Poster: Multi-query Outlier Detection Over Data Streams *

Lei Cao†, Jiayuan Wang‡, Elke A. Rundensteiner‡
†IBM T.J. Watson Research Center Yorktown Heights, NY 10598, USA
‡Worcester Polytechnic Institute Worcester, MA 01609, USA
caolei@us.ibm.com, jwang1|rundenst@cs.wpi.edu

ABSTRACT

Real-time analytics of anomalous phenomena on streaming data typically relies on processing a large variety of continuous outlier detection requests, each configured with different parameter settings. The processing of such complex outlier analytics workloads is resource consuming due to the algorithmic complexity of the outlier mining process. In this work we propose a sharing-aware multi-query execution strategy for outlier detection on data streams called SOP. The key insight of SOP is to transform the problem of handling a *multi-query outlier* analytics workload into a *single-query skyline* computation problem. SOP achieves minimal utilization of both computational and memory resources for the processing of these complex outlier analytics workload.

CCS Concepts

•**Information systems** → **Data stream mining;**

Keywords

Outlier,Stream,Multi-query

1. INTRODUCTION

Motivation. Nowadays, data-intensive stream monitoring applications ranging from credit card fraud detection, network intrusion prevention, stock investment tactical planning to telephone fraud detection necessitate the extraction of outliers from huge volumes of stream data in a near real-time fashion.

In recent years distance-based outlier methods [7, 10, 2, 4] have been widely adopted for the detection of outliers in high volume stream data due to their simplicity and insensitivity to concept drift. In the distance-based outlier model initially proposed in [7] an object O is considered as an outlier if it has fewer than k neighbors in the dataset D, where a neighbor is defined to be any other object in D that is within a distance range r from object O.

To discover abnormal phenomena from streaming data using this model of distance-based outliers, a set of input parameters have to be specified by the analyst. For example, when monitoring the potential credit fraud in bank transaction streams, analysts may look for unusual transactions whose values *in recent days* significantly differ from those of *the majority of* transactions made by peers at similar income levels. To detect such outliers utilizing the distance-based outlier technique, several parameter values have to be appropriately set. These include the r parameter that defines the notion of *significance* in transaction value dissimilarity and the k parameter that determines the *majority* of the peer transactions. In addition sliding window semantics [3] need to be applied to ensure that the outliers are continuously detected based on the *most recent* portion of the input stream only. Out-of-date information is typically no longer relevant for the interpretation of the recent outlier detection results. Thus it should be purged from the active stream window. Therefore the sliding window specific parameters, such as the *window* and *slide* sizes, also have to be specified by the analyst.

A stream outlier detection system may need to handle a large number of such parameterized queries for a variety of reasons. First, multiple analysts monitoring the same input stream may submit their outlier search requests with different parameter settings. Even a single data analyst may submit multiple queries with distinct parameter settings, because determining apriori the most effective input parameters is difficult – if not impossible – especially when faced with an unknown or widely fluctuating input stream. Furthermore, each analyst may provide their own personalized understanding of what the "most recent" portion of the data means. Hence they may submit multiple outlier detection requests with different window related parameter settings.

Thus, a stream processing system must be able to accommodate a large outlier analytics workload, and thus striving to capture the most valuable outliers in the stream.

Limitations of the State-of-the-Art. Although efforts have been made in developing efficient algorithms for distance-based outlier detection on data streams [6, 8, 1], these algorithms focus on handling *one single outlier request* with a fixed parameter setting. The simultaneous execution of multiple requests with varying *pattern* and *window* specific parameters remains largely unexplored.

Our Proposed SOP Approach. In this work we propose an innovative approach, called SOP, that efficiently handles an outlier analytics workload composed of a large number of outlier detection requests with arbitrary parameter settings, while still guaranteeing that each data point is processed *only once*.

