Multi-parameter streaming outlier detection

Theodoros Toliopoulos
Aristotle University of Thessaloniki
Greece
tatoliop@csd.auth.gr

Anastasios Gounaris
Aristotle University of Thessaloniki
Greece
gounaria@csd.auth.gr

ABSTRACT

Distance-based outlier detection techniques is a wide-spread methodology for anomaly detection. Despite their effectiveness, a main limitation is that they heavily rely on the dataset and the parameters chosen in order to establish the right status of each data point. These parameters typically include, but are not limited to, the neighborhood radius and threshold. In continuous streaming environments, the need for real-time analysis does not permit for an algorithm to be restarted multiple times with different parameters until the right combination is specified. This gives rise to the need for one technique that combines an arbitrary number of parameterizations with the use of minimal yet sufficient computer resources. In this work we both compare the state-of-the-art techniques for handling multiple queries in distance-based outlier detection algorithms and we propose a novel technique for multi-parameter distance-based outlier detection tailored to distributed continuous streaming environments, such as Spark and Flink.

CCS CONCEPTS

• **Information systems → Data stream mining**; • **Computing methodologies → Anomaly detection**; *Massively parallel algorithms.*

KEYWORDS

distance-based outlier detection, Flink, streams

ACM Reference Format:
Theodoros Toliopoulos and Anastasios Gounaris. 2019. Multi-parameter streaming outlier detection. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI '19), October 14–17, 2019, Thessaloniki, Greece.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3350546.3352520

1 INTRODUCTION

Real-time analytics is nowadays playing a key role in applied data science fueled by the recent expansion of data stream sources, such as social networks, online shops and smartphone usage and application logs. Outlier analysis [1] is a key mechanism used for identifying noise or anomalous data points in such data streams. Distance-based outlier detection [6] is a commonly used technique for identifying outliers within data streams or databases because of its simplicity and usefulness for every domain that can translate the difference between data points into a metric distance computation. Based on the distance-based approach, a data point p is an outlier if it has less than k neighbors within distance R. Many distance-based algorithms have been proposed in the literature; some of them are tailored to data streams [2, 5, 7, 11] while others are applicable only to static data, such as traditional databases [4].

The main drawback of distance-based techniques is the dependence on the input parameters. Even when a domain expert is involved, data streams keep evolving so that a specific good parameter setting at one point becomes obsolete some time later. This yields the need for more flexible techniques that can take into account multiple user queries, i.e., outlier detection queries with different parameterizations [3, 7, 13]. In a banking system, for example, each fraud type has a different pattern that needs to be detected in order to raise an alarm. Using multiple-parameters in the same algorithm can detect all of these different frauds (outliers) without stressing the system. In this work, multi-query and multi-parameter outlier detection is used interchangeably.

In addition, the data streams grow larger in the volume and speed that data is produced. This gives rise to the need for scalable analytics. Regarding distance-based outlier detection, the solutions for distributed settings are rather limited. They cover distributed spatial databases [4] and there is also a proposal for massively parallel stream solutions in [8].

In this work we focus on transferring the state-of-the-art multi-query distance-based outlier detection algorithms, presented in the next sections, in a distributed streaming setting using the Flink[1] framework which is used for massively parallel stream analytics. We make two contributions. First, we provide parallel flavors for the aforementioned algorithms by combining them with a partitioning technique and present a comparative study. Second, we propose a novel hybrid technique that is shown to be as efficient as the most efficient technique to date. Finally, our implementations are provided as an open source module so that third parties can both repeat all the experiments and build on our techniques.[2]

The remainder of this work is structured as follows. Section 2 contains the necessary information on stream and outlier detection semantics along with the main existing techniques. Section 3 introduces the first part of the multi-query space, where only the application parameters (i.e., the radius and the threshold) change while Section 4 contains the complete solution taking into account both the application and the windowing (i.e., size of window and slide) parameters. Section 5 shows the performance evaluation results using a real cluster and finally a discussion about future extensions concludes our work in Section 6.

WI '19, October 14–17, 2019, Thessaloniki, Greece

ACM ISBN 978-1-4503-6934-3/19/10…$15.00
https://doi.org/10.1145/3350546.3352520

[1] https://flink.apache.org/
[2] The code repository is at https://github.com/tatoliop/parallel-streaming-outlier-detection

2 BACKGROUND

This section consists of four parts. The first part focuses on streams and techniques that allow real-time analytics while the second part deals with definitions of the distance-based techniques. The third part of this section presents a brief reference to the state-of-the-art algorithms while the fourth one presents the common principles that are followed in their implementations along with the distributed algorithm that serves as the basis.

2.1 Streaming semantics

A data stream is defined as an infinite sequence of data points produced continuously. The data stream contains data points; each data point p is associated with a timestamp that represents its arrival time, $p.t$. Windowing is a commonly used technique in real-time analytics that helps to break the stream into smaller finite sets that are easier to be processed. A window can have a fixed size W based on a time period or on the number of data points, called time-based and count-based, respectively. Each consecutive window of a stream can either be overlapping (*sliding*) with the previous one or non-overlapping (*tumbling*) from all of the others. In the sliding windows, the overlap is defined according to either a time period or the number of data points that enter/leave the window and is called the slide size S.

