UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

ONLINE DETECTION OF OUTLIERS FOR DATA STREAMS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

MD. SHIBLEE SADIK
Norman, Oklahoma
2013

ONLINE DETECTION OF OUTLIERS FOR DATA STREAMS

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dr. Le Gruenwald, Chair

Dr. Sudarshan Dhall

Dr. Changwook Kim

Dr. Scott Moses

Dr. Sridhar Radhakrishnan

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In applications, such as Web clicks and environmental monitoring, data are in the form of a stream, each of which is an infinite sequence of data points with explicit or implicit timestamps and has special characteristics, such as transiency, uncertainty, dynamic data distribution, multi-dimensionality, asynchronous data arrival, dynamic relationships, and schema heterogeneity of data from different sources. In those applications, outliers do exist due to many reasons including human error, instrument error, catastrophe, and malicious behavior. Being able to detect outliers effectively is critical to many data management and mining tasks. However, not much research has been conducted to discover outliers in data stream applications, especially for those involving multi-dimensionality, related, heterogeneous, and asynchronous streams.

In this dissertation, two innovative outlier detection algorithms, Orion and Wadjet, which take all the data streams' characteristics into consideration are presented. Orion is designed for applications where data are from single stream. It looks for a projected dimension that reveals the outlier nature of multi-dimensional data points with the help of an evolutionary algorithm, and identifies a data point as an outlier if it resides in a low density region in that dimension. Wadjet is designed for applications where data are from multiple, heterogeneous, and asynchronous streams. It has two phases: in the first phase, it processes each stream independently like Orion, and in the second phase, it captures and continuously evaluates the cross-correlation, if any, among the data points from multiple streams, and identifies a data point as an outlier if its value does not conform to the captured cross-correlation.

Extensive theoretical and empirical analyses have been conducted to evaluate the performance of Orion and Wadjet using real and synthetic datasets. The evaluation results show that both algorithms have better accuracy and execution time than the state-of-art techniques when applied to homogeneous data stream applications. The results also show that Wadjet is effective in detecting outliers in heterogeneous data streams which cannot be handled by existing algorithms.

CHAPTER I
INTRODUCTION

1. Objective

The objective of this research is to develop efficient and accurate outlier detection techniques for single and multiple data streams that address the following characteristics:

i. The transiency of data items in data streams;

ii. The uncertainty of data items in data streams;

iii. The infiniteness of data streams;

iv. Dynamic data distribution of data streams;

v. Multi-dimensionality of data points;

vi. Dynamic cross-correlation among heterogeneous data points.

In the following sections, we present the background of data streams (Section 2) and outliers (Section 3), the significance of outlier detection (Section 4) and the major challenges of outlier detection for data streams (Section 5).

2. Data Streams and Applications

In this era of information, the data assimilation process has changed significantly. The applications like web clicks, network traffic monitoring, environmental sensor monitoring, etc. generate a sequence of data records in an orderly fashion. The

emerging popularity of this type of applications secured a name for the data model called Data Streams. A data stream is an infinite sequence of data points ordered by explicit or implicit timestamps. A data stream is further characterized by continuous arrival [1], unbounded volume [1], time-varying [2], real-time [3], high arrival rate [4], uncertainty, drifting concepts [2] and multi-dimensionality. Not all applications share all the properties but most of them share one important property in that in all cases a data stream is an ordered sequence of data points. Formally a data stream is a tuple (D, T) where D is a sequence of the data points $(d_0, d_1, d_2, d_3, \dots)$ and T is an associated sequence of the timestamps $(t_0, t_1, t_2, t_3, \dots)$ [5].

Environment monitoring is a very popular application of data streams, where a group of sensors is placed together to monitor environmental attributes like temperature, humidity, wind speed, and soil moisture. The group of sensors is accompanied by a radio transmitter. The sensors measure the environmental attributes at a regular interval and send them to a base station using a radio transmitter [6]. The base station further processes the data. In these kinds of application, each sensor produces a sequence of data points with associated timestamps. In an environmental monitoring situation, these data are important for a specific amount of time and hence we see the transiency of data points. The data points are transmitted over a radio channel; such transmissions are susceptible to corruption and interference, which cause the data values to become uncertain. Additionally, the distribution of data points changes over time. For example, the data distribution of temperature during the night is different from that during the day (usually the temperature during the night time is lower than that of the daytime). Last

but not least, these applications are envisioned for an infinite amount of time, hence produce infinite data points.

Another popular data stream application is network traffic monitoring. A typical network consists of a group of hosts, few switches and routers. In order to maintain quality of service and reliability, the network instruments are continuously monitored. Each network equipment reports a group of attribute values to the monitoring station For example, a router reports, source address, destination address, protocol name, packet size, etc; a host reports source address, packet size, application id, etc; a switch reports source mac address, destination mac address, etc. to the base station for monitoring purposes. Hence, each item produces a data stream consisting of multiple attribute values [7]. In this application, each data point has an associated timestamp and is produced online. The number of data points is infinite since the monitoring is a never-ending process. The distribution of network traffic also changes based on the busy and idle time of the network. Moreover, the data points from multiple sources are asynchronous and heterogeneous in nature since each item reports a different set of attribute values.

3. Outliers

An outlier is a data point which is significantly different from other data points in the dataset or does not conform to the expected normal behavior or conforms well to a defined abnormal behavior [8, 9]. In this definition, the phrases “significantly different,” “does not conform to the expected normal behavior,” and “conforms well to a defined abnormal behavior” are very subjective and deserve intelligent scrutiny;

therefore the definition of outlier bears some vagueness. Outliers are often mentioned as anomalous data points; an anomalous data point is one that does not conform to the expected normal behavior. The data points that are not outliers are often called inliers; we use the term inlier to represent a data point which conforms well to the expected normal behavior. Outliers in different domains are different in nature from one another. An outlier in a credit card transaction is very different from an outlier in meteorological data. Hence, different applications have their own definitions of outliers.

Outliers may appear in a dataset for numerous reasons, like malicious activity, instrumental error, setup error, changes of environment, human error, and catastrophe. Regardless of the reason behind outliers, they may be interesting to the user because they carry some different information for the user than regular data. Some people define outliers as problems, some people define them as interesting items; but in any case, they are unavoidable [9, 10]. In brief, outliers are interesting and take different forms in different types of applications. Chandola et al. [9] classified outliers into three major categories as follows.

Definition 1. Type I Outliers

Isolated individual data points in a dataset are termed as Type I outliers. By definition they are the simplest type and very easy to identify. Intuitively they are far from other data points in the dataset in terms of attribute values.

Definition 2. Type II Outliers

A data point is isolated with respect to the context. Typically, data in this type of dataset has other contextual attributes (e.g., time and location). An outlier is far from other data

points in the same context in terms of value. This is a little bit different from a Type I outlier; a Type I outlier is a data point isolated from all the other data points in the dataset. A Type II outlier was first investigated in time series data in the late seventies. Barnett & Lewis [10] defined Type II outliers as the Additive Outliers (AO) for time series data. The good thing about additive outliers is that they do not influence the other data points in the context, hence they are easy to identify [5].

Definition 3. Type III Outliers

A particular group of data points appear as outliers with respect to the entire dataset. No data point in a small subset is an outlier with respect to the other points in the subset, but as a group, they are the outliers. For contextual data like time series, the entire dataset forms a sequence, hence a particular subsequence is an outlier with respect to the entire sequence. Barnett & Lewis [10] called them Innovations Outliers (IO) for time series data. The bad thing about innovations outliers is that they influence other data points of the same context and try to hide themselves; therefore it is difficult to identify innovations outliers.

A data stream has one temporal context with each data point; so it might have a type II or type III outlier but never a type I outlier. This is because data streams are considered as infinite series and the processing has to be online. Therefore at any particular moment, only a subset of the entire dataset is present, and so a data point cannot be an outlier with respect to an entire dataset. Regardless of the type of outliers, outlier detection is a popular branch of application. We discuss the problem of outlier detection and its significance in the next section.

4. Significance of Outlier Detection

Outlier detection refers to the problem of identifying the outliers in a dataset. Since the definition of outliers is vague and application-dependent, a formal method for outlier detection is not yet developed. By definition, an outlier detection technique takes a dataset as input and outputs the outliers. Despite the vagueness of outliers, several approaches are popular for outlier detection based on the state of the input data. The first approach is called the supervised approach where the outlier detection technique assumes the availability of labeled data [11]. A supervised technique collects knowledge from labeled data and applies the collected knowledge to unlabeled data for outlier detection. The second approach is called the semi-supervised approach which requires only the inliers or the outliers to be labeled. Both of these approaches are less popular due to the lack of labeled datasets. The third and final approach is called the unsupervised approach which does not require any type of labeling, hence, is very popular for outlier detection. However, unsupervised techniques often suffer from higher false alarms [12].

Outliers are less intuitive than regular data points and trigger the curiosity of a user to investigate their causes, hence outlier detection in a dataset is an important part of the data assimilation process [6]. Different applications perform outlier detection for different purposes. One of the most popular purposes is intrusion detection. Typically intrusion causes outliers, hence the presence of outliers is a good sign of an intrusion. Other important purposes include novelty detection (e.g. for medical and public health data), damage detection (e.g. for sensor data), fault detection (e.g. for time series data), and data cleaning, [9]. Outlier detection for data streams has a wide range of

applications and its potential is limitless. Practically every monitoring system requires online outlier detection in order to detect abnormalities on-the-fly. In this section, we discuss some examples of outlier detection in two types of applications: single and multiple streams.

Outlier detection technique for single streams is appropriate where data points from one stream are independent from those from other streams. In that case each stream can be processed independently and therefore we call it single stream application. Typical applications of outlier detection for single stream include fraud detection, fault detection, error detection, etc. Fraud detection refers to the problem of detecting unauthorized transactions in bank accounts, credit cards, insurance agencies, cell phone companies, etc. Here one stream refers to the transactions from one user. This type of applications carries independent streams because transactions from one user are independent of transactions from other users. By definition, fraudulent transactions are significantly different from regular transactions and, hence, can be identified by outlier detection. Many of these applications are becoming online nowadays where transactions are monitored on-the-fly. Outlier detection for single data stream has a great potential in this area where transactions are monitored and fraudulent activities are detected on-the-fly. Basu and Meckesheimer [13] proposed the use of outlier detection for data streams to detect instrument faults. Practically this approach can be used in any industry where machine condition can be monitored on-the-fly. Each machine produces one stream consisting of the machine status, and the machine statuses from multiple machines are independent of each other. A malfunction of any instrument can be detected by outlier detection and, therefore, serious damage due to a catastrophic fault can be avoided. The

offline store and process approach detects faults offline, in which case true faults may go undetected for some time and may have severe consequences. Multiple weather stations are installed in different geographical locations where each station measures meteorological attributes. Weather stations are too far from one another that they often show very little correlation and are processed as independent streams [14, 15]. These stations are often located in remote places that are hard to monitor directly. They may produce erroneous values for numerous reasons, such as instrument fault, abrupt behavior, and erroneous setup [15]. Outlier detection is one way of detecting erroneous values.