Our contributions include: 1) Our SOP framework is the first to tackle the problem of shared execution of multiple outlier requests with arbitrary pattern and window specific parameters in the stream context. 2) The key innovation of SOP is to transform the multi-query outlier problem into a single-query skyband problem. The output of the skyband query is proven to be minimal yet sufficient

*This work is an abbreviation of our SIGMOD'16 paper [5]

DEBS '16 June 20-24, 2016, Irvine, CA, USA

ACM ISBN 978-1-4503-4021-2/16/06.
DOI: http://dx.doi.org/10.1145/2933267.2933292

for determining the outlier status of each point for any parameter setting on the workload. 3) Our customized skyband algorithm is tuned to process outlier requests with diverse parameter settings. K-SKY is proven to be optimal in the number of points being evaluated. 4) Leveraging the commonality and dominance among the data populations, we are able to utilize one specific skyband query to support multiple queries with varying window specific parameters. By this full sharing is achieved across the query windows.

2. PROBLEM DEFINITION

We use $q_i(r_i, k_i)$ to denote the outlier detection query q_i with r_i and k_i the distance range and neighbor count thresholds.

We focus on periodic sliding window semantics as proposed by CQL [3]. Each query q_i has a window size $q_i.win$ and a slide size $q_i.slide$. Each window W_c of q_i has a starting time $W_c.T_{start}$ and an ending time $W_c.T_{end}$ = $W_c.T_{start}$ + $Q.win$. Periodically the current window W_c slides, causing $W_c.T_{start}$ and $W_c.T_{end}$ to increase by $q_i.slide$. The arrival time of point *p* is denoted as $p.time$. If $W_c.T_{start} \leq p.time < W_c.T_{end}$, *p* falls into W_c.

Outliers will be generated based on the points that fall into the current window W_c, also called the population of W_c. A point p in W_c might have a different outlier status (outlier or inlier) in the next window W_{c+1}, since each window has a different population.

Definition 1. ***Distance-Based Outlier Detection In Sliding Windows.*** *Given a stream S, a streaming distance-based outlier detection query $q_i(r_i, k_i, win_i, slide_i)$, q_i continuously detects and outputs the outliers in the current window W_c after the window slides from the previous to the current window W_c.*

Multiple Outlier Detection Optimization. Outlier detection requests on the same input stream can have arbitrary settings on all four parameters *r*, *k*, *win*, and *slide*. The set of outlier requests q_i that must be concurrently processed is denoted as query group $\mathbb{Q}$. Each query q_i in $\mathbb{Q}$ is called a member query of $\mathbb{Q}$. Our goal is to minimize both the processing time and the memory space needed to answer all queries in a large outlier analytics workload.

3. VARYING THE DISTANCE-BASED OUTLIER PARAMETERS

In this section we first introduce our transformation of processing a workload composed of queries with varying *r* but fixed *k* parameters into a skyband query. Then we present our K-SKY algorithm that supports such skyband query with optimality. Next we extend K-SKY to handle outlier detection queries with arbitrary *k* and *r* parameters. In this section we assume all queries share the same sliding window parameters *win* and *slide*.

3.1 From Multi-Query Outlier Workloads to Single Query Skyband Processing

Given a query group $\mathbb{Q}$ with *varying r* but *fixed k* parameters, the goal is to design an approach that supports all member queries in $\mathbb{Q}$ with each point *p* of data stream *S* processed only once in each current window W_c. The key insight here is that given such a query group $\mathbb{Q}$ and one data point *p* in current window W_c of stream *S*, the output of one single customized K-skyband query is sufficient yet necessary to determine the outlier status of *p* with respect to all queries in $\mathbb{Q}$.

K-skyband query is a generalization of the well known *skyline* concept. As defined in [9] a K-skyband query reports all points that are *dominated* by no more than K points. The case K = 0 corresponds to a conventional skyline.

To map our problem of determining the *outlier status of a given point p* to the K-skyband problem, we have to similarly define the *domination relationship* between any pair of data points in the dataset D_{W_c}, i.e., the population of the current window W_c. The key observation here is that given any two points p_i and p_j, two key factors, namely their relative *arrival time* and the *distance* to the point *p* under evaluation, determine whether p_i is more important than p_j in terms of evaluating the outlier status of *p*.