A time window is annotated by the starting and ending timestamps $W.start$ and $W.end$ respectively which represent the range of time that the data point's arrival fall in (e.g. $W.start \leq p.t < W.end$). Any data point with $p.t < W.start$ is considered expired from the current window. In a sliding time window the starting and ending timestamps are moved every S time units. When a time window with $W_{range} = [W.start, W.end)$ slides by S time units, the new window has $W_{range} = [W.start + S, W.end + S)$. According to the definitions above, tumbling windows are a subset of the sliding ones where the slide size $S = W$. In this work we focus on sliding windows. Also, the techniques that are proposed are typically applicable to both time- and count-based windows.

2.2 Distance-based outlier detection definitions

The distance-based outlier detection problem is defined as follows.

Definition 2.1. Given a set of objects P and the application parameters R and k, report all of the objects $p \in P$ that have less than k neighbors ($p.nn < k$), where an object p' is assessed as a neighbor of p if $dist(p, p') \leq R$.

The above definition implies that each object is given a single label (being outlier or not). Adding the window semantics to the definition 2.1, the distance-based outlier detection problem for a sliding window setting is defined as follows.

Definition 2.2. Given a window with size W and slide size S on a data stream O and the application parameters R and k, for each window slide containing a set of objects $p \in P$ with $W.start \leq p.t < W.end$, report all of the objects $p \in P$ that have less than k neighbors ($p.nn < k$), where an object p' is assessed as a neighbor of p if $dist(p, p') \leq R$.

The main implication of the extensions in the definition above is that a single data point can change its status as many times as the number of window slides that take place while this point belongs to the window.

2.3 Multi-query outlier detection algorithms

The state-of-the-art distance-based algorithms that support multiple query input are *AMCOD* [7], *SOP* [3] and *PSOD* [13].

The AMCOD algorithm is an extension of the MCOD one [7], in which the use of micro-clusters drastically cuts down the range queries needed in order to find the neighbors for each data point. The algorithm supports only queries with multiple application parameters (R and k) without providing a solution for multiple windowing parameters.

The SOP algorithm creates one skyband query in order to find the sufficient yet minimal set of neighbors for each data point providing multiple query support with both application and windowing parameters.

The third algorithm from the literature is the PSOD algorithm that can handle multiple queries that include different application and windowing parameters without the use of a skyband query.

In this work, we also propose a new algorithm, coined as *pMCSKY*, which takes advantage of both the micro-clusters of the AMCOD and the slide processing with the skyband query of the SOP algorithm.

2.4 Data partitioning

In this work, the multi-query algorithms are implemented from scratch in a distributed setting. The implementation has a common feature for all 4 of the algorithms, namely the data partitioning. The partitioning technique distributes the data points to different workers where each worker runs a localized instance of the respective outlier detection algorithm, reporting the local outliers. The final phase of the distributed implementation combines the output of each worker in order to report the global outliers for the current window.

The data partitioning technique used in all of the cases is borrowed from the pMCOD algorithm in [8]. pMCOD is another extension of MCOD and the state-of-the-art distributed algorithm for continuous outlier detection in data streams with the drawback of supporting only 1 query per instance. It combines the techniques from MCOD (such as the micro-clusters explained in section 3.2) with the data partitioning technique for fast real-time outlier detection and reporting. The partitioning technique uses a VP-tree [12] in order to partition the metric space into cells. Full details and results, including the impact of number of machines on performance are in [9].

3 SUPPORTING MULTIPLE R AND k COMBINATIONS

In this section, we present how to support a wide parameter space when the R and k parameters vary using the four algorithms that were mentioned previously. We start by explaining the $R - k$ query space and the way the multiple parameters are handled in all of the outlier detection algorithms of this work; we then continue by explaining the differences between the algorithms along with their unique features.

R / k	k_1	k_2	k_3
r_1	Status assessed as inlier (1)	Status assessed as outlier (2)	Auto assess as outlier due to (2)
r_2	Auto assess as inlier due to (1)	Status assessed as outlier (3)	Auto assess as outlier due to (3)
r_3	Auto assess as inlier due to (1)	Status assessed as inlier (4)	Status assessed as inlier (5)
r_4	Auto assess as inlier due to (1)	Auto assess as inlier due to (4)	Auto assess as inlier due to (5)

Table 1: Outlier status assessment

3.1 The $R-k$ parameter space

We explore the 2-dimensional parameter space $R-k$ which means that the queries are created from the combinations of the different R and k values from the user input, i.e. if the input has 4 different k values and 5 different R values, the total number of queries is $5*4=20$.

While the metadata stored depend on the algorithm, detecting the outlierness or inlierness of each data point shares the same technique between all four solutions and is based on the following two definitions [13].

Definition 3.1. Given a 2-dimensional parameter space with $R=\{r_1, r_2, ..., r_n\}$ and $k=\{k_1, k_2, ..., k_m\}$, if a data point p is an inlier for a combination (r_i, k_y) with $1 \le i \le n,\ 1 \le y \le m$, then p is an inlier for every combination $(r_{i'}, k_y)$ with $r_{i'} \ge r_i$.

Definition 3.2. Given a 2-dimensional parameter space with $R=\{r_1, r_2, ..., r_n\}$ and $k=\{k_1, k_2, ..., k_m\}$, if a data point p is an outlier for a combination (r_i, k_y) with $1 \le i \le n,\ 1 \le y \le m$, then p is an outlier for every combination $(r_i, k_{y'})$ with $k_{y'} \ge k_y$.