A multiple streams application has more than one stream where data points from one stream are related to those from all of some of the other streams; in other words, there are some relationships among some or all streams. Network intrusion detection is an important application of outlier detection for related streams [9]. Traffic status at different network devices is continuously monitored; and a data stream is produced from each device that gives birth to a multiple data streams application [16]. Babu et al. [7] proposed the idea of network monitoring using data streams. A network monitoring application collects network packets, packet traces, active measurements of packet delay, throughput, router configuration data, etc. in order to maintain quality of service or identify potential threats. The data collected from multiple instruments (router, switch, host, etc.) in a network are highly related to each other and, therefore, can be monitored together for potential threats. An outlier detection algorithm can be used to detect any significant deviation of attribute values which may indicate an intrusion or fault. For example, a significant increase of incoming requests can be seen as a potential

denial of service attack. Outlier detection for related data streams has a good potential in patient health monitoring applications where a patient's critical body part is continuously monitored. In this kind of applications, an outlier may occur because of several reasons like patient condition, instrument error or reading error. Moreover, many data in this area take the form of time series, such as Electrocardiograms (ECG) and Electroencephalograms (EEG) where a combination of multiple time series needs to be monitored in order to detect patient condition [17]. Environment monitoring within a small region using a sensor network has become very popular in the last decade [6]. Each sensor produces a sequence of data points with timestamps and sends it to some centralized server for storage and analysis. Typically these sensors are not far from each other and data points from one sensor are highly correlated with data points from other sensors. Like all other data acquisition processes, these sensors are not outlier resistant. Outlier detection for related data streams is a practical way of monitoring these sensors in critical applications.

5. Issues of Outlier Detection in Data Streams

Data streams are new compared to the regular data model. Their characteristics introduce new issues for outlier detection techniques. In this section, we discuss these issues.

Transient

Data points are transient in a data stream [1, 18]. A particular data point is important for a specific amount of time, after which it is discarded or archived [1, 19, 20, 21]. Therefore it is important to keep the data point moving [22]; otherwise it may lose its

importance. However popular outlier detection techniques rely on the store-and-process paradigm [23, 24, 25], where the entire dataset is stored in the first phase to construct an outlier detection model, and each data point is compared to the model or other data points in the second phase to detect outliers among them. These approaches hold the data points for a long period of time and do not detect outliers as they arrive; so for streaming data, these two phase algorithms are inappropriate. A new outlier detection scheme has to be developed that processes data points online.

Requirement 1. An outlier detection technique cannot hold the entire dataset indefinitely and compare each data point D_t to the other data points to detect the outlier-ness of D_t; rather the outlier-ness of D_t should be decided immediately once D_t arrives.

Notion of Time

Unlike regular data, stream data include a notion of time. Each data point has a timestamp associated with it. The association can be explicit (where time is a data attribute) or implicit (when the exact time is not important, but the order of data items is important) [1]. The timestamp gives the temporal context for each data point; thus each data point needs to be processed based on its own temporal context. Outlier detection is no exception; by definition, a data point is an outlier if it has a significantly different value compared to other data points; but if we take temporal context into consideration, a data point must be compared to the other data points with the same temporal context (Type II outlier). Typical outlier detection techniques do not consider the temporal context of the data points [23, 25], rather they compare a data point to the entire dataset; this approach is inherently flawed for data streams since the outlier-ness of a data point can only be detected by comparing it with the data points seen so far. A temperature of

100°F may not look like an outlier if we consider the temperatures of an entire year, but it would certainly look like an outlier if we consider the temperatures of winter days only. In order to detect outliers meaningfully, an appropriate temporal context has to be selected first (if not given by the user) and then, every data point has to be processed based on its temporal context. Moreover an out-of-order data point should be processed based on its temporal context [22] as well.

Requirement 2. A data point has to be compared with the other data points with the same temporal context (occurred within the time period which is semantically related to the timestamp of the data point).

Notion of Infinity

Data streams are seen as infinite sequences of data points as they keep coming from a data source indefinitely. The most significant implication of the notion of infinity is that at any particular time, the entire dataset is not available, i.e., a random access to the entire dataset is not possible for outlier detection [1]. Many outlier detection techniques store the entire dataset first and find the outliers later [10]. An outlier detection technique for data streams cannot store all the data points seen so far because the number of data points is infinite; rather it should store only the summary of the data points seen so far using finite memory/resource and detect outliers based on the summary. For example, for outlier detection techniques that determine whether a data point D_t is an outlier based on D_t's neighbor data points, in order to compute the neighbors, a data density function should be used instead of relying on the availability of the entire dataset and using the pairwise distances of all the data points in the dataset. On top of this, the data density function has to be computed incrementally. Thus an

outlier detection model has to be incremental and cannot assume the availability of the entire dataset.

Requirement 3. In order to detect the outlier-ness of a data point D_t, the outlier detection technique should compare D_t with the summary of the other data points, instead of directly comparing D_t to the other data points. In addition, the summary should be computed incrementally.

Arrival Rate

Data points are continuously coming from a data source. The arrival rate might be fixed or variable but every application must finish processing before the next data point arrives [4], i.e., if the outlier detection is a binary classification task, then the classification has to be done before the next data point comes. If the outlier detection technique fails to process a data point before the next one arrives, the result is flooding. Typical outlier detection techniques compare each data point to all other data points in the dataset in order to detect outliers. If the dataset size is too large, these approaches would require a vast amount of time and may not be able to keep up with the arrival rate. A reasonable accuracy can be achieved if the data point is compared to a much smaller subset and the size of the subset should be decided based on the available processing time. A similar idea is also applicable to outlier detection model construction [26]. Hence the outlier detection time is bounded by the arrival rate of data streams.

Requirement 4. The set of data points or the summary of the data points, to which the current data point is compared to detect outlier-ness, should be adjusted based on the available processing time.

In some data stream applications like sensor networks, the data arrival rate is not fixed [26], but varies over time. In this kind of applications, the available processing time between every two consecutive data points is not the same. If a long period of time is available, the accuracy of outlier detection could be excelled utilizing the time before the next data point arrives; if a short period of time is available, the data point needs to be processed before the next data point arrives which may compromise outlier detection accuracy [26]. Hence the processing has to be adaptive.

Requirement 5. In case of dynamic arrival rate, the set of data points or summary of the data points, to which the current data point is compared to detect outlier-ness, should be adjusted dynamically based on the available processing time.

Concept Drift

The data distribution in a data stream changes over time [2]. This might happen because of changes in environments, changes of trends, etc. This phenomenon is known as *concept-drift* [2]. Many outlier detection techniques use data distribution to identify abnormal behavior [10]. Since data distribution for data streams changes over time, outliers detected for one data distribution might not be the same for another data distribution. For example, the distribution of traffic in a traffic monitoring system during the mornings may be entirely different from the distribution during the evenings; therefore, any assumption about data distribution may lead to incorrect results. Many statistical and machine learning based techniques assume a fixed data distribution for outlier detection [10, 27, 28]; they use a training data set to construct the outlier detection model and later detect outliers based on the model. The problem with this approach is that the training data set represents a fixed data distribution which may

produce meaningful results for some time, but if a concept drift occurs, the same training data set (or the same outlier detection model) may no longer produce meaningful results.

Requirement 6. An outlier detection technique for data streams should not assume any kind of fixed data distribution.

Uncertainty

Data points in a data stream are further characterized by their uncertainty. Data sources such as sensors in a sensor network are exposed to an open environment. They are vulnerable to external events. The unreliability of the data points in a streaming environment is one of the key challenges for working with data streams [29]. Here the general term *uncertain* is used to describe any element that cannot be relied upon with complete confidence; however it has many facets like Uncertainty (the fact is uncertain, i.e., the attribute value cannot be measured with sufficient confidence), Imprecision (the information is not as specific as it should be), Vagueness (including elements that are inherently vague), Inconsistency (more than one mutually exclusive assertion), and Ambiguity (lack of complete semantics) [30]. On the contrary, existing outlier detection techniques assume data points' values to be correct; therefore, the dissimilarity between two data points can be easily measured by distance (Euclidian or Manhattan) or cosine similarity. However, distance or cosine similarity fails to measure the similarity/dissimilarity between two data points if they are uncertain; and thus outlier detection schemes that use such measure of similarity would inherently fail to detect outliers for data streams [22, 29].

Requirement 7. An outlier detection technique for data streams should use a similarity metric that can measure the similarity between two uncertain data points.

Moreover the uncertainty may arise because some data points could be entirely missing or out-of-date in a data stream which is referred to as imperfection by Stonebraker et al. [22]. Data points may arrive late or even entirely fail to arrive in a data streaming environment. An outlier detection technique must process the existing data regardless of the fate of the failed data points. Consider an example where a senor produces one data point every hour and an outlier detection technique that requires previous three hours of data to decide the outlier-ness of the current data point. Now if the previous two hours of data failed to arrive before the current data point, the fate of the current data point must be decided based on whatever data to which the technique has access (i.e., the current data point and the data point that arrived three hours ago since other two data points in the middle are missing). The problem can be worse if a data point arrives out-of-date. In that case the out-of-date data point must be processed based on its own temporal context. To the best of our knowledge, no existing outlier detection technique processes out-of-date data based on their temporal context. Comparing a data point with other data points having different temporal contexts to identify outliers would produce erroneous results. For example, if the two missing data points arrive some time later, the outlier detection technique needs to process them based on the three hours of data points that are supposed to arrive before them.

Requirement 8. The outlier-ness of an out-of-order data point should be decided by comparing it with the data points that have the same temporal context as that of the out-of-order data point.

Multi-dimensionality

Although multi-dimensionality is not a data stream specific issue, it is worth discussing because of its impacts on outlier detection. Measuring the similarity of a data point to other data points in the dataset is a crucial part of outlier detection because an unusual data point must have very few data points that are similar to it in the dataset. Many outlier detection techniques use Euclidian or Manhattan distance to measure the similarity between data points [31, 32], but Euclidian distance becomes qualitatively meaningless to represent such similarity and causes instability of nearest neighbor for a high number of dimensions [33, 34]. This is because the distance between two similar data points and the distance between two non-similar data points are approximately equal for a high number of dimensions, which in turn makes distance-based outlier detection algorithms less effective. Furthermore, many algorithms use data density function, but the multi-dimensional data density space grows exponentially with the number of dimensions; hence data density function cannot be computed easily [35]. Arguably, outlier detection in a multi-dimensional data stream can be seen as outlier detection in a set of single dimensional data streams, but this approach is fundamentally flawed because it handles all dimensions independently and fails to address the correlation among dimensions.

Requirement 9. An outlier detection technique for data streams should use a similarity metric that can measure the similarity among the data points with a large number of dimensions.

A successful outlier detection technique for single data stream should address the aforementioned requirements. For related data streams, besides those requirements, additional ones arise, which we discuss in the next paragraphs.

Cross-correlation

Multiple data streams produce multiple data points with explicit or implicit timestamps. Some outlier detection techniques assume that data points from multiple data streams should be close to one another [32, 36] and a data point is an outlier if it is far from other data points from other streams at any point in time. This definition is too restrictive because in many applications, such as Chlorine monitoring and temperature monitoring in the same building, data points from different streams are cross-correlated although their values could be far from one another [37]. A data point is considered to be an outlier if it violates the expected cross-correlation (nonconformist to other values). The temperatures from different cities from different states could be very different from one another but they could be related. In order to detect outliers, the outlier detection technique should find the cross-correlation among the data points from different data streams and compare them to the data points based on their explored/expected cross-correlations.

