# Strategies for Detection of Correlated Data Streams

Rakan Alseghayer, Daniel Petrov, Panos K. Chrysanthis
Dept. of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260, USA
{ralseghayer,dpetrov,panos}@cs.pitt.edu

## 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. In this paper we present an effective solution for the analysis of such data streams that is based upon a 3-fold approach that combines (1) incremental sliding-window computation of aggregates, to avoid unnecessary recomputations, (2) intelligent scheduling of computation steps and operations, driven by a utility function within a micro-batch, and (3) an exploration strategy that tunes the utility function. Specifically, we propose eight strategies that explore correlated pairs of live data streams across consecutive micro-batches. Our experimental evaluation on a real dataset shows that some strategies 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.

## CCS CONCEPTS

• **Information systems → Personalization**;

## KEYWORDS

Data Streams, Data Exploration, Correlation, Search, Subsequence

**ACM Reference Format:**
Rakan Alseghayer, Daniel Petrov, Panos K. Chrysanthis. 2018. Strategies for Detection of Correlated Data Streams. In *Proceedings of 5th International Workshop on Exploratory Search in Databases and the Web (ExploreDB 2018 )*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3214708.3214714

## 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, 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 [8]. The correlation can be also used as a source for finding similarity measures faster [12], running threshold queries [15], or reducing the size of the data, but preserving some of its characteristics [9].

**Challenges** Finding correlations in data streams is a challenging task. Current methodologies approach this challenge by employing some prediction techniques [14], Discrete Fourier Transform approximations [2, 16], or using clustering and Markov chain modeling [6]. 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 [3, 4, 17]. 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 [13], [1], we proposed such an algorithm, called *PriCe-DCS*, which detects pairs of correlated data streams within micro-batches of data streams with specific intervals. *PriCe-DCS* uses the Pearson Correlation Coefficient to correlate two windows of data streams and is driven by a utility function (Section 2.3).

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 not consider the detection of correlated pairs across micro-batches independent. For example, in some exploration tasks, the goal is to 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.

In this paper, we study eight different strategies to address the different requirements of exploration tasks. Some of these strategies leverage prior knowledge (i.e., exploit the information of already detected correlated data streams in preceding micro-batches) to steer the detection of correlated pairs of data streams to explore more unique pairs of correlated data streams, or to exploit the identified pairs of data streams and assure the correlation in those pairs. This is achieved by tuning the initial values of the *PriCe-DCS* utility function parameters appropriately.

*ExploreDB 2018 , June 15, 2018, Houston, TX, USA*

ACM ISBN 978-1-4503-5847-7/18/06…$15.00
https://doi.org/10.1145/3214708.3214714

**Contributions** In this paper we make the following contributions:

- We propose eight different strategies that our proposed *PriCe-DCS* algorithm can employ when detecting constitutive micro-batches: *Blind*, *Informed*, *Untounched*, *Alternating*, *X% Non-Correlated*, *Decaying History*, *Shared Stream*, *X% Probing*. These strategies can increase the efficiency when detecting correlated live data streams and/or address the different exploration requirements by exhibiting different *detection-recall, overlapping-recall*, and *diversity* results. (Section 3)
- We experimentally evaluate and compare the behavior of the eight detection strategies. Our results using a real dataset show that our *PriCe-DCS* with *X% Probing* (i.e., 1% Probing and 5% Probing) was able to identify more diverse correlated data streams across micro-batches than the *Informed* strategy. On the other hand, our *PriCe-DCS* with *Informed* strategy was able to assure the existence of correlation in already identified correlated data streams (i.e., high *overlapping-recall*). The other strategies exhibited mixed behavior. (Section 4)

## 2 DCS FRAMEWORK

In this section, we review our *DCS* (Detection of Correlated Streams) mode of operation, introduced in [1], 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 2.1.* A micro-batch is a group of synchronized tuple subsequences over a set of data streams defined by a timestamp *interval I*.

In the system, 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 2.2.* 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 2.3.* 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 $\mu_x$ is the average (or mean) of the values of $x$, $\mu_y$ is the mean of the values of $y$, $\sigma_x$ and $\sigma_y$ are the standard deviations of the values of $x$ and $y$, respectively.

*Definition 2.4.* Two sliding windows of the same range $w$ with a slide of 1 are correlated when the PCC is more than a given threshold $\tau$ (PCC $\geq \tau$).

*Definition 2.5.* A pair of data streams in a micro-batch is correlated when it contains at least $A$ correlated sliding windows with threshold $\tau$.

