DCS: A Policy Framework for the Detection of Correlated Data Streams

Rakan Alseghayer[1(✉)], Daniel Petrov[1], Panos K. Chrysanthis[1(✉)], Mohamed Sharaf[2], and Alexandros Labrinidis[1]

[1] University of Pittsburgh, Pittsburgh, PA, USA
{ralseghayer,dpetrov,panos,labrinid}@cs.pitt.edu
[2] Department of Computer Science and Software Engineering, College of Information Technology, United Arab Emirates University, Al Ain, UAE
msharaf@uaeu.ac.ae

Abstract. There is an increasing demand for real-time analysis of large volumes of data streams that are produced at high velocity. The most recent data needs to be processed within a specified delay target in order for the analysis to lead to actionable result. To this end, in this paper, we present an effective solution for detecting the correlation of such data streams within a micro-batch of a fixed time interval. Our solution, coined DCS, for Detection of Correlated Data Streams, combines (1) incremental sliding-window computation of aggregates, to avoid unnecessary re-computations, (2) intelligent scheduling of computation steps and operations, driven by a utility function within a micro-batch, and (3) an exploration policy that tunes the utility function. Specifically, we propose nine policies that explore correlated pairs of live data streams across consecutive micro-batches. Our experimental evaluation on a real world dataset shows that some policies are more suitable to identifying high numbers of correlated pairs of live data streams, already known from previous micro-batches, while others are more suitable to identifying previously unseen pairs of live data streams across consecutive micro-batches.

1 Introduction

Motivation. More and more organizations (commercial, health, government, and security) currently base their decisions on real-time analysis of business and operational data in order to stay competitive. Towards this, they deploy a variety of monitoring applications to analyze large volumes of live data streams, that are produced at high velocity. Data analysts explore such large volumes of data streams, typically representing time series of raw measures, looking for valuable insights and interesting events.

A common method for getting a better understanding of the observed behavior conveyed in a set of data streams is to find correlations in the data streams

M. Castellanos et al. (Eds.): BIRTE 2015/2016/2017, LNBIP 337, pp. 191–210, 2019.
https://doi.org/10.1007/978-3-030-24124-7_12

[1]. The correlation can be also used as a source for finding similarity measures faster [2], running threshold queries [3], or reducing the size of the data, but preserving some of its characteristics [4].

Challenges. Finding correlations in data streams is a challenging task. Current methodologies approach this challenge by employing some prediction techniques [5], Discrete Fourier Transform approximations [6,7], or using clustering and Markov chain modeling [8]. All those approaches have their limitations, whether due to lack of absolute precision as a result of using approximations or predictions, or due to the usage of computationally expensive operations. Other approaches address this challenge by indexing the data series [9–11]. Predominantly the users are looking for pairs of (positively or negatively) correlated data streams over a short period of time. The high number of data streams implies an even bigger number of pairs—precisely $\frac{n*(n-1)}{2}$ pairs for n data streams. The time to explore completely all pairs on one computer may be prohibitively long. The challenge is exacerbated when the demand is for answers in *real-time* and for a large set of *live* data streams.

Problem Statement. Clearly there is a need for algorithms that quickly identify windows of correlated data streams. In our prior work [12,13], we proposed such algorithms, called *iBRAID-DCS* and *PriCe-DCS*, which detect pairs of correlated data streams within micro-batches of data streams with specific intervals. They both uses the Pearson Correlation Coefficient to correlate two windows of pairs of data streams.

As long as the detection of correlated pairs is considered independent across micro-batches, *PriCe-DCS* executes in the same fashion within each micro-batch, identifying the highest possible number of correlated pairs. However, there are exploration tasks that *do* consider the detection of correlated pairs across micro-batches. For example, in some tasks, the goal is to detect as many *unique* pairs of correlated data streams as possible across two consecutive micro-batches, while in others, the goal is to *assure* the perpetual correlation between them.

To that end, in this paper, we study nine policies to address the different requirements of exploration tasks. These policies may leverage prior knowledge (i.e., exploit already detected correlated data streams in preceding micro-batches) to steer the detection of correlated pairs in the current micro-batch.

Contributions. In summary, in this paper:

- We present our *DCS* (*Detection of Correlated Data Streams*) framework that employs the two base algorithms *iBRAID* and *PriCe* and is driven by a utility function [12,13], (Sect. 2).
- We propose nine different policies that our novel *PriCe-DCS* algorithm can employ when analyzing consecutive micro-batches. These policies can increase the efficiency when detecting correlated live data streams and/or address different exploration requirements. By appropriately tuning the parameters of the utility function, the different policies exhibit different *detection-recall, overlapping-recall*, and *diversity* results (Sect. 3).

– We experimentally evaluate and compare our algorithms, along with the behavior of the nine detection policies. Our results using a real world dataset show that our *PriCe-DCS* outperformed all the other algorithms. Furthermore, with the policies *Blind* and *X% Probing* (i.e., 1% Probing and 5% Probing), *PriCe-DCS* was able to identify more diverse correlated data streams across micro-batches than the *Informed* policy. On the other hand, *P-Alternating* and *Informed* policies were able to assure the existence of correlation in already identified correlated data streams (i.e., high *overlapping-recall*). The other policies exhibited mixed behavior (Sect. 4).

We discuss related work in Sect. 5 and conclude in Sect. 6.

2 DCS Framework

In this section, we review our *DCS* mode of operation, introduced in [13], its optimization objective, and our novel algorithm *PriCe-DCS* that implements its objective.

2.1 System Model

Without loss of generality, we consider a (monitoring) system that receives data from n data streams. Each data point in a data stream is a tuple t consisting of a timestamp ts and a numeric value val ($t = (ts, val)$). The timestamp captures the moment in time when the tuple was produced.

The data is produced at high velocity. The different streams produce the consecutive tuples at the same rate, and they are all synchronized. However, there are techniques to determine missing values, and to synchronize data which arrives at different rates, but they are beyond the scope of this paper.

The real-time analytical processing is performed in *micro-batches*.

Definition 1. *A micro-batch is a group of synchronized tuple subsequences over a set of data streams defined by a timestamp* interval I.

Each micro-batch, whether of the same or different data streams, is of the same size, i.e., contains the same number of tuples with consecutive timestamps within the interval. The inter-arrival time of two consecutive micro-batches specifies the maximum computational time for processing a micro-batch.

Definition 2. *The inter-arrival time is the* delay target *or* deadline d *by which the last result can be produced while analyzing a micro-batch.*

In real-time processing, ideally, the deadline d equals to the interval ($d = I$) so that there is no delay gap in processing between two consecutive micro-batches. However, it is expected to be a bit longer due to various overheads in the system, including any pre-processing of micro-batches.

2.2 Optimization Objective

Our *DCS* framework focus on analytical processing that finds correlated data streams in real-time, using the Pearson Correlation Coefficient (PCC) as a correlation metric for pairs of sliding windows of data streams.

Definition 3. *Given two numeric data streams x and y of equal length m, the PCC is calculated with the following formula:*

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

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.