Requirement 10. The outlier detection technique should be capable of detecting outliers that are non-conformist to the other data points with respect to their relationship with other data points.

Asynchronous Data Points

Data sources in a multiple data streams application may be independent of one another; and thus they may generate data points with different arrival rates. These data points are

asynchronous [38]. In order to identify whether a data point is an outlier or not, the outlier detection technique has to choose an appropriate temporal context not only for the same stream but also for other streams. The data point with a predefined temporal context can be compared with other data points with the same temporal context in other streams for outlier detection. Since data points do not arrive synchronously, it is not only difficult to choose an appropriate temporal context from all data streams, but also, at a particular timestamp, some data points from some data streams may be present while those from other data streams may not, and thus might be considered as missing data. To the best of our knowledge, no existing outlier detection technique is designed to tackle this kind of missing data.

Requirement 11. The outlier-ness of a data point should be decided as it arrives, minimizing the effect of missing data due to asynchronous processing.

Furthermore, the asynchronous behavior of multiple data streams hinders synchronous processing. Some outlier detection schemes for data streams assume all data points from multiple data streams to arrive together [32, 39] and then process them together to detect an outlier. However, in practical situations, it is difficult to achieve synchronization for different data sources [6]. Moreover the processing of a data point cannot be delayed and wait for other data points from other data streams to arrive. Therefore the data points from multiple data streams may need to be processed and outliers may need to be detected asynchronously. However, in that case it would be extremely difficult to exploit the cross-correlations among the data points from multiple streams. If an outlier detection technique ignores the cross-correlations completely, it

will fail to exploit the advantage of having multiple data streams, and might produce less accurate results.

Requirement 12. The outlier detection technique should have the capability of learning cross-correlation among streams and detecting outliers based on the learned cross-correlation asynchronously.

Dynamic Relationship

The cross-correlations among the data points from multiple data streams may vary over time and this dynamic relationship is due to two phenomena: (1) asynchronous behavior and (2) concept drift. The data point from multiple data streams have temporal correlations which may vary with the varying time differences among the data points. Imagine two temperature sensors are mounted in a close proximity to detect temperatures. One sensor produces one data point every 3 hours and another sensor produces one data point every 5 hours. The time difference between the most recent data points from the two sensors may vary from 0 to 3 hours. In that case the temporal correlations among them also vary over time. However typical outlier detection relies on comparing a data point to its cross-correlated data points; if the relationship changes over time, the cross-correlation among the data points from multiple streams has to be monitored continuously.

Concept drift is the second driving factor for dynamic relationship. If concept drifts occur independently in multiple data streams, the correlations among the data points from multiple streams vary as well. Thus the relationships among the data points become dynamic.

Requirement 13. An outlier detection technique for multiple streams should continuously monitor their cross-correlation and compare a data point with only other cross-correlated data points to decide its outlier-ness.

Heterogeneous Schemas

In a multiple data streams application, different data streams might have different schemas [38]. Comparing multiple data points with different schemas is a complicated problem and the definition of outlier is even vaguer in that case. By definition, a data point is an outlier if it is a non-conformist compared to other data points, that is, if it has a value considerably different from those of other data points. However, if we consider a heterogeneous set of data points where all data points have different attributes, it becomes intrinsically difficult to identify which data point is a non-conformist. Imagine two data streams with one producing temperature and another producing humidity of any location. The direct comparison of temperature to humidity does not make sense and, hence, neither of them can be detected as an outlier on the basis of the other. Nonetheless, it is perceivable that temperature and humidity might have a correlation between them and the observed values of temperature and humidity might violate the predefined or previously traced/predefined correlation and, therefore, one of them is an outlier. Although intuitively this kind of heterogeneous comparison may produce meaningful outliers, it requires a new definition of outliers unlike what we have seen before.

Requirement 14. An outlier detection technique for multiple data streams should be able to compare data points with the same or different schemas in order to detect outliers.

Any effective outlier detection technique for data streams has to meet the above requirements. While research on other topics for data streams has been proposed, such as system design [40, 41], query processing [19, 42, 43, 44], and data mining [2, 45, 46], very little research has been done for outlier detection. To fill the gap, in this dissertation, we propose two outlier detection techniques for single and multiple data streams, called Orion and Wadjet.

6. Contribution

Outlier detection is an integral part of any data acquisition process. Due to the lack of a true online outlier detection technique, real applications often use offline outlier detection techniques or manual processes [6]. The outlier detection techniques that exist in the literature mostly deal with the regular data model, which are not suitable for outlier detection in data streams. A very few outlier detection techniques for data streams exist in the literature [26, 31, 36, 47, 48]; but they either do not consider all the issues for data streams or require extensive human interventions. Some outlier detection techniques for multiple streams assume all the data points at any specific time period from multiple streams are equal which we find too restrictive. In this work, we propose two outlier detection techniques which tackle the issues of data streams. The first technique, Orion, is designed for single data stream and the second technique, Wadjet, is designed for multiple data streams. Both of our outlier detection techniques address the research issues regarding their respective data streams.

Orion is designed for single stream; however Orion is also applicable for the application with multiple streams where each of them is independent from one another. For each

stream, when a data point arrives, Orion identifies the data point to be an outlier if it is considerably different from the other data points Orion already received from the same stream. As data points are multi-dimensional, to detect outliers, Orion uses the concept of *projected dimension* (called p-dimension). Orion finds an appropriate p-dimension along which the outlier nature of a data point would be revealed and computes a data density function along that dimension. The data density function used is computed online and incrementally and considers the uncertainty, transiency, temporal relation among the data points and varying data distribution. Orion identifies a data point to be an outlier if it is considerably different from other data points along any p-dimension.

The second technique, Wadjet, is designed for multiple streams where the data points from one stream may or may be related to those from the other streams (we call it cross-correlation). Wadjet captures the cross-correlation among the data points from multiple streams that arrive at the same time. A data point is considered as an outlier if it shows non-conformist behavior to the other cross-correlated data points. Wadjet computes the cross-correlation among the available data points at any point in time; hence it works for an asynchronous set of data points from different streams. Wadjet is a two-phase algorithm: in the first phase, it uses Orion and detects outliers based on the temporal correlations of data points from the same stream, and in the second phase, it detects outliers based on the cross-correlations, if any, among the multiple streams. Wadjet continuously monitors the cross-correlations among the streams to effectively handle the dynamic relationships among the streams.

To the best of our knowledge, no technique for data stream outlier detection existing in the literature considers all the issues of data streams like our techniques, Orion and Wadjet.

In this dissertation, we also present the complexity analysis in terms of . time and space for Orion and Wadjet. , In addition, we report extensive experimental studies comparing our proposed techniques with the state of the art algorithms in terms of accuracy and execution time using real and synthetic datasets. In almost all cases, irrespective of datasets, Orion and Wadjet outperform existing techniques.

7. Organization

The rest of the dissertation is organized as follows: Chapter II reviews the existing work related to outlier detection with a focus on data streams. Chapter III describes our approaches and their implementations. Chapter IV presents the analytical results as well as the experimental results studying the performance of our approaches. Finally Chapter V provides conclusions and future research directions.

CHAPTER II
LITERATURE REVIEW OF OUTLIER DETECTION TECHNIQUES

Data stream management is a relatively recent research area compared to outlier detection; very few of the existing outlier detection techniques address the dynamic characteristics of data streams. Most of the outlier detection techniques for data streams are adopted from the ones for regular data. In this chapter we present some of the existing outlier detection techniques for data streams as well as regular data. These existing techniques can be classified into six categories: distance-based, density-based, sliding windows-based, auto-regression-based, statistics-based, and clustering-based. We discuss each of these categories with its state of the art representative techniques in this section.

1. Distance-Based Outlier Detection Techniques

Distance-based outlier detection techniques use distance to measure the similarity between two data points. A data point is defined as an outlier if it does not have enough similar data points.

1.1. Knorr and Ng's Distance-Based Outlier Detection

The first distance-based outlier detection technique was proposed by Knorr and Ng [23]and is very popular for outlier detection. According to Knorr and Ng [23, 49], a data point D is called (ρ, r)-outlier if D has a smaller number of neighbor data points within a radius r than a user-defined minimum number of neighbors ρ. The data point is called

a distance-based outlier with respect to the radius r and the number of neighbor data points ρ [23, 49]. According to these techniques, an outlier has fewer than ρ number of neighbor data points within the radius r.

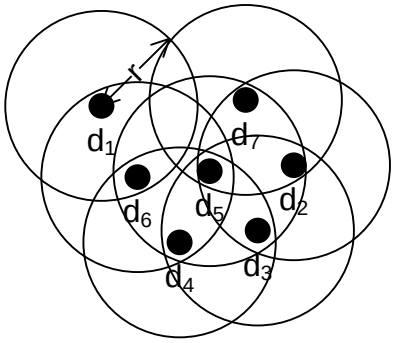

Figure 1. A typical distance-based outlier

Figure 1 illustrates the distance-based outlier technique where $D_1, \ldots, D_7$ are seven data points. For $\rho = 2$ and a given r, each data point in Figure 1 has at least 2 neighbor data points within the radius r except D_1; hence D_1 is called a $(2, r)$-outlier or distance-based outlier. The distance-based definition of outliers is very popular and unifies statistics based outlier detection using proper choice r and ρ [31]; because it does not assume any specific kind of data distribution [5, 23].

This technique does not always require pair-wise distances; it can also be defined based on data distribution or probability density distribution of data values. Knorr and Ng [49] showed the relationship between distance-based outliers and data distribution. Instead of computing the true neighbor count of a data point, this approach computes the neighbor density obtained from the data distribution. Given a data distribution function (in short, data distribution) $f(x)$, a data point D with value v and a radius r, the

neighbor density $N(v,r)$ is defined as $N(v,r) = \int_{v-r}^{v+r} f(x)dx$. The data point D is an outlier if the neighbor density is lower than the user-defined neighbor density q; hence, D is an outlier if $N(v,r) < q$. The data point is a distance-based outlier based on the given data distribution. In most cases the values q and r are taken as user-defined parameters.

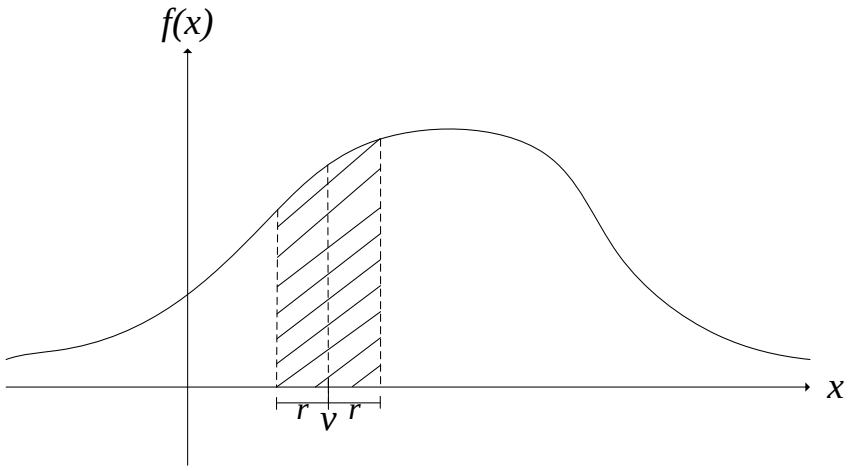

Figure 2. Distance-based outlier detection using data distribution function

Figure 2 shows the basic idea of distance-based outliers based on the data distribution function. We integrate the data distribution function $f(x)$ from $v - r$ to $v + r$ to obtain the neighbor density within the specified radius. By comparing the obtained neighbor density with the user-defined maximum density q, we identify the outliers. The maximum and minimum density could be 1 and 0.