All the $R-k$ combinations can be represented in a 2-dimensional array with each row being a R value and each column being a k value, both in ascending order. Starting from the top left cell where $r_1 = min(R)$ and $k_1 = min(k)$, a data point p is assessed by counting its neighbors in the respective range. If p is an inlier then, based on definition 3.1, it is an inlier for the remaining cells of the same column and the process continues with the first cell in the next column. On the other hand, if p is an outlier, based on definition 3.2, it is an outlier for the remaining cells of the same row and the process continues with the first cell in the next row.

Table 1 shows an example of such a representation along with the steps for a data point's p status assessment. Starting with the cell $[r_1, k_1]$, assume that the process determines that p is an inlier and thus it can immediately update the whole column $[k_1]$ with the inlier status. The process continues with the next combination of $[r_1, k_2]$ where the data point is an outlier, which means that the final part of the row r_1 (in this example the $[r_1, k_3]$ cell) is updated with the outlier status. The same process continues with the next empty cell $[r_2, k_2]$ until all of the table's cells are updated. Overall, twelve combinations are examined with only five of them to be explicitly considered.

3.2 AMCOD

AMCOD is a variation of the MCOD algorithm [7] to support multiple queries. MCOD uses the notion of micro-clusters in order to significantly decrease the number of range queries needed to establish the status of a data point. A micro-cluster has a diameter of R and contains $k+1$ data points, which means that all of these points are inliers (as they have at least k neighbors in the R range) and there is no need to store further metadata. The rest of the data points are stored in a structure called PD along with their neighbors. The algorithm only needs to evaluate the status of the points in PD at the end of each slide.

AMCOD adopts all of MCOD's structures but changes the diameter of the micro-clusters to R_{min}, the minimum value of the R parameter values, and the number of data points that are needed for the creation to $k_{max}+1$. This means that all of the data points in a cluster are inliers in all of the parameter queries. For the rest of the data points that belong to the PD set, the metadata needed to establish the status are the preceding and the succeeding neighbors along with their distance from the source point. A data point can be established as a safe inlier if it has at least k_{max} succeeding neighbors in R_{min} range.

A sketch of the algorithm is as follows. When a new data point arrives, a range query is issued on the centers of the micro-clusters. If it is within radius of a cluster, it is inserted in the cluster's list of points. Otherwise, a new range query is issued on the points of PD. All the neighbors are inserted into the preceding set and the new data point is inserted in the neighbor's succeeding list along with the distance value. If the conditions for a new cluster are met, then the data point becomes the center of the cluster and the neighbors are added to the cluster's list. Otherwise, the point is inserted into PD and the set of preceding neighbors pass through the (k-1)-skyband query, as shown in Algorithm 1, in order to store the minimal and sufficient number of neighbors. When a data point expires, it is removed from either the PD or the cluster it belongs. If the cluster's number of points falls bellow $k_{max}+1$ then it is disbanded and the remaining alive data points are re-inserted as new points (without updating their neighbors again).

Algorithm 1 AMCOD_skyband (computation of the skyband for a point $p \in PD$)

Require: A list of preceding neighbors in ascending order of distance

```
minimum_neighbors ← List()
for i = 1 to neighbors.size do
    counter ← 0
    for y = 1 to i do
        if neighbors(y).timestamp > neighbors(i).timestamp then
            counter = counter + 1
        end if
    end for
    if counter ≤ k then
        minimum_neighbors ← add(neighbors(i))
    end if
end for
```

3.3 SOP

SOP is a framework that can handle multiple parameter queries while processing each data point only once by transforming all

Algorithm 2 SOP_skyband (called for each neighbor found)

Require: A data point p with distance value d
1: $layer \leftarrow LSky.getLayer(d)$
2: counter $\leftarrow 0$
3: **for** $i = 1$ to $layer$ **do**
4: $counter = counter + LSky.layerCount(i)$
5: **end for**
6: **if** $counter \leq k - 1$ **then**
7: $LSky.insert(p)$
8: **return** true
9: **else**
10: **return** false
11: **end if**

of the queries into a single skyline computation. It also uses a k-skyband query to find the minimal and sufficient neighbors for each new data point and update the neighbor set of an old point.

Each data point of the SOP algorithm has a data structure called *LSky* where the neighbors with their distance values are stored. *LSky* is a layered structure where each layer represents a R value from the input parameters (in ascending order) and the points stored in a layer have a distance value $distance \leq R_{layer}$, i.e. in the first layer belong the neighbors with $distance \leq R_1 = R_{min}$ and in the second layer the points with $R_1 < distance \leq R_2$.

SOP uses the k-skyband query for every data point, either new or old, taking into account the chronological order of the points. For the new points, the query finds the minimal and sufficient neighbors for the point from all previous points whilst, for existing points, the query updates the neighbors from the already found ones plus the new slide's data points.

A sketch of the algorithm's steps are as follows. When a new data point arrives, a range query is issued on all of the previous data points in reverse chronological order (starting from the newest to the oldest). For each neighbor found, the k-skyband query (Algorithm 2) is called and the neighbor is either inserted into the point's *LSky* or discarded. The operation stops when a neighbor with $distance \leq R_{min}$ is not inserted in the structure, which means that the minimal set has been found. On the other hand, for each existing (old) data point, a range query is issued on the new slide's data points in reverse chronological order. For each neighbor and each point in the already created *LSky* structure, the k-skyband query is called to update the set.