Definition 4. *Two sliding windows of the same range w with a slide of 1 are correlated when the PCC is more than a given threshold τ ($PCC \geq \tau$).*

Definition 5. *A pair of data streams in a micro-batch is correlated when it contains at least A correlated sliding windows with threshold τ.*

The windows, which meet the criterion, may be consecutive or stratified over the interval defining a micro-batch.

The formalization of the algorithmic problem is as follows:

Problem: *Given a micro-batch B of a set of data streams DS with an arrival interval I, perfectly synchronized and with no missing tuples, and a deadline d, detect the number of correlated pairs of data streams, each of which has A correlated sliding windows, not necessary consecutive, with a PCC threshold of τ, by the deadline d.*

The optimum solution will be when the number of identified correlated pairs in a micro-batch are equal to the actual total number of correlated pairs. Hence, the optimization goal in DCS is to maximize the number of identified pairs by a deadline. Formally, the ratio of number of detected correlated pairs to the total number of correlated pairs is close to 1 and the metric is defined as:

$$Detection\text{-}Recall = \frac{\#\ identified\ correlated\ pairs}{Actual\ \#\ correlated\ pairs} \tag{2}$$

2.3 Base Algorithms

IBRAID-DCS is an enhancement over the work BRAID [14], where the PCC can be calculated by computing five sufficient statistics—sum of the tuples in each window, the sum of the squares of the tuples of each window, and the inner cross-product of the tuples of the two windows, for which the correlation is calculated. The sum ($sumx$) and the sum of the square of the tuples ($sumxx$)

of a window of length m of a data stream x, and corresponding inner product ($sumprodxy$) are denoted as

$$sumx = \sum_{i=1}^{m} x_i \qquad sumxx = \sum_{i=1}^{m} x_i^2 \qquad sumprodxy = \sum_{i=1}^{m} x_i y_i$$

The covariance of the two data streams x and y is

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

and the variance of the window can be calculated as according to the following formula

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

Similarly, the variance for data stream y will be denoted $vary$. Then PCC can be calculated, applying the following formula

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

The sufficient statistics can be computed either from scratch or incrementally each time a pair of data streams is explored by a new tuple. In the case of incremental calculation, the sums stored in memory are incremented by the new values and decremented by the values that are not part of the windows anymore. The same operations are performed for the sums of the squares and the inner cross products, using the respective tuples.

With that in mind, *iBRAID-DCS* is a round-robin scanning algorithm that uses the incremental computation of PCC. It analyzes the pairs of data streams in a micro-batch sequentially, starting from the first tuple for all data streams. It calculates the sufficient statistics that are needed to calculate the PCC efficiently (i.e., single pass over tuples). Next, it calculates the PCC for the first tuple for all pairs of windows. Once this is done, the windows are slid further by one tuple, the sufficient statistics are updated incrementally—the first tuple is expired/subtracted from them, and the new tuple is added. The PCC is calculated again for all pairs. Then, it keeps analyzing all data streams by a single tuple, augmenting the sufficient statistics incrementally, and recalculating the PCC. This is done until the whole micro-batch is analyzed.

iBRAID-DCS has four key advantages: (1) it is accurate, (2) easy to implement, (3) does not cause "starvation" among the pairs, and (4) it reduces the computations by half due to the usage of the sufficient statistics. *iBRAID-DCS* is experimentally shown to perform well for data streams whose data is uniformly distributed and for low correlation thresholds ($\tau < 0.5$).

PriCe-DCS is a scanning algorithm that uses a utility function to analyze the pairs of windows while reusing partial PCC computations. It analyzes the most promising pair first, which is the one with the highest utility function value:

$$Pr = PCC * (M/totalExp)/C \tag{3}$$

where PCC is the most recently calculated Pearson Correlation Coefficient for a pair of sliding windows that belong to the same pair of data streams, M is the number of correlated sliding windows found in the corresponding pair of data streams so far, $totalExp$ is the total number of analyzed pairs of sliding windows, and C is the cost of analyzing a pair of sliding windows in terms of number of computations (i.e., the number of operations needed to calculate the sufficient statistics for a pair of sliding windows). The default values are $PCC = 1$, $M = 0$, $totalExp = 1$, and $C = 1$.

Early termination happens when the A criterion of the number of correlated windows is reached for a pair, and pruning happens according to the following condition:

$$(A - correlatedWindows) > (I - slidingWindowPosition)$$

where $correlatedWindows$ are the total number of windows that are correlated in a pair of streams according to PCC τ, and $slidingWindowPosition$ is the pair's analysis location in the interval. Recall, I is the interval of the data streams.

3 Detection Policies

When the very first micro-batch arrives at the system, the system has no prior knowledge about any correlated pairs of streams. However, this is not the case after the analysis of any micro-batch that produces a set of correlated pairs of data streams. This raises the question of how to exploit the results of past micro-batch analyses, such as, picking the first pair in a new micro-batch to analyze. This question has a major impact on *PriCe-DCS*'s behavior in supporting *exploration, exploitation* or *fairness*, and its answer determines the initialization of the parameters of *PriCe-DCS*'s utility function.

In fact, we propose nine policies, which differ in the way each initializes the utility function of *PriCe-DCS*.

Blind. When the analysis of a micro-batch starts with no prior knowledge of correlated pairs of streams, *Blind* policy initializes *PriCe-DCS*'s utility function to its default values (as discussed in Sect. 2.3). It is to be noted, however, that the very first micro-batch analysis in all approaches follows the *Blind* approach[1].

Informed. The utility function is initialized based on the results of the latest micro-batch analysis. In *Informed* starting phase, *PriCe-DCS*'s utility function is initialized to the same parameter values of the correlated pairs used by the immediately previous micro-batch. The rationale behind this policy is to keep analyzing closely those pairs that already exhibited high correlation in the previous micro-batch, potentially indicating an insight of interest.

[1] In DCS [12,13], Blind was referred to as Cold Start whereas the other proposed policies here are instances of Warm Start.

Untouched. The focus is on the pairs that were not processed at all in the previous micro-batch due to the lack of any correlated windows (i.e., not chosen for analysis due to their low values of Pearson Correlation Coefficient) at the beginning of *PriCe-DCS*'s execution. Specifically, such pairs are jumpstarted by altering their previous number of correlated windows (i.e., the parameter that reflects this information) to have the value A. This increases their priority, preventing their starvation and giving them another chance to be analyzed in the new micro-batch. The rationale behind this policy is to allow such pairs another chance, potentially identifying different behavior, which remained undetected in the previous micro-batch.

Alternating. This policy gives the pairs that were not correlated in the previous micro-batch a chance to be explored through a hybrid round-robin fashion. In *Alternating* starting phase, alternately, a pair from those that are not correlated in the previous micro-batch, is picked and explored using *PriCe-DCS* followed by a pair from those which were correlated. When the starting phase concludes (i.e., touched all the pairs at least once), the pairs are processed with *PriCe-DCS* according to the utility function. By doing this, we hope to reduce the effect of starvation for those that were not correlated in the previous micro-batch.