1.2. Detecting Current Outliers: Continuous Outlier Detection over Time-Series Data Streams (DB-Outlier)

DB-Outlier is a continuous distance-based outlier detection algorithm proposed by Ishida and Kitagawa [32]. This algorithm detects distance-based outliers given user-defined distance r and minimum neighbor count k; a data point is an outlier if it has

fewer than k neighbors within distance r. DB-Outlier adopts the idea of cell-based outlier detection proposed by Knorr, Ng and Tucakov [50] and takes the cell based outlier detection to the next level called "cell based outlier detection for data streams." This algorithm assumes multiple data streams for outlier detection, compares the data points across the data streams and detects distance-based outliers.

If the number of data streams is N, it assumes there exist N data points $D_1^t, D_2^t, \dots, D_N^t$ at any particular point in time t. Each stream produces one data point at time t, $D_1^t, D_2^t, \dots, D_N^t$ are the data points from streams $S_1^t, S_2^t, \dots, S_N^t$ respectively. To detect the outliers D_i^t is compared to all D_j^t, where $1 \leq j \leq N$ and $i \neq j$, so outliers have been detected among $D_1^t, D_2^t, \dots, D_N^t$. We will discuss the DB-outlier in a step-by-step manner.

Step 1: DB-Outlier constructs a p dimensional grid structure in a p-dimensional hyperspace where each cell has a diagonal of size $\frac{r}{2}$ (length of a side $l = \frac{d}{2\sqrt{p}}$). $X_1, X_2, \dots, X_p$ are called axes and each cell in the grid is represented by $C_{x_1,x_2,\dots,x_p}$ where x_i is the value of i-th axis X_i.

Step 2: DB-Outlier defines two types of neighbor of a cell called L_1 and L_2. L_1 neighbors are adjacent cells. The maximum distance between two data points within the same cell is r. DB-Outlier defines L_2 in a similar fashion. L_2 includes all data points within the cells that can be within r distance from a cell.

Step 3: At $t = 1$, DB-Outlier distributes the data points into appropriate cells based on their values. The number of data points at any particular cell is defined by n. The

number of data point in L_1 neighbor cells is denoted by n_1 and the number of data points in L_2 neighbor cell is represented by n_2.

Step 4: In this step, DB-Outlier colors the cells based on the number of data points each cell and its L_1 and L_2 neighbors have. A cell is colored red if $n > k$ where k is the minimum neighbor count defined by the user. A cell is colored pink if $n + n_1 > k$ unless it was marked red before. A cell is colored yellow if $n + n_1 + n_2 \leq k$ and not already otherwise. Rest of the cells is colored as white. Interesting remarks can be made based on the color of the cell. No data point belongs to red and pink cells are outlier because they have more than k neighbors within r. All the data points belong to yellow cells are outliers because they don't have k number of outliers within r.

Step 5: The rest of the data points (data points that belong to white cells) can be outliers or inliers. DB-Outlier calculates the distance between a data point D_t and each of the data points x that belong to L_2 neighbor cells of the cell to which D_t resides. If n' is the number of data points x that has the distance lower than r, then D_t is an outlier if $n + n_1 + n' \leq k$; otherwise it is an inlier.

Step 6: DB-Outlier uses differential processing to detect the outliers in subsequent time steps (for $t \geq 2$). Ishida and Kitagawa [18] argue that data values for streams do not change very often, but rather they are stationary. Every time a set of data points arrives DB-Outlier identifies the data points that changed from the previous data points and process them.

Step 7: DB-Outlier re-distributes the data points into the cells again and re-color the cell based on its new L_1, L_2 neighbors and find the outliers within the data points the values of which have changed.

DB-Outlier processes a set of data points coming from multiple data streams all together and detects the outliers among them based on their pair-wise distances. This is online processing of data streams as they come by and only when their data points are changed from the previous time step; thus the processing is incremental. DB-Outlier does not store any data point for future processing, thus it maintains transiency of data streams as well. DB-Outlier processes round by round and the previous data rounds may affect efficiency but does not affect accuracy, thus the concept drift of a single data stream may not affect DB-Outlier.

Although DB-Outlier maintains the transiency very well, it never exploits temporal correlations among the data points for accuracy. Interestingly, DB-Outlier is not affected by concept drift of a single stream. This is because it identifies whether a data point D_t is an outlier by comparing it with other data points from other data streams only. So if a concept drift occurs in one stream, it could be different from data points from other streams, in that case it is not necessarily an outlier.

The biggest assumption DB-Outlier makes is fixed arrival rate. DB-Outlier assumes all data points from all streams come together and they have the same arrival rate. This assumption requires synchronization among the data streams which is very difficult to achieve and maintain [6]. This kind of architecture is popular among some sensor

networks but, in general, it is difficult to maintain synchronization for distributed objects.

DB-Outlier incorporates multi-dimensional objects but Euclidian distance for multi-dimensional data is not very effective to discriminate inliers from outliers. Moreover, DB-Outlier works for homogeneous data streams only as it does not work with data points of different data types. DB-Outlier does not address the uncertainty either since it asserts full confidence on every data point. The implicit assumption for DB-Outlier is that every data point from multiple streams should have the same value. However, this assumption is very restrictive for data streams. Data points from data streams could be correlated but may not be exactly the same. Moreover, due to dynamic distribution of data streams, the relation among the data points from multiple streams changes over time. Hence this kind of rigid assumption on data points makes the application of this technique limited.

1.3. Efficient Algorithms for Mining Outliers from Large Data Sets (EAMO)

EAMO is a distance-based outlier detection approach, however instead of specifying the distance and minimum nearest neighbor k, it defines the outliers differently [51]. Given a value for minimum nearest neighbor k, for each data point EAMO finds the distance for the k-th nearest neighbor. The top n data points with the highest distance for the k-th nearest neighbor are identified as outliers. Instead of using a nested loop algorithm to find the top n outliers, EAMO proposed an efficient partition based algorithm for outlier detection. The next following paragraphs discuss EAMO in a step by step fashion.

Step 1: EAMO partitions the entire dataset into small groups using the clustering algorithm BIRCH [52]. Each group is called a partition, P.

Step 2: EAMO computes the k-th nearest neighbor distance for each data point in each partition. The distance to the k-th nearest neighbor for data point D is denoted by $Dist^k(D)$. Once the $Dist^k(D)$ is computed for all data points in a partition, EAMO computes the upper and lower bound *P.upper* and *P.lower* for the k-th nearest neighbor of each partition P such that $P.upper \geq Dist^k(D)$ and $P.lower \leq Dist^k(D)$ where D is a data point in a partition.

Step 3: EAMO identifies the candidate partitions that may contain outliers in this step. Let us assume $minDkDist$ is the lower bound for the n outliers having the maximum k-th nearest neighbor distance. So the partitions with the upper bounds smaller than $minDkDist$ cannot contain any outliers; hence, the candidate partitions are only the partitions that have upper bounds greater than $minDkDist$, meaning the partitions with $p.upper \geq minDkDist$. The $minDkDist$ can be calculated from the lower bound of a partition such that $minDkDist = \min(P_i.lower)$ where P_i is the i-th partition and P_i has at least n data points.

Step 4: EAMO processes the data points in the candidate partitions only. In order to compute the $Dist^k(D)$ for data point D in candidate partition P, EAMO needs to consider only the neighboring partitions that are within distance $P.upper$. Thus, EAMO considers each candidate partition in batch and finds the top n outliers having the maximum k-th nearest neighbor distance.

Instead of relying on two independent parameters, neighbor distance and minimum neighbor count, EAMO relies on the k-th nearest neighbor distance. This approach is a little bit difficult for traditional distance-based outliers. In many cases, working with two independent parameters like neighbor distance and minimum neighbor count for distance-based outliers is very difficult. Hence, EAMO solves that problem.

The efficient partition based algorithm is significantly faster than the traditional nested loop algorithm. EAMO is flexible enough to incorporate any kind of distance function and clustering algorithm.

This algorithm is not designed for data streams and it is hard to re-engineer this algorithm to make it work for data streams. This is because this algorithm requires multiple passes through the datasets, hence it is not suitable for outlier detection for data streams. In addition, this algorithm does not handle other data streams issues like transiency, concept drift, infiniteness, etc. Hence it is completely inapplicable for data streams.

Moreover, EAMO uses a popular distance metric, such as Euclidian and Manhattan distance, to partition the data points. However, Euclidian and Manhattan distances are useless for outlier detection for high dimensional space, hence EAMO can only work for a small number of dimensions.

1.4. Distance-Based Outlier Detection for Data Streams (DBOD-DS)

Sadik and Gruenwald proposed a distance-based outlier detection technique, DBOD-DS, for data streams [47]. To identify whether a data point D_t is an outlier, DBOD-DS, instead of computing the neighbor count for D_t, uses a data density function to capture

the trends of the data points and calculates the neighbor density of D_t. The neighbor density of a data point in data streams is analogous to the neighbor count of a data point in regular (non-stream) data. They designed an effective data density function that handles transiency, uncertainty, concept drift and infiniteness of data streams. DBOD-DS consists of the following steps:

Step 1: DBOD-DS creates a data density function which is updated as each new data point arrives. DBOD-DS computes the neighbor density of the data point D_t within the user defined distance r by integrating the data density function within distance r.

Step 2: DBOD-DS identifies D_t as an outlier if D_t's r neighbor density is smaller than the user defined minimum neighbor density; otherwise DBOD-DS identifiers D_t as an inlier.

Step 3: DBOD-DS updates the existing data density function so that it always represents the most recent trend of the data. DBOD-DS computes the kernel value of D_t using a kernel function, which distributes the weight of occurrence of D_t into its neighboring values.

Step 4: DBOD-DS updates the existing data density function by adding the kernel value of D_t to the existing data density function. In order to give the highest weight to the most recent data point, DBOD-DS decays the weight of the older data points in the data density function and adds the new kernel value of D_t to the data density function.

Although DBOD-DS addresses all the characteristics of data streams, it is designed for single dimensional data only. In does not work for multi-dimensional data points.

Moreover, it assumes multiple data streams to be always independent of one another and, hence, cannot be applicable to related data streams.

1.5. Advantages

The underlying idea of distance-based outlier detection techniques is to separate a data point which has very few data points within a close proximity from other data points. According to Hawkins' definition [53] an "outlier would be an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism." The definition of distance-based outliers matches Hawkins' definition very well.