Let us introduce a query group $\mathbb{Q}$ used in the remainder of this section. Assume we have a query group $\mathbb{Q}$: $\{q_1(r_1)$,$q_2(r_2)$,..., $q_m(r_m)$, $q_{m+1}(r_{m+1})$, ..., $q_n(r_n)\}$, where r_m represents the *r* parameter of query q_m. The *r* parameter of q_1, q_2,...,q_n monotonically increases, that is, $r_1 < r_2 < ... < r_m < r_{m+1} < ... < r_n$.

Distance Dimension. In distance-based outlier definition, points in a dataset *D* are classified either as outliers or inliers. Thus, the process of identifying outliers in *D* is equivalent to the process of finding and eliminating inliers from it. *p* is guaranteed to be an inlier once *k* neighbors are acquired in *D*. Given two points p_i and p_j, assume $dist(p_i, p) < r_m < dist(p_j, p) < r_{m+1}$. Then p_i is the neighbor of *p* with respect to query subset $\mathbb{Q}_i$ = $\{q_m, ...,q_n\}$, while p_j is the neighbor of *p* only with respect to query subset $\mathbb{Q}_j$ = $\{q_{m+1}, ...,q_n\}$. $\mathbb{Q}_i \supset \mathbb{Q}_j$. In other words p_i satisfies the neighbor requirement of more queries than p_j. For the evaluation of *p*, p_i is more important than p_j, because p_i makes the outlier status of *p* closer to be determined with respect to all queries in $\mathbb{Q}$ than p_j. In this perspective p_i *dominates* p_j.

On the other hand, assume $r_m < dist(p_i, p) < dist(p_j, p) < r_{m+1}$. Then p_i and p_j are both neighbors of *p* for the same set of queries $\{q_{m+1}, ..., q_n\}$. In this scenario p_i and p_j equally affect the outlier status of *p* although $dist(p_i, p) \neq dist(p_j, p)$. Based on this observation we now are ready to re-define the distance function $dist(p, p_i)$ so to normalize the distance between data points. The *original* distance function is denoted as $dist_o(p, p_i)$ instead.

Definition 2. *Given a query group $\mathbb{Q}$: $\{q_1(r_1), q_2(r_2), ..., q_m(r_m), q_{m+1}(r_{m+1}), ..., q_n(r_n)\}$ with $r_1 < r_2 < ... < r_m < r_{m+1} < ... < r_n$, $dist(p, p_i) = m + 1$ if $r_m < dist_o(p, p_i) \leq r_{m+1}$ for $0 \leq m \leq n$ with r_0 defined as $-\infty$ and r_{n+1} as ∞.*

This new *normalized distance* calculated using Def. 2 now accurately represents the importance of each data point to *p*.

Time Dimension. In the streaming context the presence of the time dimension further complicates matters. In particular we cannot simply claim that one data point p_i closer to *p* impacts the status of *p* more than the other points. Instead the arrival time of the data points also has to be taken into consideration. A point p_i that arrived later in the window may have a more *decisive* impact on the outlier examination process compared to an earlier arriving p_j even if p_i is not closer to p than p_j. This is so because the *younger* a data point p_i is, the longer its neighbor relationships (if any) with *p* will persist into the future.

Domination Relationship. We now define the *domination relationship* between the pair of points in dataset D_{W_c} that takes both the distance and time dimensions into consideration.

Definition 3. ***Domination Relationship.*** *Given a query group $\mathbb{Q}$: $\{q_1(r_1), q_2(r_2), ..., q_m(r_m), q_{m+1}(r_{m+1}), ..., q_n(r_n)\}$ with $r_1 < r_2 < ... < r_m < r_{m+1} < ... < r_n$, point p_i* ***dominates*** *p_j with respect to point p if: (1) $p_i.time > p_j.time$; (2) $dist(p, p_i) \leq dist(p, p_j)$ ($p_i, p_j \in D_{W_c} - p$) and $p \in D_{W_c}$; (3) $dist(p, p_i) \leq n$, with dist() the normalized distance of $\mathbb{Q}$ defined in Def. 2.*

In other words, given a data point p_i, p_i dominates another point p_j only if p_i expires later than p_j from window W_c (Condition 1)

and it is not further away from p than p_j (Condition 2). The third condition in the domination rule filters out any data point p_i that is not a neighbor of p for any query in $\mathbb{Q}$, as otherwise this p_i would never be influencing the outlier status of p.