X% Non-correlated. This policy tries to achieve fairness of exploration through jumpstarting the lowest X% pairs in priority. The pair with the highest priority among those lowest X% is picked and explored. This continues until all those X% pairs are jumpstarted. Subsequently, *PriCe-DCS* carries out the exploration process as it usually does.

Decaying History. This policy regards the significance of the whole historical correlation information of a pair differently from recent micro-batches information. In the utility function, it alters the parameter M, which reflects the number of correlated sliding windows found for a corresponding pair, such that it becomes weighted. It gives the historical correlation information (i.e., data from micro-batches earlier than the most recent one) a weight, and then gives a higher weight to the most recent correlation information. Then, the total of both becomes the new parameter M. The goal behind this policy is to consider higher the most recent information along the exploration process as opposed to older ones.

Shared Stream. Its focus is on the group of pairs, that were not correlated in the previous micro-batch but share a data stream that was part of a correlated one in the preceding micro-batch. The idea in this policy is that a data stream that is correlated with another one, might be correlated with a third different stream as well. Thus, this policy picks a pair from this group of non-correlated pairs according to *PriCe-DCS* and explores it. Shared Streams starts all those pairs, and then, carries on using *PriCe-DCS*.

X% Probing. This policy explores the first few windows for all the pairs in a round-robin fashion. This is done to set the utility function with actual current values instead of artificial hand-crafted ones. After those few windows, *PriCe-DCS* kicks in and continues the exploration process using the utility function that had its parameters filled with actual data through the probing process.

The rational behind this policy is to take advantage the good properties of *iBRAID-DCS* during the starting phase. In some respect, this policy is a hybrid of *iBRAID-DCS* and *PriCe-DCS*.

P-Alternating. This policy mimics the multilevel queue scheduling, whereby the pairs are explored in a round-robin fashion between two groups. Those are the previously correlated pairs and the non-correlated ones. This is to give the pairs that were not correlated in the previous micro-batch a chance to be persistently processed. It picks a pair from the non-correlated ones and processes it using *PriCe-DCS*, then, it picks a pair from those that were correlated. It does that until the end of the micro batch (i.e., not only the starting phase), and carries on in this fashion by picking the pair from each group of pairs according to the utility function. By doing this, the hope is to alleviate the effect of starvation for those pairs, that were not correlated in the previous micro-batch in a persistent way, for they might exhibit some correlation beyond the starting phase.

4 Experiments and Analysis

In this section we present the evaluation of the *DCS* framework and its algorithms. Furthermore, we evaluate the nine different policies with *PriCe-DCS*, and how each policy addresses different exploration requirements. For consistency, we used the same dataset and settings as in [13].

4.1 Experimental Framework

Algorithms. We compared a baseline algorithm *Random* against our two algorithms *iBRAID* and *PriCe*, along with their *DCS* variants *Random-DCS*, *iBRAID-DCS*, and *PriCe-DCS*. Also, we studied the nine different policies that modify *PriCe-DCS* default behavior.

Testbed. We implemented all the discussed algorithms and policies in C++ 11. We ran the experiments on a computer with 2 Intel CPUs, running at 2.66 GHz, and 96 GB 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 policies in terms of *detection-recall*, *overlapping-recall*, and *diversity*. We also measure the *cost*, which is used to determine the deadlines in our experiments. Those metrics are discussed next.

Cost: This is our efficiency metric. We measured the deadline latency as the number of operations performed to detect correlated pairs of data streams. We used the number of operations as it provides the asymptotic efficiency of the policies compared to one another. This does not depend on factors such as the hardware characteristics and the operating system of the computer, on which the experiments are run, nor the efficiency of the compiler. We examined how the policies meet deadlines and how many correlated pairs they could detect under such a requirement.

Table 1. Experimental parameters

Parameter	Value(s)	Parameter	Value(s)
PCC τ	[0.75, 0.90]	w	8
A	[112, 225, 450]	# data streams	72
I	900 (180 s)	# *micro-batches*	4

Detection-Recall: This is our detection optimization criterion (Eq. 2). It reflects how capable the policy is in detecting correlated pairs out of the total actual correlated pairs. Thus, it is a ratio of the number of detected correlated pairs to the total number of correlated pairs.

Overlapping-Recall: An overlapping pair is a detected correlated pair in a given micro-batch, which was also detected as correlated in the immediately preceding micro-batch. In this metric, we find the ratio of the detected overlapping pairs to the total number of overlapping pairs in a micro-batch. Note that this metric does not apply to the very first micro-batch.

Diversity: We measure how many new pairs (i.e., not seen as a result in the most recent micro-batch) are detected in each micro-batch. This is our exploration *vs* exploitation criterion.

Dataset. *Yahoo Finance Historical Data* [15]*:* The dataset we have used in our experiments consists of 318 data streams. Those reflect the trading of 53 companies on the NYSE for the last 28 years. This gives us a total of 50,403 different pairs to analyze. 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 data stream is about 7,100 tuples. Those tuples are divided into micro-batches.

Experiments. We ran four experiments in total. The first two evaluate our base algorithms, and the latter two assess the ability of our proposed policies to detect and diversify the correlated pairs of data streams using *PriCe-DCS*. We conducted the experiments for two PCC threshold τ's, 75% and 90%, and for three different values of A, 112, 225 and 450. The values of A correspond to the 1/8, 1/4 and 1/2 of the micro-batch interval. The micro-batch interval is set to 900 tuples to simulate an inter-arrival time of 180 seconds, where each tuple is produced each 200 ms. Finally, we experimented with three deadlines corresponding to 25%, 50%, and 75% of the total operations needed to determined all the correlated pairs in a micro-batch, i.e., achieve total *detection-recall*. The experimental parameters are summarized in Table 1. In all experiments, we have divided the dataset into four mutually exclusive groups, and we ran our experiments on all of them, we found that the results are similar. Thus, we reported the results of one of those groups. Moreover, we did pick 10% for the *X% Non-Correlated* as a middle point between the policies *Untouched* and *Alternating*.

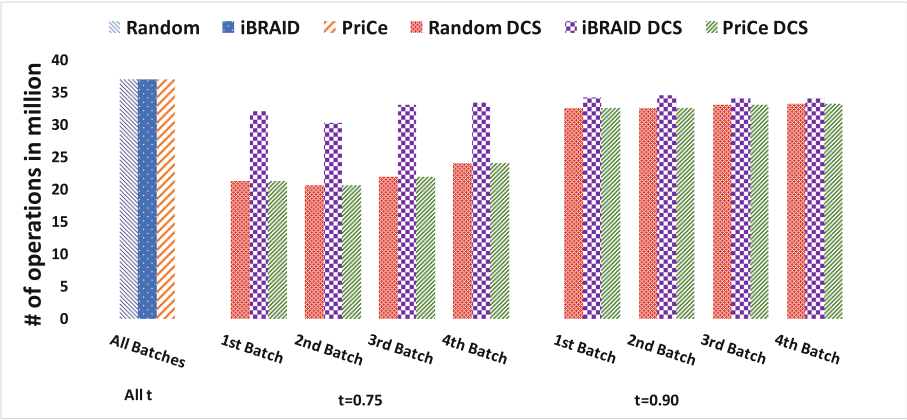

Fig. 1. The cost in number of operations for 4 consecutive micro-batches ($A = 112$).