Any discordancy test for outlier detection can be modeled using distance-based outliers [23], which means any statistical technique can be replaced by a distance-based outlier detection technique. Knorr and Ng also proved that distance-based outlier detection techniques can generalize auto-regression-based outlier detection techniques as well [23]. Thus, a distance-based outlier detection technique can be changed to any other technique by changing its two parameters, radius and neighbor density. Many approaches use a distance-based outlier detection technique as a basis and perform additional tasks for suitable data representation [32, 54] and knowledge discovery [36, 55].

1.6. Disadvantages

The first drawback of distance-based outlier techniques is that they require the concept of distance/proximity within two data points like clustering algorithms where data points are clustered into some groups based on the similarities between them [56, 57],

but clustering techniques analyze data with respect to a global view, whereas the distance-based approaches consider the nearest data points. The second and very important drawback is that the performance of distance-based outlier detection techniques relies on the user-defined radius r and neighbor count/density q.

Distance-based outlier detection works for only a small number of data dimensions [58]. For a large number of data dimensions, the distance between two data points fails to measure the similarity among the data points and hence, distance becomes unusable for similarity measurement [34, 58]. Zheng pointed out that this phenomenon is true regardless of definition of local neighborhood or k-nearest neighbor and neither of them makes any sense for high dimensional data [58]. Practically, the degree of outlier-ness based on distance metric in a high dimensional space is the same for all data points. Thus distance-based outlier detection has no use for high dimensional datasets.

Distance-based outlier detection was designed for regular data and is not suitable for data streams due to two reasons. First, it requires at least two passes for distance calculation or data distribution function calculation; therefore, we cannot use distance-based approaches incrementally for unbounded data streams. Second, distance-based outlier detection based on a data distribution function accommodates the uncertainty issue of data, but it assumes a fixed data distribution; thus is cannot be used directly for a dynamically changing environment like data streams.

2. Density-Based Outlier Detection Techniques

As reviewed in Section 1, distance-based outlier detection is very popular and statistically sound but it requires two user-defined parameters, ρ and r. These two

parameters vary a lot based on data; if the data points are far from one another, the radius r becomes large and the neighbor count ρ becomes small and vice versa. It is often becomes difficult for the user to adjust the parameters without having explicit knowledge about the dataset or the algorithm. To tackle the problem of user-defined parameters, density-based outlier detection techniques were proposed. Below we review the representative ones.

2.1. LOCI: Fast Outlier Detection Using the Local Correlation Integral

Papadimitriou et al. proposed LOCI, a density-based outlier detection technique [26]. LOCI uses a multi granularity based deviation factor (MDEF) for outlier detection. MDEF is computed by comparing the neighbor count of a data point D to the average neighbor count of all the data points in a local neighborhood of D. If D has a very few neighbors and the other data points in its local neighborhood have significantly more neighbors, then D is very likely to be an outlier; but if all the other data points have very few neighbors as well, then it is probably the common trend of the dataset, and thus D is not an outlier. Based on this fact, Papadimitriou et al. proposed a multi granularity based deviation factor [25, 59] which calculates the deviation of a data point. LOCI identifies D as an outlier if the MDEF value of D is three standard deviations apart from the average MDEF value of all the data points in the local neighborhood of D. The MDEF value of D depends upon the neighbor count of D and the average neighbor counts of all the data points; thus the radius does not affect the MDEF value very much [25, 59]. LOCI consists of the following steps:

Step 1: For a data point D, LOCI computes its number of neighbors.

Step 2: Given a local neighborhood radius r, LOCI computes the MDEF value of D by comparing the neighbor count of D to the average neighbor count of the data points in the local neighborhood of D.

Step 3: If the MDEF value of D is three standard deviations of MDEF apart from 0, D is identified as an outlier.

Step 4: If a data point is not identified as an outlier in Step 3, LOCI repeats Steps 2 and 3 with a different neighborhood radius r. This process continues until all neighborhood radiuses are checked. LOCI stops processing D once it is identified as an outlier for any neighborhood radius r.

LOCI is not designed for data stream applications; hence it does not address the transiency and temporal relationship among the data points. All the data points are treated equally and the temporal dimension is completely ignored in LOCI. LOCI is not a single-pass algorithm; so LOCI is not online or incremental.

LOCI does not address data uncertainty or concept drift. Once a neighborhood is established, the MDEF values of all the data points of a neighborhood are calculated based on the established neighborhood; therefore LOCI assumes a constant data distribution. Due to the above disadvantages, LOCI is not directly applicable to data streams.

2.2. LOF: Identifying Density-Based Local Outliers

Like LOCI, LOF [24] also uses the local density information in order to adjust the common trend of the data points. Instead of computing the neighbor count of D, LOF

computes the distance from D to the k-th nearest neighbor of D, which is called k-distance, for outlier detection. LOF defines reachability distance from k-distance which is not sensitive to statistical fluctuations of distances of data points from D. LOF computes the local reachability density of D by comparing the reachability distance of k to the reachability density of other data points within a neighborhood. Local reachability density is further used to measure the outlier-ness of a data point. The step-by-step procedure of LOF is as follows:

Step 1: For each data point D, LOF computes the k-distance of D, *k-distance (D)*, for a given value of k, which is defined as the distance between D and its k-the nearest neighbor.

Step 2: LOF computes the reachability distance of D with respect to other data points within its k-distance. The reachability distance between D and any other data point p is the Euclidian distance between D and p or the k-distance of p if the k-distance of p is greater than the Euclidian distance between D and p.

Step 3: For each data point D, LOF computes its local reachability density D, $lrd_k(D)$, which is defined as follows:

$$lrd_{MinPts}(D) = \frac{\left(\sum_{p \in N_{MinPts}(D)} reachDist(D,p)\right)}{|N_{MinPts}(D)|}$$

where $N_{MinPts}(d)$ is the number of data points within k-distance of d and $k = MinPts$. The local reachability density of D represents average reachability distance of D within its local neighborhood.

Step 4: For each data point D , LOF computes its LOF value, which is the average relative local reachability density and is defined as

$$LOF_{MinPts}(D) = \frac{\left(\sum_{p \in N_{MinPts}(d)} \frac{lrd_{MinPts}(p)}{lrd_{MinPts}(D)}\right)}{|N_{MinPts}(D)|}$$

Step 5: According to definition of the LOF, if D is deep inside the group of data points, its LOF value is close to 1 and therefore is very unlikely to be an outlier, and if D is an outlier, its LOF value should be considerably higher than 1. Thus based on the application, the user has to choose the cut-off limit for the LOF and a data point is identified as an outlier if it has a higher LOF value than the user-defined LOF value.

Although this algorithm defines the outlier-ness of a data point based on its local density only, it requires multiple passes through the dataset (k-distance, reachability distance, local reachability density and LOF all require multiple passes); hence it is not applicable for data streams as it would require the entire dataset to be available after the first pass.. Moreover, this approach also ignores the other data stream characteristics like uncertainty, concept drift, and infiniteness. The success of this algorithm depends on the effectiveness of the distance function that measures the similarity between data points.

2.3. Online Outlier Detection for Data Streams (A-ODDS)

A-ODDS is a density based outlier detection technique proposed by Sadik and Gruenwald [60]. A-ODDS makes use of two concepts Global Density Factor (GDF) and Local Density Factor (LDF) in order to detect the outlier-ness of a data point. The GDF of a data point D_t is the relative deviation of neighbor density of D_t with respect to the

average neighbor density of all data points and the LDF of D_t is the relative deviation of neighbor density of D_t with respect to the average neighbor density of the data points within the current concept. Since data streams are characterized by concept drift, A-ODDS also includes an approach to detect concept drifts. The data points within two consecutive concept drifts are considered as the data points that belong to the same concept. The details of the algorithm are as follows:

Step 1: When a data point D_t arrives, A-ODDS computes its neighbor density using the data density function discussed in the DBOD-DS algorithm (Section 1.4).

Step 2: A-ODDS computes D_t's global deviation factor, $GDF(D_t)$, where the global deviation factor of a data point is defined as the relative deviation of the neighbor density of a data point from the average neighbor density of all history data points with respect to the average neighbor density of all history data points.

Step 3: A-ODDS decides whether a concept drift has occurred or not. If a concept drift has occurred, A-ODDS updates the average neighbor density of the data points that belong to the same data distribution and computes the LDF of D_t, else it computes the LDF using the existing average neighbor density.

Step 4: A-ODDS computes the standard deviations of GDF and LDF. If the GDF or LDF value of D_t is greater than three standard deviations, A-ODDS identifies D_t as an outlier.

Step 5: A-ODDS updates the global average neighbor density and local average neighbor density using the neighbor density.

This approach shares the same drawbacks as DBOD-DS. The data density function and concept drift detection only work for single dimensional data streams. If the number of dimensions becomes large, this approach would not work. However, A-ODDS addresses other important characteristics of data streams: transiency, notion of time and infinity, uncertainty, and concept drift.

2.4. Advantages

Density-based outlier detection techniques are more sophisticated than distance-based outlier detection techniques. Density-based outlier detection techniques consider the local density of the data points. The local density of a group of data points is high if the data points are dense compared to the local density of data points that are sparse. This enables the techniques to work for a wide range of datasets. This is because, regardless of the size and sparseness of the data points in a dataset, density-based techniques work with a subset of data points that belong to a local neighborhood of a data point D_t (a set of data points that are in a close proximity from D_t). Thus, density-based approaches are applicable for many datasets and local density factor is adjusted according the sparseness of a dataset.

Moreover, these density-based approaches work even if the sparseness of the data points varies across the dataset. This is because the outlier-ness of a data point is detected based on the local neighbor of data points only. So, if a dataset consist of several groups of data points and the sparse-ness of each group t varies, the density-based approaches determine the outlier-ness of a data point based one the sparse-ness of the data points that belong to the nearest group from the data point.

2.5. Disadvantages

Density-based approaches are computationally more expensive compared to distance based approaches. This is because density-based approaches require local neighborhood computation which needs pair-wise distances between every pair of data points. On the top of that, neighbor density or reachability distance of D_t is compared to those of all other data points that are within local neighborhood of D_t. Hence, density-based approaches are computationally expensive. Moreover, popular density based approaches such as LOCI and LOF require multiple passes over the dataset; thus, it is difficult to re-engineer them for data streams. This is because multi-pass algorithms are not suitable for data stream outlier detection due to unavailability of entire dataset.

Moreover, one required step of the density-based approaches is computing the local neighborhood of a data point. Computing the local neighborhood requires Euclidian distance computation. If the number of dimensions becomes large, the Euclidean distance stops working as a similarity metric. In that case local neighborhood does not possess any significance and hence density based approaches collapse.

3. Sliding Windows-Based Outlier Detection Techniques

3.1. Overview of the Techniques

A sliding window holds the most recent subset of the data points [31]. It is temporary data storage for the data points in a data stream. At any time a sliding window can hold a fixed amount *w* of data points; typically the size *w* is defined by the user.