Based on the domination relationship defined in Def. 3, the outlier status of p with respect to all queries in $\mathbb{Q}$ can now be correctly answered based on the skyband points delivered by one single $(k-1)$-skyband query denoted as Q^s, namely the K-skyband query with K specified as k-1.

Lemma 1. *Given a query group $\mathbb{Q}$, for any data point p, the output of the skyband query Q^s corresponding to $\mathbb{Q}$, denoted as $\mathbb{S}_p$, is **sufficient** and **necessary** to continuously determine the outlier status of p with respect to all queries in $\mathbb{Q}$.*

3.2 The K-SKY Algorithm

Although the traditional *K-skyband* algorithms could be applied to support our Q^s query [9, 11], we now design a customized algorithm called K-SKY that more efficiently supports the multiple outlier detection queries. K-SKY encompasses two optimization principles, namely *time-aware prioritization* and *least examination*. K-SKY is proven to be optimal in minimizing the number of data points to be evaluated in the skyband point discovery process.

Time-Aware Prioritization Principle. In sliding window streams the data points are naturally ordered by their arrival time. In other words, all data points can effectively been considered to be sorted on their arrival time attribute upon arrival. By the definition of the domination relationship, later arrivals will never be dominated by the earlier arrivals. Leveraging this property we prioritize the order in which the K-SKY algorithm processes the data points. More specifically K-SKY always conducts the search with a later arriving data points first order. By this if one data point is not dominated by more than k points in the distance attribute and thus considered to be a skyband point, then it is not necessary to evaluate it again. This is so, because it will be guaranteed to never be dominated by other points evaluated later. Thus all skyband points can be discovered in one pass over the data set.

Better yet, given a data point p_i with $dist(p_i, p)$ no larger than the smallest r value r_1 in $\mathbb{Q}$, if p_i has already been dominated by k points when evaluated, K-SKY can be terminated immediately. This is so because all remaining (unevaluated) points would be dominated by at least these k points that dominate p_i. Therefore K-SKY can safely terminate without even examining all points.

Least Examination Principle. In the sliding window context, an existing point p in the previous window W_{c-1} needs K-SKY to update its skyband points when the stream slides to the current window W_c. The key observation here is that given the skyband points of the window W_{c-1}, to acquire the skyband points of a new window W_c, only a small fraction of points in W_c need to be evaluated, namely the new arrivals and the unexpired skyband points of W_{c-1}.

This is so because any existing data point p_i in W_c could not possibly be a skyband point in window W_c if p_i is not also a skyband point in W_{c-1}. If p_i is not listed in the *skybandPoints* set of W_{c-1}, p_i must be dominated by at least k data points p_j in *skybandPoints*. By the domination rule defined in Def. 3, if p_j dominates p_i, $p_j.time > p_i.time$. This indicates p_j would not expire earlier than p_i. If p_i is still valid in window W_c, p_j would also remain valid. Therefore in W_c, p_i could not possibly be a skyband point, since it is still dominated by at least k data points.

Leveraging the time-aware prioritization and least examination optimization principles, K-SKY is able to discover all skyband points by scanning the data set at most once. Furthermore, it may terminate without even seeing all data points.