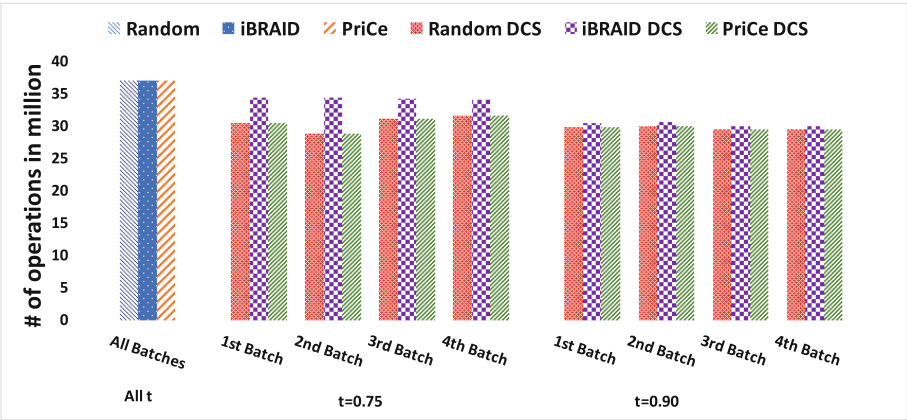

Fig. 2. The cost in number of operations for 4 consecutive micro-batches ($A = 225$).

4.2 Experimental Results

In this section, we present the results of four experiments that we conducted to evaluate the ability of the policies to detect and diversify in real-time the correlated pairs in data streams.

Experiment 1 (Figs. 1, 2 and 3)**.** In our first experiment, we measured the execution *cost* or latency in *number of operations* of each algorithm to detect the specified number A of correlated pairs of sliding windows in four consecutive micro-batches.

As expected, *Random*, *iBRAID*, and *PriCe* have the same number of operations due to exhaustive processing of the pairs. These do not use A for either early termination nor pruning. Thus, with fixed number of data streams, intervals,

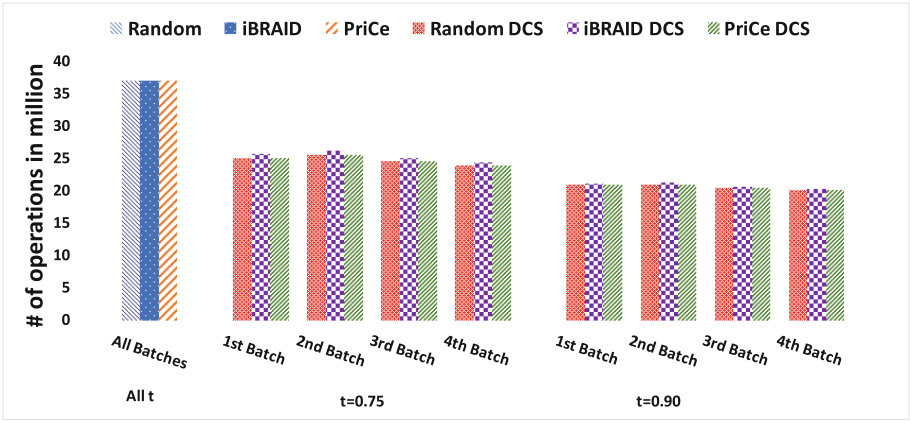

Fig. 3. The cost in number of operations for 4 consecutive micro-batches ($A = 450$).

and window range, the number of operations induced by the three algorithms is identical. They always consume the same amount of operations regardless of the other parameters (i.e., PCC τ and A). The impact of *DCS* mode on all three algorithms is clearly visible in all figures.

In Fig. 1, we notice that the higher the PCC τ, the more operations are executed by the algorithms. This is due to the fewer number of windows that are highly correlated according to the PCC τ. This results in higher latency in detecting them by the algorithms. On the other hand, in case of low PCC τ, we see that *DCS* terminates the analysis process early, as soon as it reaches the A criterion of number of correlated sliding windows. We also observe that *iBRAID-DCS* consistently underperforms the other *DCS* algorithms in latency. This is a consequence of the *iBRAID* scheduling scheme, where it processes all the pairs in a round-robin fashion to avoid starvation. This leads to having a pair pruned at a late stage of the analysis.

In Fig. 2, we notice that *iBRAID-DCS* exhibited higher latency in the cases where PCC $\tau = 0.75$. This is counter intuitive. To clearly state the reason, we observe that the lower the A criterion, the later the pruning will occur in case of no high correlated windows were processed. With that in mind, we say that the pairs of data streams in Fig. 2 in the case of PCC $\tau = 0.75$ has produced high amount of correlated windows, enough to delay the pruning towards the end, but not enough to terminate the analysis early. Therefore, the performance of *iBRAID-DCS* was lower with low PCC τ.

Our last explanation is also supported by our experimental results in Fig. 3, which show clearly that with high A, the *DCS* mode is able to reach a better performance than low A. The reason is that with $A = 450$, if a pair encounters no correlated windows yet, it can be pruned by midway of the analysis process.

Finally, *DCS* mode of operation was able to enhance the performance of the algorithms up to 1.8 times (Fig. 3).

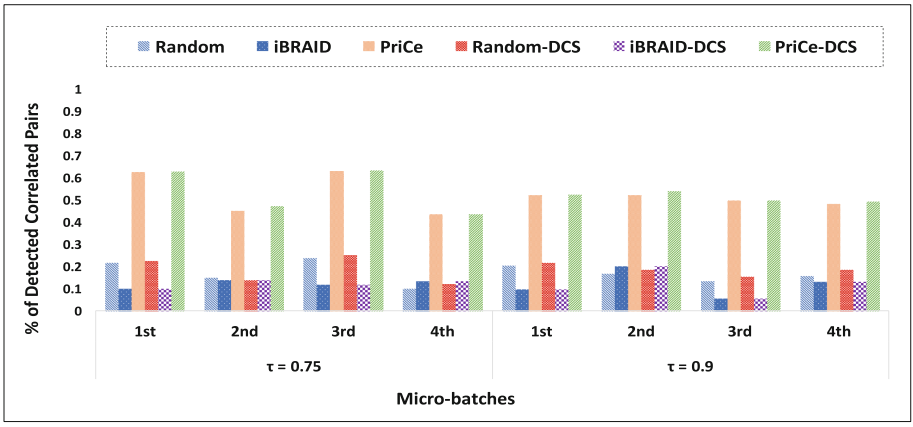

Fig. 4. The % of correlated pairs of streams detected by all algorithms at 25% of the interval I ($A = 112$).