3.4 PSOD

PSOD is a framework that can handle queries from the 2-dimensional parameter space without the use of a skyband query. The work on this parameter space is a precursor for the 4-dimensional space explained in Section 4, where the solution is complete.

The algorithm is based on the definitions 3.1 and 3.2 in order to assess the status of a data point fast. It does not use a skyband query in order to find the minimal and sufficient set of neighbors for each data point; however the range queries stop when a data point is established as a safe inlier or an unsafe inlier. It uses a data structure similar to *LSky* to store the neighbors based on their distance to the source point p and the different R values, called $p.NT$. For each combination (r_i,k_y), the algorithm only needs to sum up the number of neighbors from each table cell up to i in order to find the total neighbors for p.

A sketch of the algorithm is as follows. For each new data point p, a range query is issued on the data points of the window. The $p.NT$ structure is updated for every neighbor o found. If the o is not a safe inlier, $o.NT$ is also updated with p. In our implementation, we use the previously explained *LSky* structure in order to store the neighbors since it is similarly implemented with the NT table that the PSOD framework uses for the R-k parameter space.

3.5 pMCSKY

pMCSKY (which stands for **p**arallel **M**icro-**C**luster and **Sky**band-based solution) is a novel combination of the MCOD's micro-clusters and PD set along with the SOP neighbor structure and skyband query (Algorithm 2).

As explained in [10] and [8], the micro-clusters of MCOD is a key tool for decreasing the range queries for a new data point and the PD set helps to establish its status in a faster time. AMCOD uses this tool but its main drawback is its data structures: it uses a preceding neighbors set, which has passed through the skyband query 1, and a set for all succeeding neighbors without trying to compress their size (unless the point becomes a safe inlier). In the pMCSKY algorithm, the diameter of a micro-cluster and the number of points it needs to be formed is the same as the AMCOD algorithm, R_{min} and $k_{max} + 1$ respectively.

As already explained, the SOP algorithm has a data structure (*LSky*) that contains the minimal yet sufficient set of the total neighbors of a data point, which includes both preceding and succeeding neighbors. This is employed in pMCSKY as well. The structure's main advantages is the low storage space it needs along with the fast count of the neighbors for each R value. Finally the k-skyband query of the algorithm takes into consideration the stream's chronological order and the *LSky* structure in order to quickly assess if a neighbor belongs to the minimal set or not.

Algorithm 1 shows the main body of pMCSKY which is explained as follows. When a new data point p arrives, a range query is issued on the centers of the micro-clusters. If it is within radius of a cluster, it is inserted in the cluster's list of points. Otherwise a new range query is issued on the data points in reverse chronological order. Each neighbor passes through the k-skyband query to assess whether it belongs to the minimal set or not and is either inserted into the $p.LSky$ structure or not, respectively. If the conditions for a new cluster are met, then the data point becomes the center of the cluster and the neighbors are added to the cluster's list and removed from the PD set. Otherwise p is inserted into PD and its status as an inlier, outlier or safe inlier is established. For each existing (old) data point o, a range query is issued on the new slide's data points in reverse chronological order. For each neighbor and each point in the already created $o.LSky$ structure the k-skyband query is called to update the set. When a data point expires, it is removed from either the PD or the cluster it belongs. If the cluster's number of points falls bellow $k_{max} + 1$, then it is disbanded and the remaining alive data points are re-inserted as new points.

Algorithm 3 PMCSky

Require: A data point p, the micro-clusters MC and the list of data points $data\_list$

```
skyband_list = p.LSky
if p.LSky = null AND p.mc = 0 then
  for i = 1 to MC.size do
    distance ← calculate(p,MC_i)
    if distance ≤ R_min/2 then
      MC.insert(p)
      p.mc ← MC.id
    end if
  end for
  if p.mc = 0 then
    skyband_list ← data_list
  end if
else
  skyband_list ← skyband_list + data_list.new_slide_points
end if
for i = 1 to skyband_list.size do
  distance ← calculate(p,skyband_list(i))
  inserted = SOP_skyband(skyband_list(i), distance)
  if inserted = false AND distance ≤ R_min/2 then
    Break FOR
  end if
end for
```

4 SUPPORTING MULTIPLE R, k, W AND S COMBINATIONS

In this section, we deal with a broader parameter space, where both the application (R and k) and the window (W and S) parameters vary along with the three algorithms that were implemented for that case. Same as the previous section, we start by explaining the query space and the way it is handled in all of the outlier detection algorithms of this work and continue by explaining the differences and issues between the algorithms during their implementation.

4.1 The $R-k-W-S$ parameter space

We explore the 4-dimensional parameter space $R-k-W-S$, which means that the queries are created from the combinations of the different parameter values from the user input including windowing parameters (window size W and slide size S). The total number of queries can be computed by the product of the numbers of distinct parameters, similarly to the description in Section 3.1.

