), the highest price for the day, the lowest price for the day, the amount of shares traded that day, and the adjusted close (calculated according to the standards of the CRSP, Center for Research in Security Prices). The length of each time series is about 7100 timestamps.

4.2 Experimental Results

Our objective has been to get as many results as possible while maintaining interactivity. In this section, we present the results of four experiments that we conducted to evaluate the interactivity and the sensitivity of our proposed algorithms to the length of subsequences and the target correlation threshold. Recall that all variations of *PriCe* and their names in the figures are summarized in Table 1.

Experiment 1: In our first experiment, we measured the latency of each algorithm to produce the first 1%, 5%, 10% and 20% of the results, measured with our first metric of response time, i.e., *number of operations* (Fig. 1) as well as the memory footprint, measured as number of (cached) values in memory (Fig. 2). The target correlation threshold for this experiment is 0.9 and the subsequence length is set to 64 timestamps. All 8 variations of *PriCe* outperform the baseline algorithm *iBRAID*. Most of *PriCe* variations have comparable performance. Our flagship algorithm *PriCe M Ratio [P]* requires only 0.06634 of the operations performed by the baseline to produce the first 20% of the results – 34076581 vs 927230124. This is a speed-up of more than *15 times*.

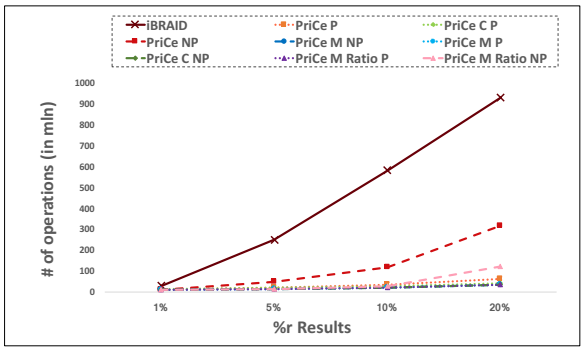

Figure 1: The cost in number of operations to deliver the first 1%, 5%, 10%, and 20% of the results.

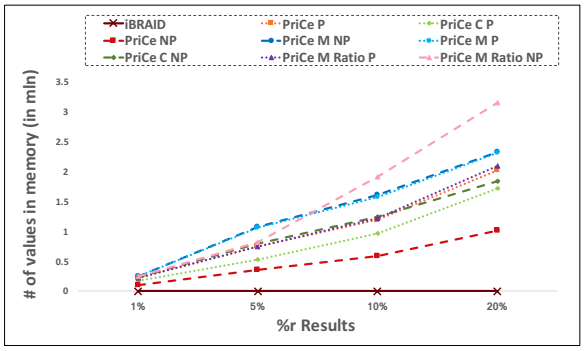

Figure 2: The memory footprint for each algorithm at 1%, 5%, 10%, and 20% of the results.

The rate at which the memory cost increases for *PriCe* is significantly faster than the baseline *iBRAID*. *iBRAID* occupies a fixed amount of memory, as all pairs get explored together, and the overhead of essential statistics is minimized. It is not a surprise that *PriCe M Ratio [P]* has a bigger memory footprint than *iBRAID*, but it outperforms the other three variations of the *PriCe* algorithm.

Experiment 2: In our second experiment, we studied the sensitivity of the algorithms to the length of the subsequence. We set the target threshold to be 0.9, and then, measured the cost and the memory footprint to produce the first 20% of the results. We ran the algorithms for 6 different lengths of the subsequences – 8, 16, 32, 64, 128, and 256 (Fig. 3). *PriCe M Ratio [P]* consistently outperforms all the algorithms for subsequence length of 32 or more timestamps, and it shows comparable results with the other variations of *PriCe* for shorter subsequences (30 M). The baseline algorithm exhibits between 2 and 15 times higher cost compared to *PriCe M Ratio [P]* – between 400 M and 600 M

Similarly to the previous experiment, *iBRAID* has the smallest memory footprint. *PriCe M Ratio [P]* has comparable footprint to *PriCe [P]* and smaller than *PriCe [NP]* and *PriCe M [P]* (Fig. 4).

Experiment 3: In our third experiment, we studied the sensitivity of our algorithms to the target threshold. We set the subsequence length to 64 timestamps, and we ran it for 9 different values of τ – $0.1, 0.2, \ldots, 0.9$. The statistics are for the first 20% of the results (Fig.