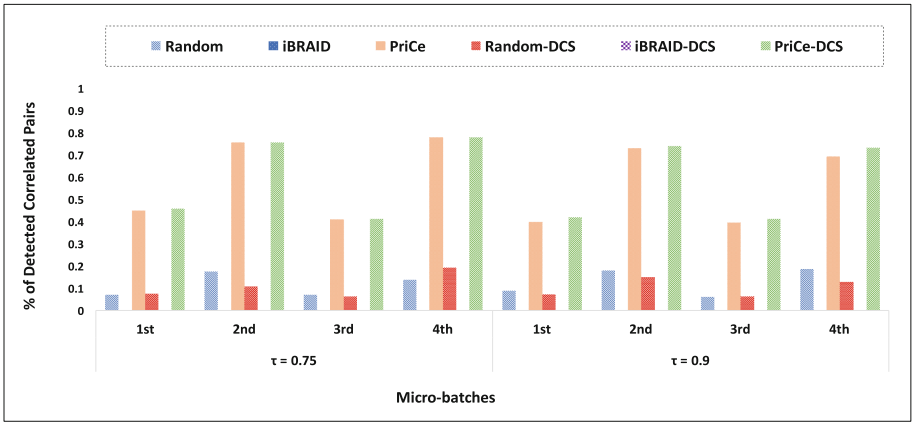

Fig. 5. The % of correlated pairs of streams detected by all algorithms at 25% of the interval I ($A = 225$).

Experiment 2 (Figs. 4, 5, 6, 7 and 8)**.** In this experiment, we studied the *Detection-Recall* of each algorithm with respect to a given deadline. We set the deadline to be 25%, 50%, and 75% of the processing duration of each interval and measured the percentage of the number of correlated pairs each algorithm was able to detect. The results are shown in Figs. 4, 5 and 6 for the deadline 25%, in Fig. 7 for the deadline 50%, and in Fig. 8 for the deadline 75%.

In general, we notice that *DCS* mode of operation in all cases for all algorithms outperforms the original algorithms. This is attributed to the pruning and early termination features of *DCS*, which allow the algorithms to analyze other pairs and detect more correlated data streams.

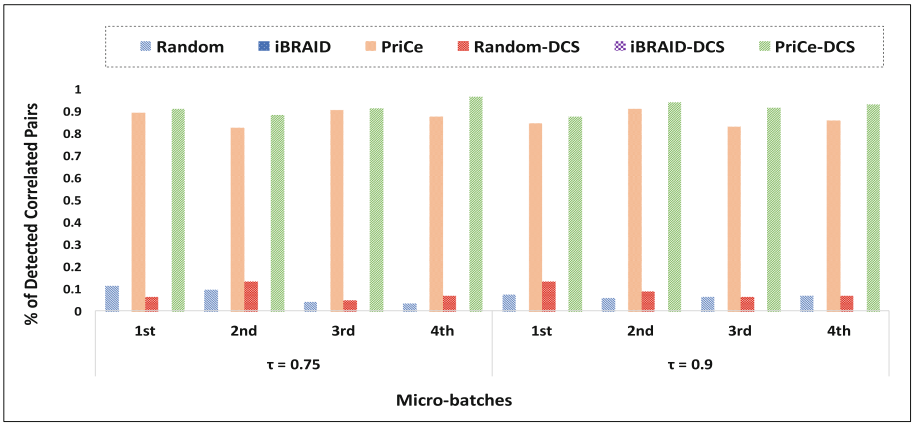

Fig. 6. The % of correlated pairs of streams detected by all algorithms at 25% of the interval I ($A = 450$).

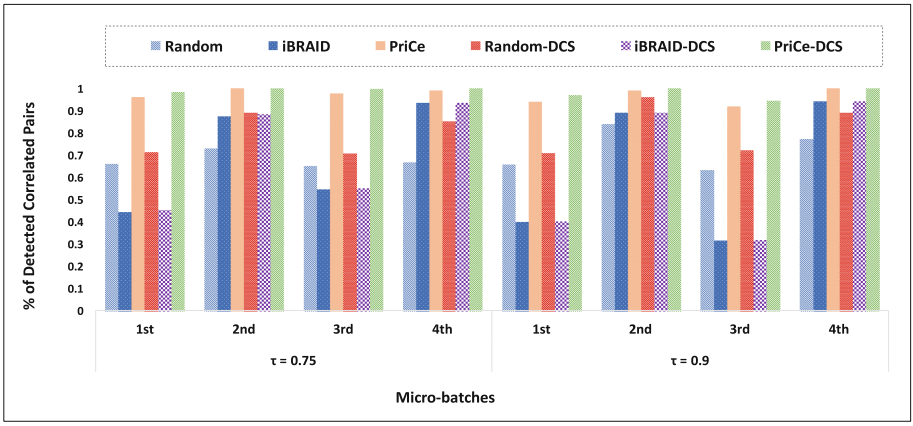

Fig. 7. The % of correlated pairs of streams detected by all algorithms at 50% of the interval I ($A = 225$).

In Fig. 4, we notice that *iBRAID* and *iBRAID-DCS* detected lower percentage of correlated pairs of data streams than *PriCe* and *PriCe-DCS*, while *PriCe* and *PriCe-DCS* have comparable performance. We attribute this to the fact that with a low value of A, we are expecting the required A number of correlated pairs to be detected quickly by *PriCe* and *PriCe-DCS*, while the round-robin scheduling of *iBRAID* and *iBRAID-DCS*, which process all the pairs one pair at a time, is not affected by the value of A. We also notice that *Random* performed slightly better than *Random-DCS* in the 2nd micro-batch, and this is due to the random scheduler nature that picks a pair in a random fashion. In addition, we observe that the average detection percentages for *Random* with $A = 112$ for all PCC τ at the deadline 25% is 18% comparing to 52% for *PriCe*.

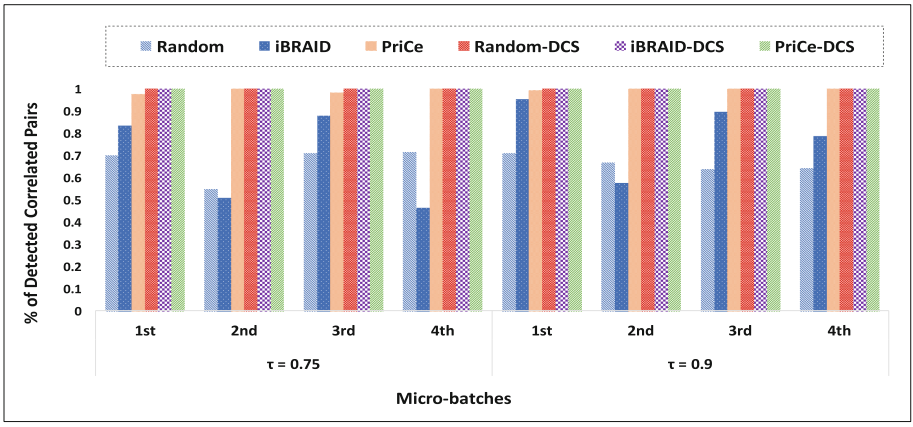

Fig. 8. The % of correlated pairs of streams detected by all algorithms at 75% of the interval I ($A = 450$).