Expanding the representation of 3.1, we use a 3-dimensional array to indicate the number of outliers for each combination. Each row represents a R value, each column a k value and the third dimension represents the W values. Notice that there is no S representation because we use the greatest common divisor of all the S values in order to find the greatest slide size that can be employed for each of the input combinations. Each algorithm presented below distinguish itself in the way it reports the outliers for each given slide size value.

Another extension in the 4-dimensional case includes the Definition 4.1 stemming from Definition 3.1. With this extension, if a data point p is assessed as an inlier in cell $[r_i, k_y, w_a]$, we can automatically establish its status as in inlier in all cells $[r_{i'}, k_y, w_{a'}]$ with $i' \geq i$ and $a' \geq a$.

Definition 4.1. Given a 3-dimensional parameter space with $R = \{r_1, r_2, ..., r_n\}$, $k = \{k_1, k_2, ..., k_m\}$ and $W = \{w_1, w_2, ..., w_z\}$ if a data point p is an inlier for a combination (r_i, k_y, w_a) with $1 \leq i \leq n$, $1 \leq y \leq m$, $1 \leq a \leq z$, then p is an inlier for every combination $(r_{i'}, k_y, w_{a'})$ with $r_{i'} \geq r_i$ and $w_{a'} \geq w_a$.

4.2 Algorithms

The parameter space $R-k-W-S$ is similar to the $R-k$ space in the sense that the algorithms only need minor changes in order to function properly for all queries. To begin, we chose to use the SOP, PSOD and the novel algorithm pMCSKY for these queries, since AMCOD only works with the $R-k$ space and according to Section 5 is significantly inferior to the other three.

The first step is the handling of parameter W. When W varies, the algorithms need to hold the neighbors of each data point for each window size. This creates a big overhead with regards to the required memory of the system as the list of neighbors in smaller windows is a subset of the bigger windows. In order to eliminate the unnecessary neighbor saving, all three of the algorithms create one window with $W = W_{max}$. This allows to save all of the neighbors needed to establish the status of the data point in the biggest window, which is the superset of all the window sizes. In addition, all three algorithms give more weight to the younger neighbors (the ones that are going to expire later in the window).

Because pMCSKY uses micro-clusters, a further improvement needs to be made in order to function properly. If a micro-cluster is formed by the data points of the window W_{max}, information about a younger point's status can be concealed. Figure 1 shows an example of a situation, where the micro-cluster is formed from all of the data points in the window. Data point $p1$ seems to belong to the micro-cluster and its status is automatically assessed as an inlier. However in the small window, it does not have enough neighbors and thus it belongs to this window's outliers. With this in mind, the pMCSKY algorithm only creates micro-clusters formed by data points in the window with $W = W_{min}$.

The second step is concerned with the parameter S. When S varies, the algorithms need to report outliers at different times for each query. All the algorithms use the greatest common divisor S_{gcd} of the S values in order to establish a common S value that can provide the necessary computations for all of the input values. SOP takes the S_{gcd} and reports the outliers for each S value when enough slides have passed. PSOD further processes S_{gcd} and computes S^{var}, which is a list of the slides that the outliers needs to be reported. For each S_i^{var}, the algorithm reports the outliers of the respective S value. For the pMCSKY algorithm, we have chosen to implement the SOP algorithm's reasoning.

5 EXPERIMENTAL EVALUATION

The experimental setting is as follows. All of the experiments are done using a cluster of 3 machines so that the algorithms are tested in a distributed setting. The 3 machines have similar resources and more specifically, the first one has a 6-core/12-thread CPU with 64GB of RAM, the second one has a 8-core/8-thread CPU with 32GB of RAM and the third machine has 4-cores/8-threads with 32

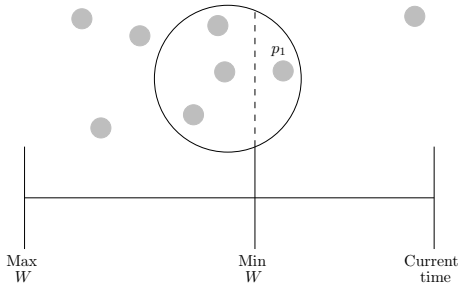

Figure 1: Micro-clusters in the whole window

GB RAM. All of the machines are connected through an Ethernet connection of 1 Gbps speed.

We have used two real-world datasets. The first one is named Stock[3] and has one dimension while the second one is named TAO[4] and has three dimensions. We read both datasets as streams and we divide the experimentation using two use cases, when we vary R and k only, and when we vary all parameters. For the first use case, termed as $R-k$, all four of the algorithms discussed in the previous sections have been tested using both datasets and with different window sizes and slide percentages in order to have as much coverage regarding different scenarios as possible. In these experiments the times presented represent the average time per slide for each algorithm, aggregated over 200 slides. For the second use case, termed as $R-k-W-S$, the three algorithms were tested using a variety of application and windowing parameters with the time presented representing the average total run time of each algorithm's instance; since slides differ, it is meaningless to present average time per slide and also, we need to resort to finite datasets for comparison. Note that the accuracy of the algorithms is always 100% since all of the techniques are exact.

Apart from the techniques discussed, we also present an example of independent pMCOD instances running in parallel emulating a naive multi-query algorithm in order to compare the difference in running times between many instances of the single-query algorithm versus one instance of the multi-query algorithms. The choice of the pMCOD algorithm derives from the fact that all 4 implementations presented in this work have the data partitioning technique of this algorithm as their basis. Also pMCOD is the state-of-the-art distributed outlier detection algorithm for continuous streams [8].