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 (Real-time Correlation Detection): 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* $\tau$*, 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 Algorithm

*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 $\tau$, and *slidingWindowPosition* is the pair's analysis location in the interval. Recall, $I$ is the interval of the data streams.

## 3 DETECTION STRATEGIES

In this section, we propose eight different detection strategies. 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. For example, in picking the first pair in a new micro-batch to analyze. This question has a major impact on *PriCe-DCS*, since its answer can be used in the initialization of the parameters of its utility function. In fact, the proposed strategies 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, we call that a *Blind* starting phase. In *Blind* starting phase, *PriCe-DCS*'s utility function is initialized to its default values (as discussed in Section 2.3). It is to be noted, however, that the very first micro-batch analysis in all approaches follows the *Blind* approach[1].

**Informed** Here, the utility function is initialized based on the results of the latest micro-batch analysis. We utilize the same parameter values of the correlated pairs used by the immediately previous micro-batch. The rationale behind this strategy is to keep analyzing closely those pairs that already exhibited high correlation in the previous micro-batch, potentially indicating an insight of interest.

**Untouched** Here, we focus on the pairs that were not processed at all 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, we propose to jumpstart such pairs by altering their previous number of correlated windows (i.e., the parameter that reflects this information) to have the value $A$. This will increase their priority, preventing their starvation and giving them another chance to be analyzed in the new micro-batch. The rationale behind this strategy is to allow such pairs another chance, potentially identifying problematic behavior, which remained undetected in the previous micro-batch.

**Alternating** In this strategy, we aim to give the pairs that were not correlated in the previous micro-batch a chance to be explored through a hybrid round-robin fashion. We pick a pair from those and explore it using *PriCe-DCS*. Then, we pick a pair from those who were correlated, and explore it alternately. We do this until we conclude the starting phase (i.e., touched all the pairs at least once), and we carry on later with *PriCe-DCS* according to the utility function. By doing this, we hope to reduce the affect of starvation for those that were not correlated in the previous micro-batch.

**X% Non-Correlated** Here, we try to achieve fairness of exploration through jumpstarting the lowest X% pairs in priority. We pick the pair with the highest priority among those lowest X%, and explore it. We continue until we have jumpstarted all those X% pairs, and reflect that start on their utility function. Consecutively, *PriCe-DCS* carries out the exploration process naturally.

**Decaying History** This strategy regards the significance of the 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 strategy is to consider higher the most recent information along the exploration process as opposed to older ones.

**Shared Stream** In Shared Streams, we have a pool of pairs that were not correlated in the previous micro-batch. However, each pair of those share a data stream that was part of a correlated one in the preceding micro-batch. The idea in this strategy is to say that if we find a data stream that is correlated with another data stream, then it might be correlated with a third, different stream as well. Thus, in this strategy, we pick a pair from this group of non-correlated pairs according to *PriCe-DCS* and explore it. We do this until we start all those pairs, and then we carry on using *PriCe-DCS*.

**X% Probing** In this strategy, we explore the first few windows for all the pairs in a round-robin fashion. We do this 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.

## 4 EXPERIMENTS AND ANALYSIS

In this section we present results from the evaluation of *PriCe-DCS* with the different strategies, and we also study how each strategy addresses different exploration requirements. For consistency, we used the same dataset and settings in [1].

### 4.1 Experimental Framework

**Algorithms** We study the different strategies that modify *PriCe-DCS* default behavior.

**Testbed** We implemented all the discussed strategies 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.

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

**Metrics** We evaluated the performance of the strategies in terms of *detection-recall*, *overlapping-recall*, and *diversity*. We also measure the *cost*, which is used to determine the deadlines in our experiments.

*Detection-Recall*: This is our detection optimization criterion (Eq. 2). It reflects how capable the strategy 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*: We call pairs that were detected in a given micro-batch and also were detected in the preceding micro-batch as an overlapping pair. 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.

*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 strategies 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 strategies meet deadlines and how many correlated pairs they could detect under such a requirement.

**Dataset** *Yahoo Finance Historical Data* [5]: 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 two experiments to measure the cost for all strategies, based upon *PriCe-DCS*. We did it for two PCC threshold $\tau$'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 milliseconds. 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. We also ran an experiment to measure the *detection-recall, overlapping-recall*, and *diversity* for some strategies with PCC threshold $\tau$ = 90% and $A$ = 112, and deadlines 25% and 50% of the total operations. 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 strategies *Untouched* and *Alternating*.