In Fig. 5, we see that *PriCe* scheduling demonstrated the effectiveness of its priority function in capturing more correlated pairs at an early deadline, especially when PCC τ is high. That means, it elects the pairs to explore more intelligently than the other two algorithms. We also notice that with high value of A and early deadline, *iBRAID-DCS* fails to detect any correlated pair of data streams. With respect to *Random* and *Random-DCS*, we observe the same as in Fig. 4, where *Random* performed slightly better than *Random-DCS* due to the random scheduler nature. We also note that the average detection percentage for *Random-DCS* with $A = 225$ for all PCC τ at the deadline 25% is 12% compared to 57% for *PriCe*.

In Fig. 6, the observations are as similar as they are in Fig. 5, however, we realize that *PriCe-DCS* has detected more than 87% the correlated pairs of data streams, and this is due to the priority scheduling of *PriCe* and the aggressive pruning at high A. The *Random-DCS* fails to meet that, since it picks pairs in an unpredictable way, and this delays the analysis duration for each pair, hence, delaying its pruning.

In Fig. 7, we note that *PriCe-DCS* outperforms all the other algorithms. This is for the obvious reason of having more processing time to advance the sliding windows and capture more correlated windows between a pair of data streams. As a result, it reaches the criterion A (=225) of declaring the pair as a correlated one quickly. For $A = 112$ and $A = 450$, our observations are similar as in Fig. 7.

Finally, towards the end of the micro-batch analysis process, we see clearly in Fig. 8 that all the algorithms are detecting pairs with an overall relatively higher percentage than the earlier deadlines, and this is expected with more time to analyze the pairs of data streams. We also observe the effect of pruning clearly on all algorithms under *DCS* mode. This is a result of the A criterion being very high, which leads to early pruning for non promising pairs of data streams.

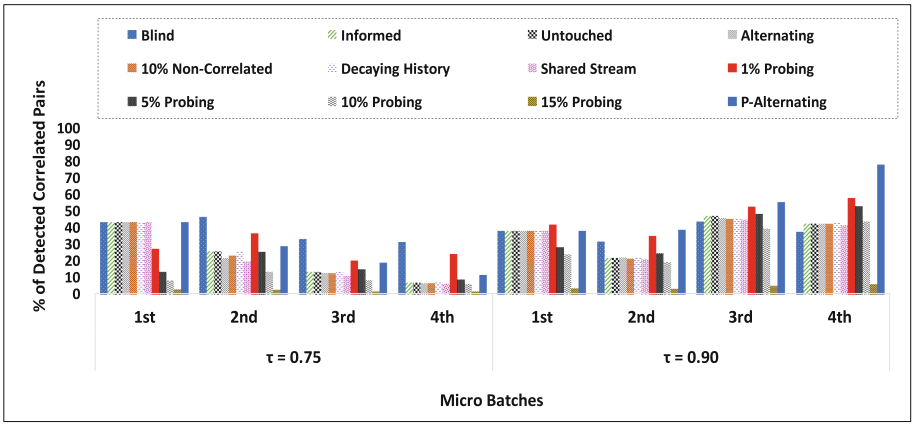

Fig. 9. The % of correlated pairs of streams detected by all policies at 25% of the interval I ($A = 112$).

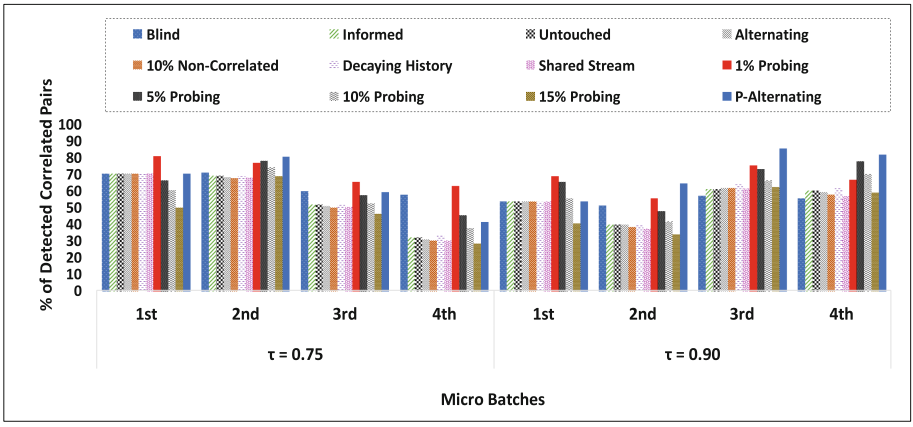

Fig. 10. The % of correlated pairs of streams detected by all policies at 50% of the interval I ($A = 112$).

Thus, all the correlated pairs of data streams were detected earlier than the 75% deadline. The same holds for $A = 112$ and $A = 225$.

Experiment 3 (Figs. 9 and 10)**.** Our previous two experiments show that *PriCe-DCS* exhibits the best performance overall. In our third experiment, we studied the *detection-recall* of each policy with respect to a given deadline. We set the deadline to be 25%, 50%, and 75% of the processing duration of each interval and measured the percentage of the number of correlated pairs each policy was able to detect. We experimented with the values of PCC τ and A, shown in Table 1. All experiments produced similar results; thus, we report the results for the deadline 25% and 50% only.

Figure 9 shows the *detection-recall* for the 25% deadline with $A = 112$. We notice that *PriCe-DCS* with policy *P-Alternating* outperforms the rest with PCC $\tau = 0.90$. On the other hand, the *1% Probing* outperforms the rest when PCC $\tau = 0.75$. This is attributed to the fact that with higher PCC τ values it is highly likely to have fewer correlated windows in a given pair, and vice versa. Having that in mind, with lower PCC τ, exploring the initial 1% of a pair, can lead to an indication of whether the pair is correlated or not. On the other hand, when the pair has scarce correlated windows, it is expected that with persistent exploration of non-correlated pairs, to find new correlation discoveries. Note that the first micro-batch always starts with a *Blind* policy, as there are no historical information about the data streams.

Figure 10 shows the *detection-recall* for the 50% deadline with $A = 112$. It is clear that they exhibit similar behavior overall, except for the 2nd micro-batch, where the policy *P-Alternating* won by a negligible margin over the *X% Probing* policies.

Experiment 4 (Table 2)**.** In this experiment, we studied the impact of historical information on the effectiveness of detecting correlated pairs. This includes the trade-off between exploration and exploitation in the approach for detecting correlated pairs. We use the metrics *detection-recall*, *overlapping-recall*, and *diversity* to illustrate that impact.

In Table 2, we show the results for the algorithm *PriCe-DCS* with *Blind* policy as a baseline. In addition, we show the results of *PriCe-DCS* with *Informed* and *Untouched* as the most exploitative starting phase varieties. This means that they keep detecting the same pairs of data streams that they already have detected. We also show the winners from Experiment 3 (i.e., *1% Probing*, *5% Probing*, and *P-Alternating*).