Finally, every plot's y axis on the $R-k$ use case represents the performance using a log scale in order for the results to be shown more clearly since two of the algorithms have very similar running times that differentiate in milliseconds. Another issue of attention is that the R and k application parameters have been chosen so that the percentage of outliers for each dataset is around 1%.

5.1 $R-k$ use case

In this use case, we run two types of experiments with 16 (4 R values combined with 4 k values) and 100 (10 R values combined with 10 k values) queries, respectively. The window size is either

[3]available from https://wrds-web.wharton.upenn.edu/wrds
[4]Available from http://www.pmel.noaa.gov

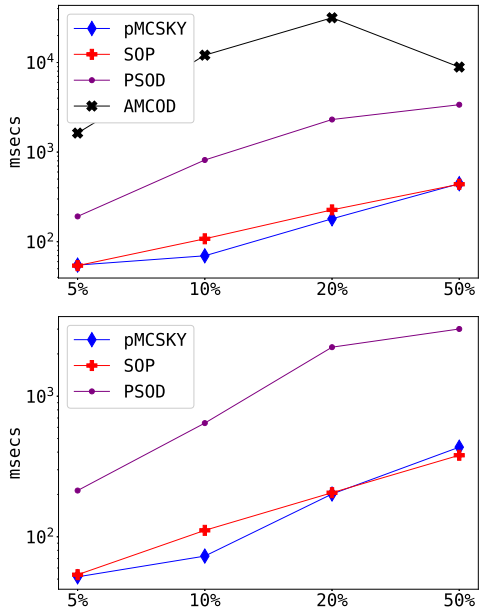

Figure 2: Experiments using Stock with $W = 10k$, 16 queries (top) and 100 queries (bottom)

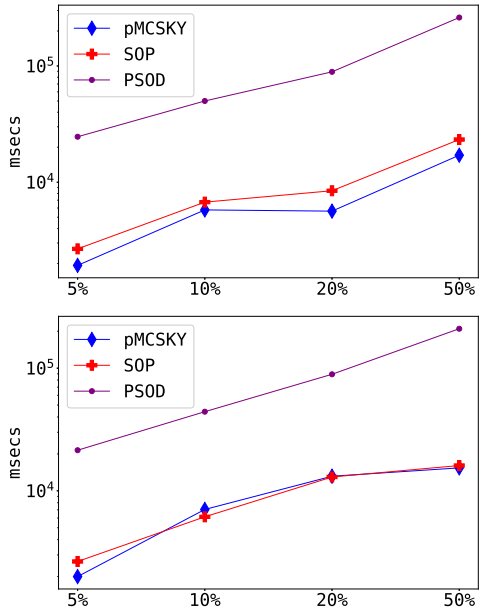

Figure 3: Experiments using Stock with $W = 100k$, 16 queries (top) and 100 queries (bottom)

10K or 100K. For the slide size, we have experimented with $S = \{5\%, 10\%, 20\%, 50\%\}$ of the window size.

Figure 2 shows the results of the experiments using the 4 algorithms (AMCOD, pMCSKY, SOP and PSOD) on the 1-dimensional

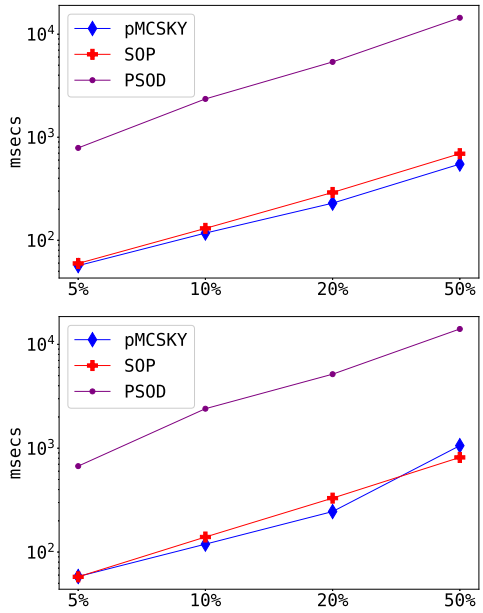

Figure 4: Experiments using TAO with $W = 10k$, 16 queries (top) and 100 queries (bottom)

dataset with $W = 10k$. In this experiment, it can be seen that AMCOD is clearly inferior by at least an order of magnitude and is not tested any further. This is due to the fact that the algorithm depends solely on the micro-clusters. The skyband query is used only on the preceding neighbors after they are found through a range query on the whole window whilst the list with the succeeding neighbors is increasing as the window progresses. This not only has a big effect on the memory usage but on the running times as well because of the full-scale range queries. On the other hand, the micro-clusters are not formed regularly with most of the data points ending up on the PD set because of the R_{min} and k_{max} restrictions.

Figure 3 refers to the same setting with $W = 100K$ examining pMCSKY, SOP and PSOD only. Figure 4 employs the 3-dimensional dataset with $W = 10k$. The results indicate that in all of the experiments, the PSOD algorithm is less efficient than the pMCSKY and the SOP algorithms. This is due to the fact that the former needs more range queries in order to update the neighbor list of each data point when the window slides; in addition, it does not use any skyband query to decrease the number of stored neighbors. Another remark about the PSOD algorithm is that the speed-down when the slide size increases is in most cases close to linear, whereas it is sublinear for the other two techniques.