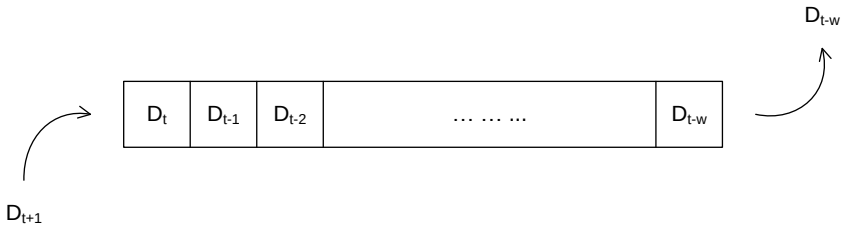

Figure 3. A typical sliding window

The oldest data point in the sliding window is removed as a new data point enters. Figure 3 shows a typical sliding window of size w where the newest data point is D_t and oldest data points is D_{t-w}. As the data point, D_{t+1} comes in the oldest data point D_{t-w} goes out. Once the data points are stored in the temporary storage, the application can access them randomly. So the applications that use sliding windows have a small subset of data for random access. Typically a sliding window based outlier detection technique identifies the outliers inside the window. More precisely, the data point which is identified as an outlier by a sliding window based technique does not conform to the other residents of the sliding window.

Once random access to a subset of the entire dataset is available, any random access algorithm can be used to detect outliers in that subset. Some outlier detection techniques for data streams use a sliding window and detect the outliers inside the window [13, 31, 36] using multi-pass algorithms. The algorithms may run periodically [54] or for every round [13]. In the next sections, we describe these algorithms.

3.2. Detecting Distance-based Outliers in Streams of Data (STORM)

Angiulli and Fassetti proposed a sliding window based outlier detection technique called STream OutlieR Miner (STORM) [31, 54]. STORM finds the distance based

outliers in the sliding window. The difference between STORM and a typical distance-based outlier detection technique is that STORM finds the outliers within a sliding window in lieu of an entire dataset. STORM consists of the following steps:

Step 1: As a new data point D_t comes in STORM creates a new data stream object that corresponds to D_t and stores the data stream object in the sliding window. If the sliding window is full, STORM discards the oldest data stream object and stores the newly created data stream object. The size of the sliding window is chosen by the user which represents the most interesting current subset of the data.

Step 2: STORM finds all the data stream objects in the sliding window within the radius r from the newly created data stream object. STORM maintains a list of references to all the preceding data stream objects within the radius r inside the newly created data stream object. Every data stream object has a counter which counts the number of succeeding data stream objects within the radius r inside the sliding window.

Step 3: Some of the data stream objects are chosen as pivot objects. A pivot objects has all the properties of a data stream object along with a list of distances from the pivot object to all other data stream objects in the sliding window. The pivot objects offer efficient range search.

Step 4: If the newly created data stream object is a pivot object, then a list of distances to all other data stream objects is created inside the pivot object. All the other pivot objects are updated upon creating a new data stream object.

Step 5: The outlier detection subroutine is invoked periodically based on a user-defined frequency to check each data stream object in the sliding window to see if it is an

outlier. For each data stream object, the outlier detection subroutine calculates the number of preceding data stream objects and the number of succeeding data stream objects. If the sum of these two numbers is greater than the user-defined minimum neighbor count of a data stream object, the data stream object is identified as an outlier; hence its corresponding data point is an outlier.

STORM considers the temporal characteristics of a data point in a data stream. Each data point remains in the sliding window for a fixed amount of time. As a data point is identified as an outlier compared to all the other data points in the sliding window, STORM automatically assumes all the data points in the sliding window are equally related; hence STORM addresses the temporal relationship of the data points as well.

STORM is not a true online incremental algorithm. STORM stores a subset of the data stream inside a sliding window and invokes the outlier detection subroutine periodically. Hence the fate of a data point is not confirmed as soon as the data point comes in. If the user mistakenly chooses the frequency of outlier detection greater than the sliding window size, some data points may be discarded without being checked for their outlier-ness.

STORM does not consider the uncertainty of the data points. STORM does not assume any data distribution; hence the concept drift of the data stream does not affect the algorithm directly, but the choice of the sliding window size should be such that the sliding window can accommodate all the data points in the current concept. Since the sliding window size is independent of the concept drift in this approach, we can conclude that STORM does not address the concept drift. Since STORM is not a true

online and incremental technique and does not address the issues of uncertainty and concept drift, it is not suitable for data stream applications.

STORM uses Euclidean distance in order to measure the similarity between data points. Thus, if the number of dimensions grows, Euclidean distance fails to represent the similarity among the data points and, hence, STORM would fail to identify outliers. Therefore, STORM is not suitable for multi-dimensional data points.

3.3. Automatic Outlier Detection for Time Series: An Application to Sensor Data (ODTS)

Basu and Meckesheimer [13] proposed an outlier detection algorithm for time series when the data points are difficult to model. They argue that the data points closer in time are more likely to be correlated. Hence they try to use the data points which are closer to each other to identify the outliers inside the sliding window. The algorithm is discussed here in brief; for our convenience we name the algorithm ODTS.

Step 1: ODTS maintains a sliding window of a user-defined size to hold a finite subset of the data points for outlier detection. As each data point D_t comes in, ODTS computes the median of the data points in the sliding window.

Step 2: ODTS computes the absolute value of the distance between the median data point and D_t. If this distance is greater than the user-defined maximum distance threshold, D_t is identified as an outlier and ODTS replaces D_t in the sliding window with the median data point.

ODTS is a true online incremental algorithm and a very simple approach for outlier detection when it is difficult to model the data. The computational complexity is linear

with respect to the sliding window; hence it provides a very good result within a very short time. As ODTS maintains a sliding window, only recent data points are stored in the sliding window; hence ODTS addresses the temporal characteristics of data streams. Since only the recent data points participate in the outlier detection, it accommodates the temporal relationship as well.

ODTS asserts each data point with full confidence; hence ODTS does not accommodate the uncertainty of data streams. Although the sliding window summarizes the recent data rounds, it has no relation with the data distribution; as a result concept drift is also ignored in ODTS.

ODTS is designed for single dimensional data points. ODTS can only work for multi-dimensional data points if it considers individual dimensions separately. However, in that case it would fail to identify the outliers with infrequent combinations of dimensional values. Therefore ODTS is not suitable for multi-dimensional data streams either.

3.4. Online Outlier Detection in Sensor Data Using Non-Parametric Models (ODSD)

Subramaniam et al. [36] proposed an in-network outlier detection technique for hierarchical sensor networks. For convenience we name the algorithm, ODSD. ODSD is also based on density-based outliers. It identifies the outliers within the sliding windows of the individual sensors. ODSD assumes that the sensors are arranged in a hierarchical fashion. At the bottom level the children nodes identify the outliers in their respective sliding windows. In the second level the parent nodes identify the outliers in their

respective sliding windows. The sliding windows of the first level parent nodes accommodate the data coming from the respective children nodes. The next levels' parents work in a similar fashion. At the top level, the leader node collects data from all the sensor nodes and finds the outliers. The algorithm is discussed step-by-step here.

Step 1: The bottom level sensors store data in their respective sliding windows as soon as they collect data. The size of the sliding windows is defined by the user.

Step 2: Each sensor samples its sliding window and constructs a data distribution function for the sliding window. The neighbor density of the newly collected data point D is computed by integrating the data distribution function within a user-defined radius. If the neighbor density is lower than the user-defined minimum neighbor density, ODSD identifies D as an outlier. ODSD uses a kernel density estimator for data distribution computation; therefore, instead of increasing the frequency of occurrence of a value by 1, ODSD increases the frequency of occurrence of a value by p, and increases the frequency of occurrence of other values by a fraction of $1 - p$.

Step 3: All the base level sensors report the collected data to their respective parent nodes. A parent node constructs a new sliding window sampling all the data points received from its children nodes. Finally the parent node identifies the outliers using a similar algorithm to the one that the children nodes used for their outlier detection. All nodes use the LOCI algorithm to detect outliers. In the second level the parent node has to check only the data points that are marked as an outlier by the children nodes.

Step 4: ODSD goes up in the hierarchy using the same technique described in Step 3 until it reaches the top level. D is an outlier if the top level node identifies it as an

outlier. In order to reduce the communication cost, each sensor reports the data to the parent node when the data distribution changes instead of reporting the data every time. The data distribution is monitored by each sensor; the newly obtained data distribution is compared with the previous data distribution using Jenson-Shannon divergence [36]. If the distance (based on Jenson-Shannon divergence) between the two distributions goes beyond a user specified value, the child node reports data to the parent node.

The major advantages of ODSD are that it is a decentralized, in-network and online outlier detection technique. Since ODSD does not assert each sample data point with full confidence but rather distributes the probability of occurrence of a value to the neighbor values, ODSD addresses the uncertainty issue of data streams.

The algorithm uses a random sampling technique to sample the data points from the sliding windows of the leaf nodes. The sample size and the window size (the same sizes for all sensors) are both the user-defined parameters and the random sample algorithm samples the data points from the sliding window regardless of concept drift. Hence the obtained data distribution function may contain the data points from multiple data distributions if concept drifts occur. Therefore, the obtained data distribution may not always reflect the current data distribution.

The outlier detection algorithm at each level is linear with respect to the window size. Hence, for a multi-level sensor network, outlier detection takes place at different levels and the overall time complexity is the sliding window size times the number of levels of the sensor network. Furthermore there is a communication involved between two levels of nodes. The communication time is very high compared to the computational time.

Typically the sensor network communication is lossy [61]. ODSD requires extensive levels of communication among the sensor nodes. The higher level of communication, the higher level of error is introduced. Hence the success of this approach is still in question. ODSD works for single dimension data only. ODSD also uses a data density function, but if the number of dimensions grows, the density space grows exponentially with it. Hence, constructing a multi-dimensional data density function is not scalable at all. Thus, ODSD's application would be very limited to a small number of dimensions only.

ODSD also considers cross-correlation among the data points from multiple streams, but only for equality, meaning the values of the data points from multiple streams have to be equal. This assumption does not hold in many practical applications such as environmental monitoring [15], chlorine content monitoring [48], etc. and hence, ODSD is very limited in real applications. Finally, ODSD assumes synchronous and homogeneous data points from multiple streams which are even more restrictive.

3.5. Incremental Outlier Detection in Data Streams Using Local Correlation Integral (Stream LOCI)

Stream LOCI [62] is an outlier detection technique based on LOCI [59] (Section 2.1). LOCI is a density based outlier detection technique that considers the local density of a data point D to compute the MDEF of D. MDEF defines the outlier-ness of a data point. Stream LOCI modified LOCI in order to make it suitable for data streams. Stream LOCI introduces a sliding window to hold the most recent subset of the data points. Once a new data point D arrives, Stream LOCI updates the sliding window by replacing the oldest data point in the sliding window with D and adjusting the neighbor density of

D accordingly. Then it applies the same sequence of steps as LOCI does and detects outliers within the sliding window.

This is a very typical approach for detecting outliers for data streams by modifying outlier detection techniques for regular data points using a sliding window. However, this approach fails to illustrate the effective size of the sliding window as the size is decided on an ad-hoc basis. Moreover, Stream LOCI inherits other problems from LOCI such as its ineffectiveness for multi-dimensional data points.

3.6. Advantages

The three sliding window-based techniques discussed in this section have a few common advantages. A sliding window can be manipulated as each data point comes in, so the updating process of a sliding window is online and incremental. In general the update procedure requires discarding an old data point and storing a new one; hence the update process is computationally efficient. At the same time a sliding window contains the most recent subset of the dataset; so a sliding window based technique identifies the outliers based on the recent subset and addresses the temporal characteristic of data streams.