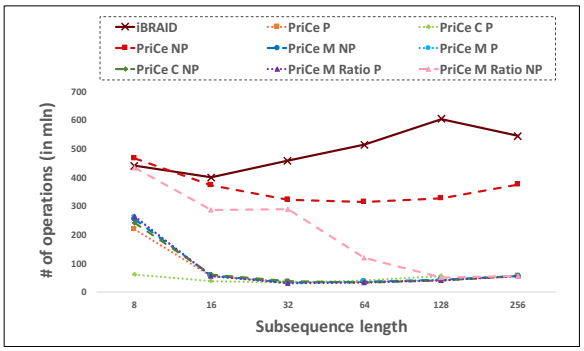

Figure 3: The cost in number of operations to deliver the first 20% of the results.

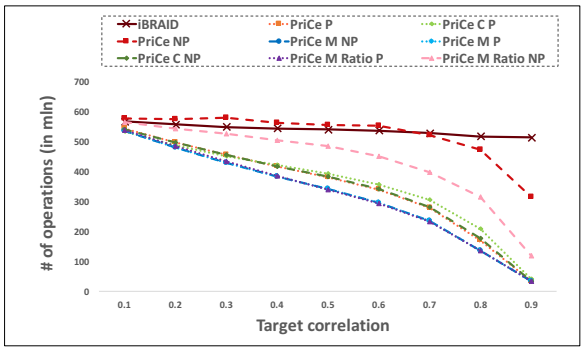

Figure 5: The cost in number of operations to deliver the first 20% of the results.

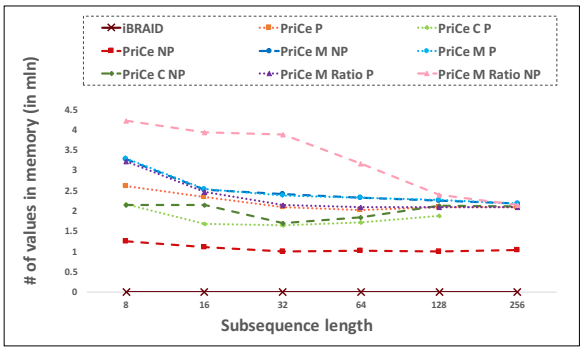

Figure 4: The memory footprint for each algorithm at 1%, 5%, 10%, and 20% of the results.

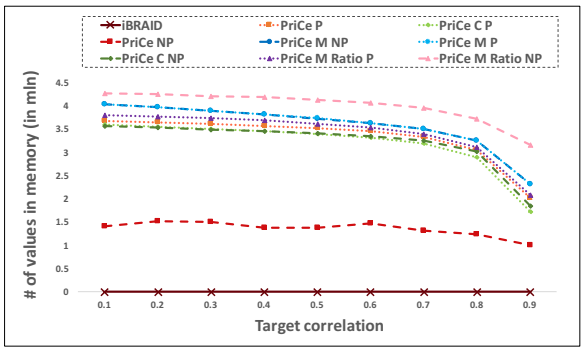

Figure 6: The memory footprint for each algorithm at 20% of the results.

5). Our flagship algorithm *PriCe M Ratio [P]* exhibits the smallest cost for all values of the target threshold in comparison with the other algorithms. For the highest target correlation 0.9, out flagship algorithm shows a better performance by 15 times, as discussed earlier in Experiment 1.

The memory footprint of *PriCe M Ratio [P]* is significantly larger than the baseline algorithm *iBRAID*. This is expected, as the different time series are not explored together, which requires a number of statistics to be (cached) in memory. Our flagship requires less memory than two of the other algorithms and has comparable results with four of the variations of *PriCe* (Fig 6).

Experiment 4: In our last experiment, we studied the overhead of running our algorithms in a given environment. For this purpose, in this experiment we measure the latency of each algorithm using our second metric for response time - namely *wall clock time*.

For consistency, in this experiment, we reuse the setup of Experiment 1 – the subsequence length is set to 64, the target threshold correlation is 0.9 and we measure the response time to produce the first 1%, 5%, 10% and 20% of the results.