**Table 1: Experimental Parameters**

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

## 4.2 Experimental Results

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

**Experiment 1 (Figure 1)** In this experiment, we studied the *detection-recall* of each strategy 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 strategy was able to detect. We experimented with the values of PCC $\tau$ and $A$, shown in Table 1. All experiments produced similar results; due to limited space we report here only for the deadline 25%, which is the most strict one.

In Figure 1, we show the percentage of detected correlated pairs at the 25% deadline and with $A$=112. We notice that *PriCe-DCS* with strategy *1% Probing* outperformed the rest except in the 4th micro-batch where the *5% Probing* outperformed the rest. This clearly indicates that inducing a little overhead and running round-robin at the beginning of each micro-batch for certain number of windows is the most effective way to initialize the utility function when exploring micro-batches. This result is intuitive as we try to determine at each beginning the behavior of the data streams in the current interval. This way, we distinguish the pairs of data streams that are correlated blindly without discrimination as opposed to, for example *Informed* and *Untouched*, which use the history of the preceding micro-batch.

**Experiment 2 (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 showed the results for the algorithm *PriCe-DCS* with *Blind* strategy as a baseline. In addition, we showed 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 showed the winners from Experiment 1 (i.e., *1% Probing* and *5% Probing*).

We noticed that the *X% Probing PriCe-DCS* achieved the highest *detection-recall* on average for both deadlines. This is expected due to the early sampling of the exploration space. In addition, *Informed PriCe-DCS* achieved the highest *overlapping-recall* on average in the deadline 25%. This is also expected because *Informed PriCe-DCS* does not alter the parameters of the utility function, instead, it

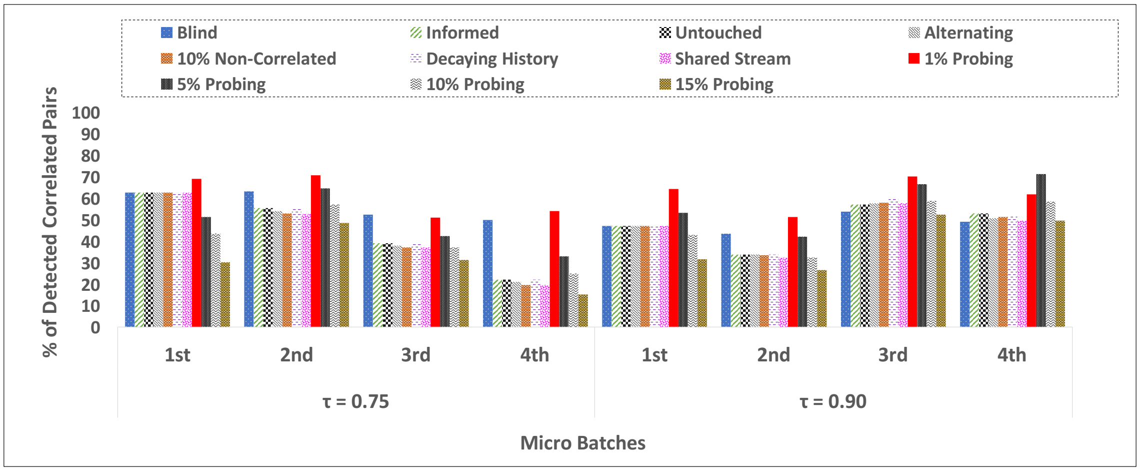

**Figure 1: The % of correlated pairs of streams detected by all starting phases at 25% of the interval $I$ (the correlation criterion $A = 112$).**

carries all the information of pairs from previous micro-batches as they are.

Finally, the explorative strategies (i.e., *X% Probing* and *Blind*) achieved a higher *diversity* in detecting correlated pairs than the exploitative ones (i.e., *Informed* and *Untouched*). This can be explained because the exploitative strategies 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 experiment, we found that *PriCe-DCS* with *X% Probing* is the best strategy for detecting correlated live data streams. In the second experiment, we found that *Informed* and *Untouched* strategies are more suitable for exploiting the space of exploration, and for finding correlated pairs regardless of *diversity*. However, *X% Probing* strategy do 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—dynamic, when the data is processed as it is produced [2, 6, 10, 14, 16], and static, when the data is collected upfront and it forms the search space for finding the correlated subsequences [7, 11]. The latter is beyond the scope of our work. In this section, we discuss the state-of-the-art of computationally cheap identification of correlated data streams.