We now turn our attention to the two best performing techniques, namely pMCSKY and SOP. While pMCSKY is in most of the cases a little better than SOP (reaching up to 35% faster running times), when the slide size increases to 50%, SOP runs faster. Essentially, the pMCSKY algorithm is a combination of the micro-clusters of the pMCOD and AMCOD algorithms with the skyband query and the *LSky* structure of SOP. This combination has all the benefits and the drawbacks of the micro-clusters. More specifically, on the one hand, it is beneficial by eliminating the need for neighbor search and storage as well as decreasing the number of data points that need to be processed in order to find the window's outliers. On the other hand, the R_{min} and k_{max} restrictions, like the AMCOD algorithm, cause less clusters to be formed, most of which are more vulnerable to breaking up. When a micro-cluster is destroyed, the remaining alive data points that belonged to the cluster need to be reinserted into the window by computing from the beginning the range queries for possible neighbors. This is the main reason that the algorithm is slower than the SOP one when the slide is 50% of the window size.

Algorithm	Average time per slide	Total time
pMCOD (16 instances)	141 ms	72998 ms
pMCSKY (16 queries)	55 ms	10819 ms
SOP (16 queries)	54 ms	10710 ms
PSOD (16 queries)	191 ms	38595 ms

Table 2: Average time and total time per algorithm

A final point regarding the Figures 2, 3 and 4 is the difference between the 16 and 100 query setting. The running times of all of the algorithms, when we increase the queries from 16 to 100 is increased as expected. The encouraging remark is that the difference in the running times is in the orders of a few seconds, which means that the algorithms can scale to a large number of queries without suffering from significant performance degradation.

The final experiment includes the single-query algorithm pMCOD. We run the 16-query setting using 16 instances of pMCOD running in parallel. All of the experiments were done on the 1-dimensional dataset with $W = 10k$ and $S = 5\% * W = 500$. Table 2 shows the results of the experiment. The average time per slide of the pMCOD algorithm is 141 ms, almost 3 times the average time of the pMCSKY and SOP algorithms, whilst the total time it took for all of the instances to end is 7 times more than the time one instance of pMCSKY needed. Attention should also be given in the comparison between the pMCOD and PSOD algorithms. While pMCOD's average time per slide is lower than PSOD's, the total time it took for the 16 instances is 2 times more than the total time of PSOD, which means that despite the low running times of each individual instance of the single-query algorithm, the total time is increased.

5.2 $R - k - W - S$ use case

In this use case, we compare the algorithms pMCSKY, SOP, PSOD on a total of 2500 queries in order to examine their behavior when both the application and windowing parameters increase. The 2500 queries come from 10 distinct values for R and k and 5 distinct values for W and S. The W parameter ranges from 10k to 50k whilst the S parameter ranges from 500 to 10000 with the lowest percentage between window and slide being the 1% ($W = 50k, S = 500$) and the maximum being the 100% ($W = 10k, S = 10k$). The datasets used are both the 1-dimensional (Stock) and 3-dimensional (TAO) and we measured the average running time until the algorithm instance was completed.

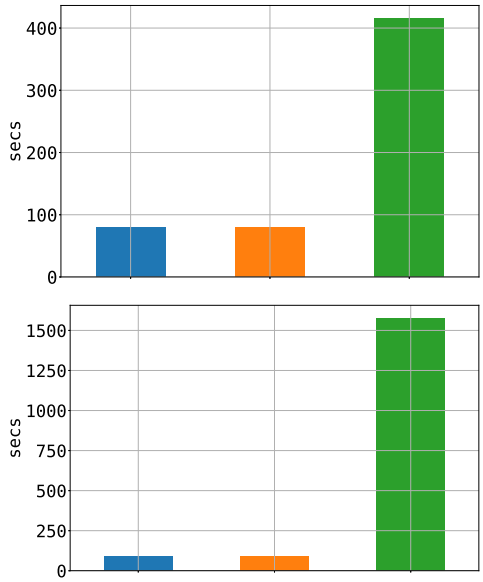

Figure 5: Experiments using Stock (top) and TAO (bottom) for pMCSKY (blue-left bar), SOP (orange-middle bar) and PSOD (green-right bar)

Figure 5 presents the results. As the figure shows, the PSOD algorithm is still lacking in efficiency compared to the other two algorithms. The two winning algorithms differ marginally, i.e., their difference is a few seconds while the whole experiment lasts for a couple of minutes and more. This is an expected result because, when the window grows bigger (by increasing the max W variable), the number of neighbors that the PSOD algorithm stores is also increasing. On the contrary, SOP and pMCSKY do not store as many neighbors because of the skyband query. If we drill down to compare the two faster algorithms, we can observe that pMCSKY is by a small fraction worse than SOP. This happens because a new restriction is added to micro-clusters (they need to be formed by data points on the smaller window), which results in fewer micro-clusters being created. However the small difference between the algorithms implies that even a small number of micro-clusters can have a positive impact on the performance.