One can notice that the *P-Alternating PriCe-DCS* achieved the highest *detection-recall* on average for both deadlines. This is expected due to the persistent processing of the non-correlated and correlated pairs in a round-robin fashion. In addition, although *Informed PriCe-DCS* achieved high *overlapping-recall* on average, *P-Alternating* has the highest of all for both deadlines. This is to be expected because *Informed PriCe-DCS* does not alter the parameters of the utility function, instead, it carries all the information of pairs from previous micro-batches as they are, and in the case of *P-Alternating*, the persistent processing of the already correlated pairs manifests itself in the highest *overlapping-recall*.

Finally, the explorative policies (i.e., *Blind*) achieved the highest *diversity* in detecting correlated pairs than the exploitative ones (i.e., *Informed*, *Untouched*, and *P-Alternating*), in fact, they have achieved the lowest *diversity* on average. This can be explained because the exploitative policies have some kind of informative approach on how to analyze the data streams, whether this is from previous micro-batches or some other source. Thus, they will keep exploring the pairs that were already correlated in previous micro-batches.

Take Away: In our first two experiments, we found that *PriCe-DCS* outperformed all other algorithms. In the third, we found that *PriCe-DCS* with *X%*

Table 2. Results of Experiment 4

	Batches	25% Deadline					50% Deadline				
		1st	2nd	3rd	4th	Avg.	1st	2nd	3rd	4th	Avg.
	Full Correlated	429	580	236	234	–	429	580	236	234	–
	Full Overlapped	–	364	215	181	–	–	364	215	181	–
PriCe Blind	Correlated	163	184	103	88	–	231	298	135	130	–
	Overlapped	–	70	67	54	–	–	140	98	84	–
	Unseen Before	163	114	36	34	–	231	158	37	46	–
	Detection-Recall	0.380	0.317	0.436	0.376	0.377	0.538	0.514	0.572	0.556	0.545
	Overlap-Recall	–	0.192	0.312	0.298	**0.267**	–	0.385	0.456	0.464	**0.435**
	Diversity	1	0.620	0.350	0.386	**0.589**	1	0.530	0.274	0.354	**0.540**
PriCe Informed	Correlated	163	126	111	99	–	231	232	144	141	–
	Overlapped	–	81	99	87	–	–	144	136	118	–
	Unseen Before	163	45	12	12	–	231	88	8	23	–
	Detection-Recall	0.380	0.217	0.470	0.423	0.373	0.538	0.400	0.610	0.603	0.538
	Overlap-Recall	–	0.223	0.460	0.481	0.388	–	0.396	0.633	0.652	0.560
	Diversity	1	0.357	0.108	0.121	**0.397**	1	0.379	0.056	0.163	**0.399**
PriCe Untouched	Correlated	163	126	111	99	–	231	232	144	141	–
	Overlapped	–	81	99	87	–	–	144	136	118	–
	Unseen Before	163	45	12	12	–	231	88	8	23	–
	Detection-Recall	0.380	0.217	0.470	0.423	0.373	0.538	0.400	0.610	0.603	0.538
	Overlap-Recall	–	0.223	0.460	0.481	0.388	–	0.396	0.633	0.652	0.560
	Diversity	1	0.357	0.108	0.121	**0.397**	1	0.379	0.056	0.163	**0.399**
PriCe 1% Probing	Correlated	179	203	124	135	–	295	323	178	156	–
	Overlapped	–	123	101	90	–	–	178	129	111	–
	Unseen Before	179	80	23	45	–	295	145	49	45	–
	Detection-Recall	0.417	0.350	0.525	0.577	0.467	0.688	0.557	0.754	0.667	0.666
	Overlap-Recall	–	0.338	0.470	0.497	0.435	–	0.489	0.600	0.613	0.567
	Diversity	1	0.394	0.185	0.333	0.478	1	0.449	0.275	0.288	0.503
PriCe 5% Probing	Correlated	121	143	114	124	–	281	278	173	182	–
	Overlapped	–	110	100	96	–	–	184	145	134	–
	Unseen Before	121	33	14	28	–	281	94	28	48	–
	Detection-Recall	0.282	0.247	0.483	0.530	0.385	0.655	0.479	0.733	0.778	0.661
	Overlap-Recall	–	0.302	0.465	0.530	0.433	–	0.505	0.674	0.740	0.640
	Diversity	1	0.231	0.123	0.226	0.395	1	0.338	0.162	0.264	0.441
PriCe P-Alternating	Correlated	163	224	131	182	–	231	375	202	191	–
	Overlapped	–	108	124	130	–	–	209	187	173	–
	Unseen Before	163	116	7	52	–	231	166	15	18	–
	Detection-Recall	0.380	0.386	0.555	0.778	**0.525**	0.538	0.647	0.856	0.816	**0.714**
	Overlap-Recall	–	0.297	0.577	0.718	**0.531**	–	0.574	0.870	0.956	**0.800**
	Diversity	1	0.518	0.053	0.286	0.464	1	0.443	0.074	0.094	0.403

Probing is the best policy for detecting correlated live data streams for low values of PCC τ. On the contrary, *PriCe-DCS* with *P-Alternating* is the best policy for detecting correlated live data streams for high values of PCC τ. In the last experiment, we found that *P-Alternating*, *Informed* and *Untouched* policies are more suitable for exploiting the space of exploration, and for finding correlated pairs regardless of *diversity*. However, *Blind* policy does detect more diverse pairs across micro-batches along the exploration process.

5 Related Work

The processing of data and fast discovery of correlated subsequences of time series is tackled in two scenarios with respect to the production of data: (1) static, when the data is collected upfront and it forms the search space for finding the correlated subsequences [14,16,17], and (2) dynamic, when the data is processed as it is produced [5–8,18]. The former is beyond the scope of our work. In this section, we focus on the latter, and in particular the state-of-the-art of computationally cheap identification of correlated data streams.

RainMon [5] proposes a 3-stage technique to mine bursty data streams. The received signals are first decomposed, in order to obtain a smoothed representation of the data. In the next stage, called summarization, the received data goes through incremental principal component analysis in an effort to outline the long-term trends in the streams and to identify anomalies, if there are any. In the last stage, named Prediction, the system forecasts trends, relying on the output from the summarization stage. In our work, we do not make predictions and use the data as it is delivered to identify correlated data streams.

A framework for identification of highly correlated pairs of data streams is also presented in StatStream [6]. One of the assumptions of the work is that only an approximation of the PCC is sufficient to identify the pairs of highly correlated data streams. Based on this assumptions, the authors proposed a twofold approach to efficiently identify the pairs of interest. They employ a computationally cheap Discrete Fourier Transformation (DFT) technique to calculate an approximation of the PCC. Furthermore, they proposed an n-dimensional grid structure, which stores the DFT statistics and PCC approximations of each stream, whereby neighboring cells reflect highly correlated streams. This is the springboard for identification of the highly correlated pairs of data streams. However, DFT is known for its poor performance on data streams, which mimic white noise, thereby the required number of DFT coefficients to precisely represent the data streams is high, which induces significant amount of computations. Our work differs in the calculation of PCC. Furthermore, we calculate PCC incrementally over sliding windows, and our framework calculates it precisely for each pair, over each sliding window. Our studies showed that once a pair of data streams is selected as being highly correlated due to a high value of the approximated PCC, a precise calculation of the PCC is required to prove the hypothesis. This operation requires two passes on the data. Similarly to StatStream, our framework supports sliding windows on data streams. We evaluate the possibilities to extend our framework to support landmark windows and damped windows in the future.