3.7. Disadvantages

The choice of outliers in a sliding window is very much dependent on the current residents of the sliding window. An outlier of a sliding window can appear as an inlier for a different choice of data points for the sliding window. This problem is severe for data streams because they change over time and an outlier for a particular window may appear as an inlier in another window; hence the notion of an outlier in a data stream

with respect to a window is not very concrete. Figure 4 illustrates this point. In Figure 4 (a) the sliding window contains the data points from D_t to D_{t+5}. The data point D_{t+4} seems to be an outlier with respect to the other data points in the sliding window because it is far from them, but in Figure 4(b) the data point D_{t+4} seems like an inlier because in Figure 4(b) the data point D_{t+4} does not look far away with respect to the other residents of the sliding window. In summary, the outliers in a sliding window are very subjective to the residents of the sliding window; therefore, selecting the residents of a sliding window is the most challenging part of a sliding window based approach.

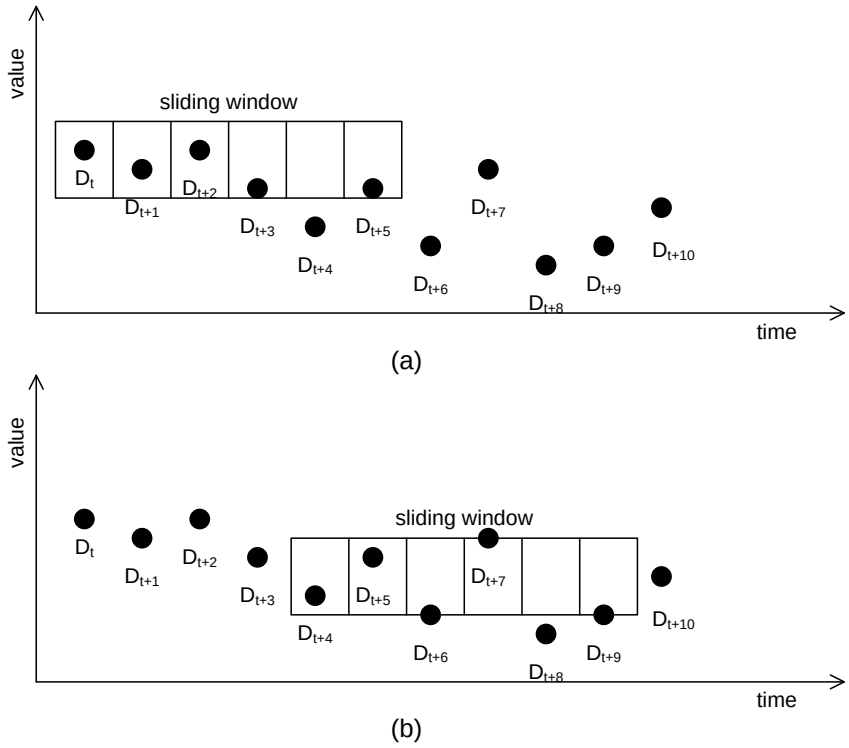

Figure 4. Choice of data points for the sliding window

In brief, the accuracy of a sliding window based technique depends on the size of the sliding window, how often the outliers are detected in the sliding window, etc. [61]. So far these parameters are selected by the user on an ad-hoc basis. Nevertheless, an inlier

can be shown as an outlier by changing the window size [13, 61]; thus an outlier detection technique which uses a sliding window works well if the window size is chosen appropriately. Moreover, each technique interprets the window size in its own way. For example, the significance of the sliding window in the technique proposed by Basu and Meckesheimer [13] is very different from that in the technique proposed by Anguiulli and Fassetti [31] or Subramaniam et al. [36]. Consequently the optimal performance for each algorithm requires a different sized sliding window (e.g., the optimal performance for [13] can be achieved when the sliding window size is small but the optimal performance for [31] can be achieved when the sliding window size is very large). In most cases the sliding window size selection requires either the details about the technique or ad-hoc trial and error modifications.

4. Auto-Regression-Based Outlier Detection Techniques

4.1. Overview of the Techniques

Auto-regression-based techniques for outlier detection are very popular for time series outlier detection [10]. Some outlier detection techniques for data streams adopt auto-regression [63, 64, 65]. Most of the auto-regression based techniques work similarly in that they establish a model based on the data points received so far. As each data point comes in, it is compared with the established model and a metric is obtained based on the comparison (e.g., distance from estimated value, variance, maximum likelihood ratio, etc.). The metric often represents the outlier-ness of the data point. If the established metric for the data point goes beyond a certain limit (*aka* cut-off limit), the data point is identified as an outlier.

Figure 5 shows a flowchart for a typical auto-regression based technique. Different techniques use different metrics and comparison methods [10]. Most popular approaches for time series data compare the data points with the predicted values. At each time step, the model predicts a data value called the predicted value and receives a value called the true value. If the distance between the predicted value and the true value is greater than the cut-off limit, the data point is called an outlier. Myriad ways are available for building auto-regression models [66, 67] and various approaches design their own auto-regression models according to their data patterns (e.g., linear, quadratic and harmonic) and applications of data streams. In the next three sections we discuss three auto-regression based techniques for outlier detection in detail.

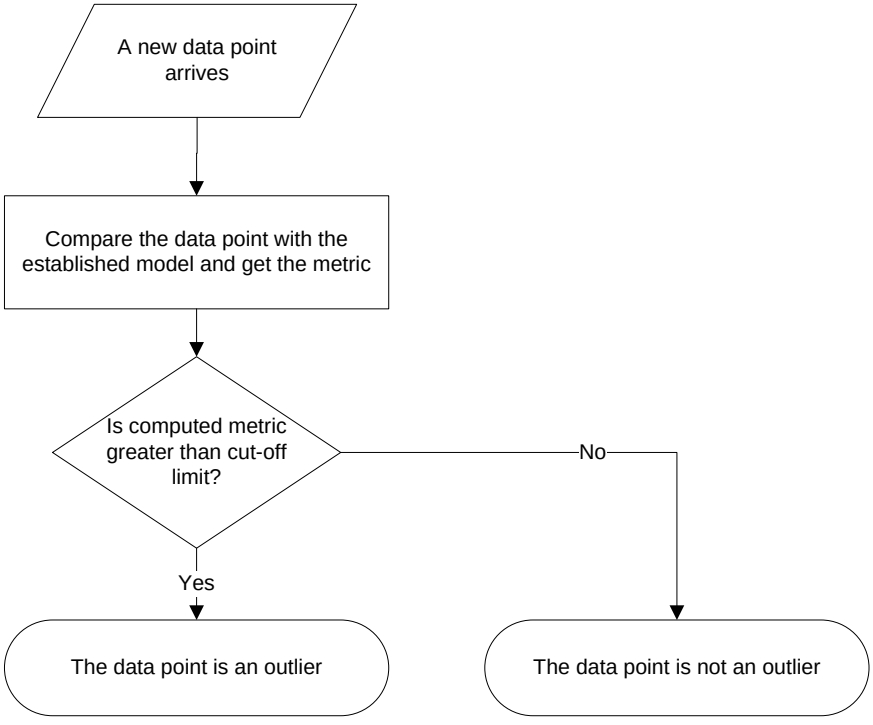

Figure 5. A flowchart for a typical auto-regression-based technique

4.2. A Kalman Filter-Based Approach for Outlier Detection in Sensor Networks (KFOD)

Shuai et al. [63] proposed an in-network outlier detection technique, called KFOD, for sensor networks. They assume the reading of a sensor has a temporal dependency. KFOD has two modules: (1) state transition module and (2) measuring module. The state transition module uses a first order auto-regression [66] model which uses the least squares method for coefficient approximation and the measuring module uses the other sensor readings to exploit spatial correlations. KFOD consists of the following steps:

Step 1: As each data point is obtained by a sensor, the sensor invokes the state transition module that estimates the current reading based on the previous readings using the first order auto-regression model [66].

Step 2: Each sensor collects the readings from its neighbor sensors. The measuring module measures the current reading of the sensor using the neighbor sensors' readings. They use inverse distance weighting (IDW) where the weight is inversely proportional to the distance of the neighbor sensors to measure the sensor reading from the neighbor sensors' readings [63].

Step 3: The two values collected from Step 1 (the approximated reading based on the previous readings) and Step 2 (the approximate reading based on the neighbor sensors' readings) are combined to produce the ultimate approximated value for the sensor. If the distance between the true value and the predicted value is greater than the user defined threshold, the data point is identified as an outlier.

KFOD is an online technique for outlier detection. It never stores the history data points but rather adjusts the coefficient of the modules based on the data points when they arrive; hence KFOD is incremental. KFOD discards a data point after updating the coefficients with it value; thus it preserves the transiency of the data points. KFOD uses the temporal relationship among the data points to update the coefficients.

KFOD does not assume any fixed data distribution and uses only the current data round and the next to the current data round. Hence, this approach is not vulnerable to concept drift; but the relation between the successive data rounds is kept fixed; thus, it does not address concept drift. KFOD does not address the uncertainty of data streams.

KFOD uses the first order auto-regression model to estimate the current reading. The efficiency of the first order auto-regression model is limited to linear trends. i.e., it works for linear changes of the data points; it fails to model more complicated relations among the data points. In addition, the relation between the consecutive data points of a data stream changes over time which is very difficult to model using the first order auto-regression; because the first order regression can only model linear changes. If relation between data points is quadratic or harmonic, the first order regression would not be able to model that.

KFOD assumes implicit communication among the neighbor sensors which is not possible in every sensor network. The spatial relationship among the sensors is hard to model because spatial modeling requires extensive knowledge and the relationship among the sensors. Moreover, the threshold selection for the maximum allowable distance between the predicted value and the true value depends on the accuracy of the

auto-regression model, the spatial relationship among the sensors and the combination of the outputs of the two modules. Hence, it is difficult for the user to select an appropriate threshold distance without trial and error.

4.3. Malicious Node Detection in Wireless Sensor Networks Using an Auto-regression Technique (ART)

Curiac et al. [64] proposed an auto-regression based anomaly detection technique for wireless sensor networks. They argued that the true values measured by a sensor have a deterministic component instead of a truly random component; therefore they used a time series analysis approach to explore the deterministic component of the values. The entire approach is executed in the base station and upon receiving a new data point, the base station compares the received data point with the predicted value. A data point is an outlier if the difference between the data point and the predicted value is greater than the user-defined threshold. We name this approach ART for our convenience. ART uses the fourth order auto-regression model [66] and consists of the following steps:

Step 1: As each data point is received at the base station, the auto-regression coefficients are updated recursively using the least squares method.

Step 2: ART predicts the current reading using the auto-regression model built inside the base station. The base station constructs an individual auto-regression model for each sensor and maintains an individual distance threshold for each sensor.

Step 3: The predicted value is compared with the obtained value. If the absolute distance between them is greater than the user-defined distance threshold, the sensor node is detected as an anomalous sensor node.

ART is an online, incremental approach for outlier detection for sensor networks. It also addresses the transiency characteristic of the data points. The temporal relationships among the data points are exploited by the fourth order auto-regression model but the coefficients are selected based on the entire history which does not necessarily reflect the updated relationships among the data points. The use of the entire history data points for coefficient selection does not reflect the recent trend of a data stream; thus ART does not address concept drift.