Finally, Figure 6 presents the results of the three main algorithms on the 1-dimensional dataset with the difference that this time, the dataset has 5 times more data points than in the previous experiment. The results of this case is as expected with the remark that the pMCSKY and SOP's running times degraded faster than PSOD in comparison to the previous experiment. While the mean value of the data for both experiments is the same, the second experiment's standard deviation is doubled from the first one. This in turn means that micro-clusters are not formed as often as the first experiment on the pMCSKY algorithm while the skyband query's termination condition is more difficult to achieve thus the need to issue more range queries for both SOP and pMCSKY.

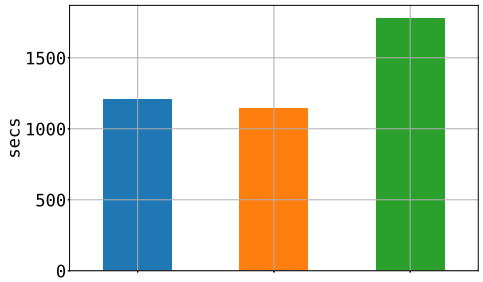

Figure 6: Experiments using Stock with 5 times more data points for pMCSKY (blue-left bar), SOP (orange-middle bar) and PSOD (green-right bar)

6 CONCLUDING REMARKS

This work expanded the available solutions of multi-parameter distance-based outlier detection algorithms for data streams through transferring the relevant state-of-the-art algorithms to a distributed setting. Moreover, a novel solution is presented that is shown to have similar, if not slightly better, performance than the best performing existing solution. More specifically, we have implemented in Flink and investigated the top three algorithms to date, namely SOP, PSOD and AMCOD, which support multiple query parameters; in addition, for the first two solutions, we have provided support for both multiple query and multiple application parameters. More importantly, we also combined techniques from the SOP algorithm and the first distributed distance-based outlier detection algorithm pMCOD to take advantage of the benefits of both thus introducing the pMCSKY solution. The experimental evaluation using a real cluster with three physical machines and 18 cores provided strong evidence that SOP and pMCSKY are the best performing solutions, with their differences being small. In summary, this work both introduces a new multi-parameter distance-based outlier technique and provides a comparative study of the state-of-the-art algorithms in a distributed setting. All of our implementations have been made publicly available.

We identify two main ways for future extensions. The first one is concerned with removing the restrictions from the micro-clusters in the pMCSKY solution. The restrictions on the radius, number of points and window size are shown to outweigh the benefits of the micro-clusters. By completely removing or just mitigating the constraints, the algorithm can significantly speed-up the processing of each slide. The second direction for future work is to try and eliminate unnecessary communication between the machines of the cluster. This can be done with a more advanced partitioning scheme that will be able to better balance the load on the machines taking into account more criteria than the number of points allocated to each partition, such as the communication required.

ACKNOWLEDGMENT

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE (project code:T1EDK-01944)

REFERENCES

[1] Charu C. Aggarwal. 2013. *Outlier Analysis*. Springer.

[2] F. Angiulli and F. Fassetti. 2007. Detecting distance-based outliers in streams of data. In *CIKM*. 811–820.

[3] Lei Cao, Jiayuan Wang, and Elke A Rundensteiner. 2016. Sharing-aware outlier analytics over high-volume data streams. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 527–540.

[4] Lei Cao, Yizhou Yan, Caitlin Kuhlman, Qingyang Wang, Elke A. Rundensteiner, and Mohamed Y. Eltabakh. 2017. Multi-Tactic Distance-Based Outlier Detection. In *ICDE*. 959–970.

[5] Lei Cao, Di Yang, Qingyang Wang, Yanwei Yu, Jiayuan Wang, and Elke A Rundensteiner. 2014. Scalable distance-based outlier detection over high-volume data streams. In *ICDE*. 76–87.

[6] Edwin M. Knorr, Raymond T. Ng, and Vladimir Tucakov. 2000. Distance-based Outliers: Algorithms and Applications. *The VLDB Journal* 8, 3-4 (2000).

[7] Maria Kontaki, Anastasios Gounaris, Apostolos N Papadopoulos, Kostas Tsichlas, and Yannis Manolopoulos. 2016. Efficient and flexible algorithms for monitoring distance-based outliers over data streams. *Information systems* 55 (2016), 37–53.

[8] Theodoros Toliopoulos, Anastasios Gounaris, Kostas Tsichlas, Apostolos Papadopoulos, and Sandra Sampaio. 2018. Parallel Continuous Outlier Mining in Streaming Data. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 227–236.

[9] Theodoros Toliopoulos, Anastasios Gounaris, Kostas Tsichlas, Apostolos Papadopoulos, and Sandra Sampaio. 2019. Continuous Outlier Mining of Streaming Data in Flink. *CoRR* abs/1902.07901 (2019).

[10] Luan Tran, Liyue Fan, and Cyrus Shahabi. 2016. Distance-based outlier detection in data streams. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1089–1100.

[11] D. Yang, E.A. Rundensteiner, and M.O. Ward. 2009. Neighbor-based pattern detection for windows over streaming data. In *EDBT*. 529–540.

[12] Peter N Yianilos. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, Vol. 93. 311–321.

[13] Guanzhe Zhao, Yanwei Yu, Peng Song, Geng Zhao, and Zhe Ji. 2018. A Parameter Space Framework for Online Outlier Detection Over High-Volume Data Streams. *IEEE Access* 6 (2018), 38124–38136.