We ran the experiment for a subset of the dataset - only 2 companies, which corresponds to 12 time series. Figure 7 shows the results for our implementation in C++. The results for our implementation in Java are similar even though the performance gap between *PriCe M Ratio [P]* and the baseline *iBRAID* is larger.

In Figure 7, the wall clock metric shows a difference of only 1.5 times between *PriCe M Ratio [P]* and *iBRAID*, despite the significant difference in terms of number of operations – 15 times. The former took 871666 msec to deliver the first 20% of the results; the latter took 1221845 msec to achieve the same task. This clearly shows that despite the improvements in computing PCC, the context switch to compute the priority function and movement of data from memory to the CPU is significant. As part of our future work, we plan to investigate ways to reduce this overhead. This experiment also shows that despite the significant overheads to compute the priority function, *PriCe M Ratio [P]* outperforms *iBRAID*.

The performance of the rest of the *PriCe* variations is comparable to *PriCe M Ratio [P]* and they all outperform *iBRAID*. We ran the experiment for subsequence length 8, 16, 32, 128 and 256 as well, and we got similar results. In all cases, *PriCe M Ratio [P]* consistently performs better than *iBRAID*.

5 RELATED WORK

The processing of data and fast discovery of highly correlated subsequences of time series (TS) is tackled in two scenarios with respect to the production of data – static, when the data is collected upfront and it forms the search space for finding the correlated subsequences [7, 10, 13], and dynamic, when the data is processed as it is produced – the scenario of datastreams [12, 14]. The latter

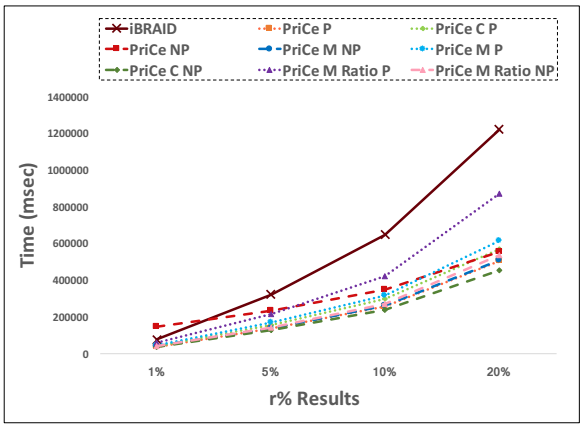

Figure 7: Wall clock time (C++)

is beyond the scope of our work. Also, works on approximate solutions and ones that use techniques in the frequency domain (e.g., [10]) are beyond the scope of this paper.

In BRAID [13] the authors propose a technique to find correlated pairs of subsequences. Their work extends to the point that the two subsequences might not start at the same point in time - there might be a lag between them - i.e. the subsequences are of the same length, but one of them starts l timestamps after the other one. They navigate the data space sequentially and preemptively. They calculate the PCC of each pair of subsequences with lag 0 as they navigate the space and use these values to average out (or "smoothen") the PCC for larger lags. The authors use powers of 2 in order to build a multilevel "smoothened" PCC as a geometric progression of the lag. Our work can be extended to support lagged pairs of subsequences at the cost of keeping in memory the essential statistics from the timestamps at which each of the subsequences starts. Our work also differs from BRAID as we focus on interactive exploration of the correlated subsequences and our results are always 100% accurate as we do not estimate the PCC and our algorithms always find all pairs of correlated subsequences.

The authors of [7] propose an extension to SQL, which allows the definition of new types of queries. Those cannot be expressed easily with the traditional operators such as *GROUPBY* and *COUNT* – queries, which run arithmetic operators over ranges of data entries. They also propose a sampling-guided, data-driven search space navigation technique for interactive data exploration. They build a grid over the search space and calculate a number of attributes for each cell of the space. In their example, they use the SDSS dataset, and the attributes they calculate for each cell are, for example, average brightness of the cell and the number of stars in the cell. These attributes are precomputed offline. They use a best-first heuristic and a priority queue to navigate the order of exploration of the cells. The algorithms we propose also prioritize the exploration of pairs of subsequences, which are more likely to produce results. Unlike [7], we use the smallest step possible for advancement of the subsequences - one timestamp. The granularity of the grid might impact significantly the search space navigation. We also do not precompute any data.