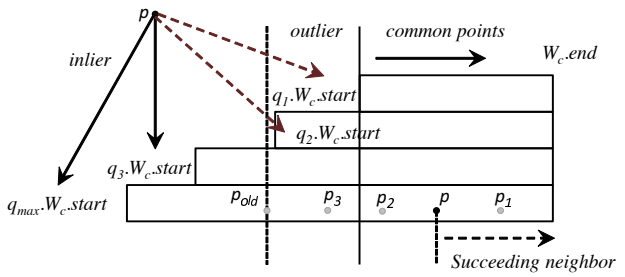

Figure 1: Queries with varying window sides

K-SKY is shown to be *optimal* in minimizing the number of points being evaluated in the execution process. Due to space constrain, the proof is omitted here.

3.3 Sharing-Aware Multi-Skyband Solution

We now relax our problem to consider varying both k and r parameters. One simple approach to handle such scenario would be to divide this workload into groups, each of which contains queries with the identical k parameter value. This then would simplify our problem into a *multi-skyband query problem* with only the k parameter varying. Intuitively our problem then could be handled by directly applying K-SKY on each group of queries. However this solution requires the independent identification and maintenance of the skyband points for each group of queries. This inevitably leads to significant wastage of CPU and memory resources.

We thus propose a sharing-aware solution that efficiently solves this *multi-skyband query problem*. The key observation here is that a large number of skyband points are likely to be shared across these skyband queries. By maintaining the skyband points in one integrated data structure, given a point p_i, only one single skyband point evaluation operation is required to correctly answer all skyband queries. This way we assure that multiple skyband queries are supported by K-SKY, while still guaranteeing that each data point is evaluated exactly *only once*. Due to the space constrain, the details of this solution is omitted here.

4. VARYING SLIDING WINDOW PARAMETERS

4.1 Varying the Window Parameter - Win

Here we first examine the scenario when the window sizes vary, while the slide size remains stable. Therefore all queries slide to a new window at the same time. In other words, they are *synchronized*. All queries require output at exactly the same moment, i.e., at time $W_c.end$ in Fig. 1. This observation leads to an important characteristic. Given a query group $\mathbb{Q}$ with member queries having the same slide size but arbitrary window sizes, $\mathbb{Q}$ can be supported with one skyband query with respect to q_{max} denoted as q^s_{max}, namely the member query with largest window in $\mathbb{Q}$ as in Fig. 1. Intuitively this is so because the largest window covers all smaller windows. Therefore skyband points discovered in the largest window can be utilized to answer all queries in the group.

Therefore by employing the K-SKY algorithm and collecting the skyband points for this special skyband query q^s_{max}, this outlier query group $\mathbb{Q}$ can be correctly answered with each point p in the data stream S evaluated only once in each window that contains p.

As shown in Sec. 3.2, K-SKY gives more preference to the points arriving later than the points arriving earlier. That is, K-SKY always processes the later arrivals first. On the other hand the later arrivals in the stream happen to be the common points among

the data populations covered by the current windows of different queries (Fig. 1). Therefore K-SKY naturally leverages the data commonality among the windows of distinct queries. Redundant computations are eliminated.

4.2 Varying the Slide Parameter

Next, we consider the case where all queries have the same window size, while their slide sizes vary. Unlike the previous varying window size case, these queries are not synchronized. That is, their windows move at a different pace. Therefore no stable relationship holds across the data populations covered by the active windows with respect to different queries. In other words there is no such query whose active window continuously contains the windows of other queries. Therefore the above strategy supporting queries with various window sizes does not handle this case.

To solve this problem, given a query group $\mathbb{Q}$, we build a single *swift query* q_{sft} that correctly answers all member queries of $\mathbb{Q}$. Q_{sft} has the same window size as all member queries in Q, while its slide size is set as the greatest common divisor on the slide sizes of all the queries in $\mathbb{Q}$.

Intuitively by definition of the greatest common divisor, $\forall q_i \in \mathbb{Q}$, we have $q_i.slide \bmod q_{sft} = 0$. Therefore at any time t_j when $q_i \in \mathbb{Q}$ produces an outlier result $q_i.outlier$, q_{sft} would also be producing result $q_{sft}.outlier$. Furthermore, since $q_i.win = q_{sft}.win$, the points covered by the window of q_i and q_{sft} would be identical at t_j. Therefore at any t_j $q_i.outlier = q_{sft}.outlier$. Hence Q_{sft} is sufficient to represent all queries in $\mathbb{Q}$.