StatStream was further improved to handle "uncooperative" data streams in [7], but it still calculates an approximation of PCC only. The proposed technique employs structured random vectors. The experimental results show that the proposed technique outperforms linear scan and the Discrete Fourier Transformations, proposed in StatStream [6]. Our framework, similarly to this work, updates the required statistics in fixed amount of time. However, it differs in PCC calculations, whereby *DCS* calculates PCC of the pairs of data streams

precisely for each sliding window and avoids the need to be calculated later and at a higher cost, once a pair is selected as being "promising".

Detecting similarities between data streams can be achieved through correlation identification techniques. Four different distance measures for similarity of data streams were proposed in [18]: "Autocorrelation Distance (ACD)", "Markovian Distance (MD)", "Local Distance Distribution" and "Probabilistic Local Nearest Neighbor". *ACD* is the version of similarity metric used in our work (PCC), but used for self-correlation, i.e., when a data stream is correlated to itself, whereby one of the windows starts with a lag from the other one. All discussed methods are used to find the first nearest neighbor (1NN) of given data stream only. Our approach, however, identifies all pairs of correlated data streams and is not limited to 1NN only.

Anomaly detection over data streams can also be used as a correlation identification method. A solution is presented in [8] uses the *MD* approach, listed above. Specifically, the presented solution relies on a twofold approach, whereby data streams clustering is combined with Markov chain modeling. The former identifies groups (or clusters) of similar data streams. The latter conveys a possibility for the system to identify anomalies in the data streams in each cluster. In the context of the system, anomalies are considered to be transitions in the Markov chains, which have probability below a certain predefined threshold. Our work may not only be adjusted to identify anomalies, whereby an anomaly is a pair of windows with PCC below a certain threshold, but it also provides analysts with insights about the data. This is done by employing cheap incremental computations, avoiding computationally expensive operations such as building Markovian transition matrices.

6 Conclusions

In this paper, we presented the complete Detection of Correlated Data Streams (DCS) framework, which offers effective solutions for detecting the correlation of data streams within a micro-batch of a fixed time interval for specific analysis requirements. Specifically, we first discussed our novel *PriCe-DCS* algorithm, which combines (1) incremental sliding-window computation of aggregates and (2) intelligent scheduling, driven by a utility function. Then we discussed how the *DCS* framework facilitates the implementation of different policies that tune the utility function in order to meet the exploration and exploitation requirements of analysis tasks.

We implemented and evaluated nine policies that initialize/tune the utility function of *PriCe-DCS*. Other policies could potentially be specified. As opposed to the policies that address explorative objectives, such as *Blind* and *X% Probing*, the policies that address an exploitative objective, such as *P-Alternating*, *Informative*, and *Untouched*, use the result of the preceding micro-batch analyses as part of the initialization of the analysis of the current micro-batch.

Our experimental evaluation using real world dataset showed that the policies that address explorative objectives (i.e., *Blind* and *X% Probing*) detected more

unique correlated data streams. It also revealed that for low PCC τ, *PriCe-DCS* with *X% Probing* outperformed the rest of the policies in terms of *detection-recall*, and for high PCC τ values, *PriCe-DCS* with *P-Alternating* performed the best.

Acknowledgment. This paper was partially supported by NSF under award CBET-1609120, and NIH under Award U01HL137159. The content is solely the responsibility of the authors and does not represent the views of NSF and NIH.

References

1. Kalinin, A., Cetintemel, U., Zdonik, S.: Searchlight: enabling integrated search and exploration over large multidimensional data. PVLDB **8**(10), 1094–1105 (2015)
2. Orang, M., Shiri, N.: Improving performance of similarity measures for uncertain time series using preprocessing techniques. In: ACM SSDBM, pp. 31:1–31:12 (2015)
3. Zacharatou, E.T., Tauheedz, F., Heinis, T., Ailamaki, A.: RUBIK: efficient threshold queries on massive time series. In: ACM SSDBM, pp. 18:1–18:12 (2015)
4. Lee, D., Sim, A., Choi, J., Wu, K.: Novel data reduction based on statistical similarity. In: ACM SSDBM, pp. 21:1–21:12 (2016)
5. Shafer, I., Ren, K., Boddeti, V.N., Abe, Y., Ganger, G.R., Faloutsos, C.: RainMon: an integrated approach to mining bursty timeseries monitoring data. In: ACM SIGKDD, pp. 1158–1166 (2012)
6. Zhu, Y., Shasha, D.: StatStream: statistical monitoring of thousands of data streams in real time. In: VLDB, pp. 358–369 (2002)
7. Cole, R., Shasha, D., Zhao, X.: Fast window correlations over uncooperative time series. In: ACM SIGKDD, pp. 743–749 (2005)
8. Jankov, D., Sikdar, S., Mukherjee, R., Teymourian, K., Jermaine, C.: Real-time high performance anomaly detection over data streams: grand challenge. In: ACM DEBS, pp. 292–297 (2017)
9. Zoumpatianos, K., Idreos, S., Palpanas, T.: Indexing for interactive exploration of big data series. In: ACM SIGMOD, pp. 1555–1566 (2014)
10. Idreos, S., Papaemmanouil, O., Chaudhuri, S.: Overview of data exploration techniques. In: ACM SIGMOD, pp. 277–281 (2015)
11. Feng, K., Cong, G., Bhowmick, S.S., Peng, W.C., Miao, C.: Towards best region search for data exploration. In: ACM SIGMOD, pp. 1055–1070 (2016)
12. Petrov, D., Alseghayer, R., Sharaf, M., Chrysanthis, P.K., Labrinidis, A.: Interactive exploration of correlated time series. In: ACM ExploreDB, pp. 2:1–2:6 (2017)
13. Alseghayer, R., Petrov, D., Chrysanthis, P.K., Sharaf, M., Labrinidis, A.: Detection of highly correlated live data streams. In: BIRTE, pp. 3:1–3:8 (2017)
14. Sakurai, Y., Papadimitriou, S., Faloutsos, C.: BRAID: stream mining through group lag correlations. In: ACM SIGMOD, pp. 599–610 (2005)
15. Yahoo Inc.: Yahoo finance historical data (2016)
16. Kalinin, A., Cetintemel, U., Zdonik, S.: Interactive data exploration using semantic windows. In: ACM SIGMOD, pp. 505–516 (2014)
17. Mueen, A., Nath, S., Liu, J.: Fast approximate correlation for massive time-series data. In: ACM SIGMOD, pp. 171–182 (2010)
18. Mirylenka, K., Dallachiesa, M., Palpanas, T.: Data series similarity using correlation-aware measures. In: ACM SSDBM, pp. 11:1–11:12 (2017)