6 CONCLUSIONS

In this paper we presented a number of priority-based search algorithms for interactive exploration of correlated time series (TS). Our work aims to assist analysts in finding pairs of TS, in which subsequences of values exhibit certain levels of correlation. Providing initial results to the user in interactive fashion helps her to refine further her exploratory queries, which allows for speeding up the refinement process. Our solution uses the Pearson Correlation Coefficient (PCC) as a metric of correlation of two subsequences of time series, and it is trivial to extend it to negative correlation and self correlation as we consider the absolute value of the PCC, and we are not limited to one subsequence per time series.

Our experimental evaluation using real data shows that our flagship algorithm outperforms the baseline approach by more than 1500%. This is attributed to our algorithm using a priority function that accounts for PCC and considering the computation cost, the number of results that are produced by a pair of TS, and the total number of explored pairs of subsequences at any given point in time.

Acknowledgments. We would like to thank the anonymous referees for their helpful comments and suggestions for improving this paper.

REFERENCES

[1] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. 2003. Evaluating Probabilistic Queries over Imprecise Data *ACM SIGMOD'03*, 551–562.
[2] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. 2004. Querying imprecise data in moving object environments. *IEEE Transactions on Knowledge and Data Engineering* 16(9):1112–1127.
[3] Michele Dallachiesa, Gabriela Jacques-Silva, Buğra Gedik, Kun-Lung Wu, and Themis Palpanas. 2015. Sliding Windows over Uncertain Data Streams. *Knowledge and Information Systems* 45(1):159–190.
[4] Kaiyu Feng, Gao Cong, Sourav S. Bhowmick, Wen-Chih Peng, and Chunyan Miao. 2016. Towards Best Region Search for Data Exploration *ACM SIGMOD'16*, 1055–1070.
[5] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. 2015. Overview of Data Exploration Techniques *ACM SIGMOD'15*, 277–281.
[6] Yahoo Inc. 2016. Yahoo Finance Historical Data. 2016, https://finance.yahoo.com/quote/YHOO/history
[7] Alexander Kalinin, Ugur Cetintemel, and Stan Zdonik. 2014. Interactive Data Exploration Using Semantic Windows *ACM SIGMOD'14*, 505–516.
[8] Dongeun Lee, Alex Sim, Jaesik Choi, and Kesheng Wu. 2016. Novel Data Reduction Based on Statistical Similarity *SSDBM '16*, 21:1–21:12.
[9] X. Lian, L. Chen, and J. X. Yu. 2008. Pattern Matching over Cloaked Time series *IEEE ICDE'08*, 1462–1464.
[10] Abdullah Mueen, Suman Nath, and Jie Liu. 2010. Fast Approximate Correlation for Massive Time-series Data *ACM SIGMOD'10*, 171–182.
[11] Mahsa Orang and Nematollaah Shiri. 2015. Improving Performance of Similarity Measures for Uncertain Time series Using Preprocessing Techniques *SSDBM '15*, 31:1–31:12.
[12] Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. 2005. Streaming Pattern Discovery in Multiple Time-series *VLDB '05*, 697–708.
[13] Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. 2005. BRAID: Stream Mining Through Group Lag Correlations *ACM SIGMOD'05*, 599–610.
[14] Ilari Shafer, Kai Ren, Vishnu Naresh Boddeti, Yoshihisa Abe, Gregory R. Ganger, and Christos Faloutsos. 2012. RainMon: An Integrated Approach to Mining Bursty Timeseries Monitoring Data *ACM KDD '12*, 1158–1166.
[15] Emma M. Stewart, Anna Liao, and Ciaran Roberts. 2016. Open PMU: A Real World Reference Distribution Micro-phasor Measurement Unit Data Set for Research and Application Development. 10/2016 2016,
[16] G. Trajcevski, A. Choudhary, O. Wolfson, L. Ye, and G. Li. 2010. Uncertain Range Queries for Necklaces *IEEE MDM'10*, 199–208.
[17] Eleni Tzirita Zacharatou, Farhan Tauheedz, Thomas Heinis, and Anastasia Ailamaki. 2015. RUBIK: Efficient Threshold Queries on Massive Time series *SSDBM '15*, 18:1–18:12.
[18] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2014. Indexing for Interactive Exploration of Big Data series *ACM SIGMOD'14*, 1555–1566.