Therefore a query group $\mathbb{Q}$ with varying slide sides can be supported by one skyband query with respect to this special outlier query q_{sft}. It is straightforward to determine at runtime when to output the outlier detection results for each query by tracking for each query the time at which the window slides.

4.3 Varying Both Win and Slide Parameters

We now describe our solution for the case when both window parameters, namely win and slide, vary. This solution is a straightforward combination of the techniques introduced in the last two sections. In particular, we simply build one *single swift query* that has the largest window size among all member queries and its slide size as the greatest common divisor of the slide sizes of all member queries. A specific skyband query with respect to this single swift query will then be employed to collect skyband points, namely the evidence to prove the outlier status of a given point p.

5. RELATED WORK

Distance-based Outliers on Streaming Data. With the emergence of digital devices generating data streams, outliers on streaming data are one type of anomalies recently studied [6, 8, 1]. Existing work [6, 8, 1] focuses primarily on processing a single outlier detection request. In particular [1] leverages the observation that the neighbors p_i of a point p that arrived after p do not expire before p expires. They make a distinction between the preceding neighbors of p, P_p, i.e., those that will expire before p, and the succeeding neighbors of p, S_p, those that will persist during the entire lifetime of p. They first introduce the idea of a "safe inlier" as a point p with $\geq$ k succeeding neighbors.

[8] further improves on [1] by leveraging the *safe inlier* concept of [1]. That is, it organizes the points into a queue based on the number of their succeeding neighbors, so that it can efficiently schedule the necessary checks that have to be made when the window slides. However it still relies on range query searches to process the newly arriving points. Therefore it cannot provide real time responsiveness when applied to high velocity streaming data. [8] also touches on supporting multiple outlier detection queries for the case of varying pattern-specific parameters. Given a data point p they first utilize a range query to find all points that satisfy the neighbor condition of all queries in the query group. Then a post-processing step is applied to filter the unnecessary points from this large neighbor set to reduce the maintenance costs. In our work by directly transforming the multi-query outlier problem into the single query skyband problem, we only collect the necessary evidence that is sufficient to answer multiple outlier queries, therefore significantly outperforming this method.

By leveraging the temporal relationships among stream data, [6] overcomes the limitation of prior methods [1, 8] of undertaking full range query searches by discovering as early as possible safe inliers in the scan process. However multiple detection requests are not supported in [6].

6. CONCLUSION

In this work, we present the first solution, called SOP, for efficient shared processing of a large number of distance-based outlier detection requests over sliding window streams. SOP requires only one single pass over the data points to support a huge workload composed of a large number of outlier detection requests with arbitrary input settings for all pattern and window specific parameters.

7. ACKNOWLEDGEMENT

This work is supported by NSF grants: IIS-1018443 & 0917017.

8. REFERENCES

[1] F. Angiulli and F. Fassetti. Distance-based outlier queries in data streams: the novel task and algorithms. *Data Min. Knowl. Discov.*, 20(2):290–324, 2010.

[2] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *PKDD*, pages 15–26, 2002.

[3] A. Arasu, S. Babu, and J. Widom. The cql continuous query language. *VLDB J.*, 15(2):121–142, 2006.

[4] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *KDD*, pages 29–38, 2003.

[5] L. Cao, J. Wang, and E. Rundensteiner. Sharing-aware outlier analytics over high-volume data streams. In *SIGMOD Conference*, 2016.

[6] L. Cao, D. Yang, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner. Scalable distance-based outlier detection over high-volume data streams. In *ICDE*, pages 76–87, 2014.

[7] E. M. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, pages 392–403, 1998.

[8] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos. Continuous monitoring of distance-based outliers over data streams. In *ICDE*, pages 135–146, 2011.

[9] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[10] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *SIGMOD Conference*, pages 427–438, 2000.

[11] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *IEEE Trans. Knowl. Data Eng.*, 18(2):377–391, 2006.