Detecting similarities between data streams can also be achieved through correlation identification techniques. Four different distance measures for similarity of data streams were proposed in [10], namely, “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 be used as a correlation identification method. A solution is presented in [6] 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

Our novel *PriCe-DCS* algorithm, previously proposed in our Detection of Correlated Streams (DCS) framework, combines (1) incremental sliding-window computation of aggregates, (2) intelligent scheduling, driven by a utility function, and (3) an exploration strategy that tunes the utility function. In this paper, we presented a number of exploration strategies that initialize/tune the utility function of *PriCe-DCS* differently in order to meet the exploration and exploitation requirements of analysis tasks. We studied eight exploration strategies. The *Informative* and *Untouched* strategies that address an exploitative objective use the result of the predeceasing micro-batch analysis as part of the initialization of the analysis within the current micro-batch. On the other hand, the strategies that address explorative objectives (i.e., *X% Probing*) detected more unique correlated data streams. We found out that *PriCe-DCS*, combined with *X% Probing* outperformed the rest of the strategies in terms of *detection-recall*.

**Table 2: Results of Experiment 4**

| | | 25% Deadline | | | | | 50% Deadline | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Batches | 1st | 2nd | 3rd | 4th | Avg. | 1st | 2nd | 3rd | 4th | Avg. |
| | Total Correlated | 429 | 580 | 236 | 234 | – | 429 | 580 | 236 | 234 | – |
| | Total Overlapped | — | 364 | 215 | 181 | – | — | 364 | 215 | 181 | – |
| PriCe Blind | Detected Correlated | 203 | 253 | 127 | 115 | – | 337 | 452 | 192 | 194 | – |
| | Detected Overlapped | — | 110 | 87 | 75 | – | — | 256 | 157 | 133 | – |
| | Unseen Before | 203 | 143 | 40 | 40 | – | 337 | 196 | 35 | 61 | – |
| | Detection-Recall | 0.473 | 0.436 | 0.538 | 0.491 | 0.485 | 0.786 | 0.779 | 0.814 | 0.829 | 0.802 |
| | Overlapping-Recall | — | 0.302 | 0.405 | 0.414 | 0.374 | — | 0.703 | 0.730 | 0.735 | 0.723 |
| | Diversity | 1 | 0.565 | 0.315 | 0.348 | **0.557** | 1 | 0.434 | 0.182 | 0.314 | **0.483** |
| PriCe Informed | Detected Correlated | 203 | 197 | 135 | 124 | – | 337 | 402 | 205 | 199 | – |
| | Detected Overlapped | — | 193 | 135 | 118 | – | — | 295 | 191 | 164 | – |
| | Unseen Before | 203 | 4 | 0 | 6 | – | 337 | 107 | 14 | 35 | – |
| | Detection-Recall | 0.473 | 0.340 | 0.572 | 0.530 | 0.479 | 0.786 | 0.693 | 0.869 | 0.850 | 0.800 |
| | Overlapping-Recall | — | 0.530 | 0.628 | 0.652 | **0.603** | — | 0.810 | 0.888 | 0.906 | 0.868 |
| | Diversity | 1 | 0.020 | 0 | 0.048 | 0.267 | 1 | 0.266 | 0.068 | 0.176 | 0.378 |
| PriCe Untouched | Detected Correlated | 203 | 197 | 135 | 124 | – | 337 | 402 | 204 | 199 | – |
| | Detected Overlapped | — | 193 | 135 | 118 | – | — | 295 | 190 | 164 | – |
| | Unseen Before | 203 | 4 | 0 | 6 | – | 337 | 107 | 14 | 35 | – |
| | Detection-Recall | 0.473 | 0.340 | 0.572 | 0.530 | 0.479 | 0.786 | 0.693 | 0.864 | 0.850 | 0.798 |
| | Overlapping-Recall | — | 0.530 | 0.628 | 0.652 | **0.603** | — | 0.810 | 0.884 | 0.906 | 0.867 |
| | Diversity | 1 | 0.020 | 0 | 0.048 | 0.267 | 1 | 0.266 | 0.069 | 0.176 | 0.378 |
| PriCe 1% Probing | Detected Correlated | 276 | 298 | 166 | 145 | – | 356 | 447 | 219 | 197 | – |
| | Detected Overlapped | — | 167 | 119 | 104 | – | — | 245 | 176 | 147 | – |
| | Unseen Before | 276 | 131 | 47 | 41 | – | 356 | 202 | 43 | 50 | – |
| | Detection-Recall | 0.643 | 0.514 | 0.703 | 0.620 | **0.620** | 0.830 | 0.771 | 0.928 | 0.842 | **0.843** |
| | Overlapping-Recall | — | 0.459 | 0.553 | 0.575 | 0.529 | — | 0.673 | 0.819 | 0.812 | 0.768 |
| | Diversity | 1 | 0.440 | 0.283 | 0.283 | 0.502 | 1 | 0.452 | 0.196 | 0.254 | 0.476 |
| PriCe 5% Probing | Detected Correlated | 229 | 245 | 157 | 167 | – | 396 | 501 | 220 | 223 | — |
| | Detected Overlapped | — | 158 | 138 | 126 | – | — | 326 | 195 | 169 | — |
| | Unseen Before | 229 | 87 | 19 | 41 | – | 396 | 175 | 25 | 54 | — |
| | Detection-Recall | 0.534 | 0.422 | 0.665 | 0.714 | **0.584** | 0.923 | 0.864 | 0.932 | 0.953 | **0.918** |
| | Overlapping-Recall | — | 0.434 | 0.642 | 0.696 | 0.591 | — | 0.896 | 0.907 | 0.934 | 0.912 |
| | Diversity | 1 | 0.355 | 0.121 | 0.246 | 0.431 | 1 | 0.349 | 0.114 | 0.242 | 0.426 |

## 7 ACKNOWLEDGEMENT

We thank Alexandros Labrinidis and Mohamed Sharaf for their valuable feedback on this publication, partially supported by NIH under Award U01HL137159. The content is solely the responsibility of the authors.

## REFERENCES

[1] Rakan Alseghayer, Daniel Petrov, Panos K. Chrysanthis, Mohamed Sharaf, and Alexandros Labrinidis. 2017. Detection of Highly Correlated Live Data Streams *(BIRTE '17)*. 3:1–3:8.

[2] Richard Cole, Dennis Shasha, and Xiaojian Zhao. 2005. Fast Window Correlations over Uncooperative Time Series *(KDD '05)*. 743–749.

[3] 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.

[4] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. 2015. Overview of Data Exploration Techniques *(ACM SIGMOD'15)*. 277–281.

[5] Yahoo Inc. 2016. Yahoo Finance Historical Data. (2016). https://finance.yahoo.com/quote/YHOO/history

[6] Dimitrije Jankov, Sourav Sikdar, Rohan Mukherjee, Kia Teymourian, and Chris Jermaine. 2017. Real-time High Performance Anomaly Detection over Data Streams: Grand Challenge *(DEBS '17)*. 292–297.

[7] Alexander Kalinin, Ugur Cetintemel, and Stan Zdonik. 2014. Interactive Data Exploration Using Semantic Windows *(ACM SIGMOD'14)*. 505–516.

[8] Alexander Kalinin, Ugur Cetintemel, and Stan Zdonik. 2015. Searchlight: Enabling Integrated Search and Exploration over Large Multidimensional Data. *Proc. VLDB Endow.* 8, 10 (jun 2015), 1094–1105.

[9] Dongeun Lee, Alex Sim, Jaesik Choi, and Kesheng Wu. 2016. Novel Data Reduction Based on Statistical Similarity *(SSDBM '16)*. 21:1–21:12.

[10] Katsiaryna Mirylenka, Michele Dallachiesa, and Themis Palpanas. 2017. Data Series Similarity Using Correlation-Aware Measures *(SSDBM '17)*. 11:1–11:12.

[11] Abdullah Mueen, Suman Nath, and Jie Liu. 2010. Fast Approximate Correlation for Massive Time-series Data *(ACM SIGMOD'10)*. 171–182.

[12] Mahsa Orang and Nematollaah Shiri. 2015. Improving Performance of Similarity Measures for Uncertain Time series Using Preprocessing Techniques *(SSDBM '15)*. 31:1–31:12.

[13] Daniel Petrov, Rakan Alseghayer, Mohamed Sharaf, Panos K. Chrysanthis, and Alexandros Labrinidis. 2017. Interactive Exploration of Correlated Time Series *(ExploreDB'17)*. 2:1–2:6.

[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] 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.

[16] Yunyue Zhu and Dennis Shasha. 2002. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time *(VLDB '02)*. 358–369.

[17] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2014. Indexing for Interactive Exploration of Big Data series *(ACM SIGMOD'14)*. 1555–1566.