ART does not address the uncertainty of data streams. ART is also very similar to KFOD in that it is not vulnerable to concept drift but it does not directly address concept drift either.

Data streams are defined as infinite sequences of data points; hence one fixed threshold may not be appropriate for the entire life-time of data streams. Moreover, it is difficult to establish an individual threshold for each sensor.

4.4. Adaptive Methods for Activity Monitoring of Streaming Data (AMSD)

Puttagunta and Kalpakis [65] proposed a forgetting factor based recursive least squares algorithm for adaptive incremental model construction. This approach is very similar to the previous approaches except it identifies the changes of trends as well as outliers. We call it AMSD. AMSD works as follows:

Step 1: As a new data point is received, it is stored in a sliding window, the size of which is decided by the user. Once AMSD receives enough data points, it starts building an auto-regression model. The coefficients of the auto-regression model are

updated using the recursive least squares method as each data point is received. A predicted value is estimated before changing the coefficients.

Step 2: The absolute difference between the estimated value and the true value is computed. If the distance is greater than the maximum error threshold, the data point is identified as an outlier. If the distance is lower than the minimum error threshold, the data point is identified as an inlier. If the distance is in-between the minimum error threshold and the maximum error threshold, the data point is identified as a potential outlier. The potential outliers are stored in a window.

Step 3: For a potential outlier, if a change-of-point is detected within the previous τ data rounds, the potential outlier is not an outlier. τ is a user-defined parameter which outliers represents the minimum size of the change detection window. The number of potential outliers is computed within the last τ data rounds.

Step 4: If the number of potential outliers is lower than the maximum number of error overshoot, a user defined threshold, the data point is identified as an outlier. Once a data point is identified as an outlier, the latest update of the coefficients is discarded.

Step 5: If the number of potential outliers is greater than the user-defined threshold (maximum number of error overshoot) or the coefficients of the auto-regression model change beyond a user-defined threshold, AMSD identifies that a change has occurred and the first point of the window is identified as a change point and the rest are unmarked. A fresh auto-regression model is started from the change point.

AMSD is an online, incremental algorithm for anomaly detection. AMSD accommodates the transiency and the temporal relationships among the data points but

it does not address the uncertainty. AMSD is capable of detecting the change of data and uses only the data points from the current concept; hence it accommodates concept drift pretty well.

AMSD has at least twelve user-defined parameters and most of them do not have any physical interpretation to a user unless the user knows the algorithm and the application very well. It is very difficult for a user to select an appropriate value for each of them. Hence the success of this approach is questionable for real applications.

4.5. Advantages

Auto-regression-based techniques are typically computationally inexpensive. Most of the time the complexity of the model depends upon the history data used, the number of components in the auto-regression model and the number of data dimensions. Auto-regression based techniques are very appealing because of their affordable computational complexity. Moreover data stream applications are bounded by the arrival rate, i.e., every computation has to be done before the next data point comes in [4]. The efficient computation of an auto-regression based technique offers very lucrative execution time for each data round which makes auto-regression based techniques very good candidates for outlier detection for data streams or time series.

In some applications an outlier is replaced by an estimated value [13, 61]. An auto-regression based technique automatically provides an estimated value as a product. Moreover, a good auto-regression-based approach addresses the dynamic nature of the data points in a data stream which makes auto-regression based approaches very

popular for data prediction [68, 69]. They offer significant advantages over other techniques if the application requires a predicted value for an outlying data point.

4.6. Disadvantages

The success of an auto-regression model depends on the quality of the auto-regression model and the data pattern. If the data pattern shows linear changes, a linear auto-regression model would perform better; and if the data pattern shows harmonic changes, a harmonic auto-regression model would perform better [64]. Hence it is not easy to decide an appropriate auto-regression model without knowledge of the data pattern [64].

Moreover, the data pattern in a data stream is not constant [61]. Sometimes the data pattern in a data stream shows a linear trend, while it may show non-linear at other points of time. Hence any assumption about the data trend may not be appropriate forever for data streams. The explicit assumption about the data trends makes an auto-regression model less appealing for data streams.

Another noticeable flaw of auto-regression based techniques is their magic cut-off limits. The magic cut-off limit not only depends upon the data values but also depends upon the auto-regression model chosen and the efficacy of the auto-regression model. If the accuracy of the auto-regression model is very good, a small cut-off limit is good enough to differentiate outliers from inliers, whereas a large cut-off limit is necessary for a poorly performing auto-regression model. In most cases, the cut-off limit is expected as a user-defined parameter [63, 64, 65] and the parameter selection is not a trivial task for the users.

A multi-dimensional auto-regression technique is even harder to design. If the number of dimensions grows, it requires an extremely high number of data points to find an effective auto-regression model for multi-dimensional data streams. Hence, auto-regression-based approaches have limited applicability for multi-dimensional data streams.

5. Statistical Outlier Detection Techniques

5.1. Overview of the Techniques

Techniques based on statistic [10, 70, 71] and machine learning [27, 28] assume a fixed probability distribution for data values. Typically the data distribution is obtained by a training dataset and as each data point comes in, the data point is compared with the mean [10] or box plot [70] of the obtained data distribution. A data point is identified as an outlier if it is in the low probability region of the data distribution. It should be noted that although there are some techniques available in the literature, researchers from other disciplines like medical science and chemistry often use statistical based techniques.

5.2. Informal Identification of Outliers in Medical Data (IDMD)

Laurikkala et al. [70] studied the box plot based outlier detection technique, which we call IDMD, for vertigo and female urinary incontinence data. This technique is not directly used for data stream applications and consists of the following steps:

Step 1: The entire dataset (all data points in the dataset) is used to compute the five numbers: lower extreme (minimum possible value), lower quartile (75% of the data

points are higher than lower quartile), median, upper quartile (75% of the data point are smaller than upper quartile) and upper extreme (maximum possible value).

Step 2: IDMS defines two thresholds: (1) the lower threshold and (2) the upper threshold. The lower threshold = lower quartile – 1.5 x (upper quartile – lower quartile) and the upper threshold = upper quartile + 1.5 x (upper quartile – lower quartile).

Step 3: The data point is revisited and is identified as an outlier if its data value is smaller than the lower threshold or greater than the upper threshold.

IDMD is not designed for data stream applications. The algorithm is not online and requires at least two passes of the entire dataset for outlier detection. It is not incremental either as it requires the entire dataset for outlier detection. IDMD requires each data point to be present in the system until it finishes the first pass; so it does not preserve the transiency. Moreover, IDMD treats each data point similarly; thus it does not address the temporal relationship. It does not assume any data distribution but summarizes the data distribution using the five variables presented in Step 2 and assumes that the entire dataset follows the same characteristics; so it ignores concept drift as well. Since this approach is not designed for data streams, it does not address any of their issues.

5.3. Detection of Outliers in Reference Distributions: Performance of Horn's Algorithm

Solberg and Lahti [71] conducted a study to evaluate Horn's algorithm for outlier detection. Horn's algorithm is very similar to the previous algorithm IDMD, except that

it constructs a Gaussian distribution instead of calculating five numbers for the box plot. This algorithm has the following steps:

Step 1: The entire dataset is used to approximate the Gaussian data distribution with the presence of outliers in the dataset based on the maximum likelihood method.

Step 2: A lower threshold and an upper threshold are established using the same formula discussed in Step 2 in Section 5.2.

Step 3: Each data point is revisited to identify outliers. A data point is an outlier if it goes beyond either the upper threshold or the lower threshold.

Horn's algorithm is not designed for data streams; hence it does not address any of their characteristics. Even for regular data, Horn's algorithm does not perform as well as expected. Solberg and Lahti [71] argued that this is because every dataset cannot be summarized by the Gaussian data distribution; and Horn's algorithm only works if a dataset can be summarized by the Gaussian data distribution.

5.4. Anomaly Detection over Noisy Data Using Learned Probability Distribution (Eskin's Algorithm)

Eskin [27] proposed an anomaly detection algorithm for noisy data based on expectation maximization. This method assumes that the percentage of outliers is very low compared to that of inliers. It assumes that a dataset is a mixture of two types of data, inliers and outliers. Eskin's algorithm computes the data distribution which is composed of the inliers' distribution and the outliers' distribution. A data point is an outlier if it conforms to the outliers' distribution or does not conform to the inliers' distribution. The details of Eskin's algorithm are discussed step-by-step here:

Step 1: At the beginning Eskin's algorithm assumes that every data point is an inlier and uses a machine learning method (Naïve Bayes or Maximum Entropy) to model the probability distribution of the dataset. Initially, the probability distribution for outliers is a prior probability distribution since all the data points are assumed to be inliers.

Step 2: Each data point is revisited to compute the logarithm of the maximum likelihood of the two cases: (1) when the data point is an outlier and (2) when the data point is an inlier.

Step 3: A data point is moved into the set of outliers if the logarithm of the maximum likelihood when the data point is an inlier is lower than the logarithm of the maximum likelihood when the data point is an outlier.

Step 4: If a data point is moved into the outlier set, the data distribution of the outliers and inliers are recomputed from the remaining set of outliers and inliers.

Eskin's algorithm addresses the uncertainty of the data points, but does not address other issues of data streams. It requires multiple passes over the data to identify outliers and, thus, is not suitable for data streams.

Eskin's algorithm does not consider the transiency or the temporal characteristic of data streams. It works with the entire dataset at a time. Eskin's algorithm assumes all data are available at any point of time; thus it cannot be applicable for data streams. This is because no store-and-process algorithm is applicable for data streams. Eskin's algorithm does not address concept drift and its implementation is not incremental.

Moreover, Eskin's algorithm requires the data distributions of the outliers and the inliers to be recomputed each time the set of outliers and the set of inliers change. It requires many passes over the dataset. The complexity increases with the size of the dataset; therefore, Eskin's algorithm is not useful for a very large dataset.

5.5. Advantages

Statistical techniques often require very small time complexity for detecting outliers. Once a probability distribution is established, these techniques compare a data point with the distribution; hence they can detect outliers very fast. If the data points follow a fixed distribution, these techniques can successfully identify outliers with respect to the obtained data distribution.

5.6. Disadvantages

Data streams are highly dynamic in nature and their distribution changes over time [2]. No fixed data distribution is good enough for an entire data stream; hence summarizing dynamic data streams with static data distributions produces questionable results.

Data points in a data stream have temporal correlations with each other. Statistical techniques ignore such correlations when generating the data distribution. Statistical techniques do not consider all the characteristics of data streams like concept drift, transiency, and temporal relationship; moreover they are not online and incremental; therefore they are not applicable for data stream applications.

6. Cluster-Based Outlier Detection Techniques

6.1. Overview of the Techniques

Many clustering techniques based on machine leaning produce outliers as a byproduct of the clustering techniques [9]. A clustering approach establishes a way of measuring distance (popular distances are the Euclidean distance, Manhattan distance, Mahalanobis distance and Edit distance) between two data points. The data points are grouped into some clusters based on the distances among them. A user-defined threshold is used for cluster selection for each data point. A data point is identified as an outlier if it does not fit any of the obtained clusters. A cluster-based outlier detection technique often uses supervised or semi-supervised approaches for cluster formation [56] and dynamically measures the compactness/goodness of the obtained clusters.

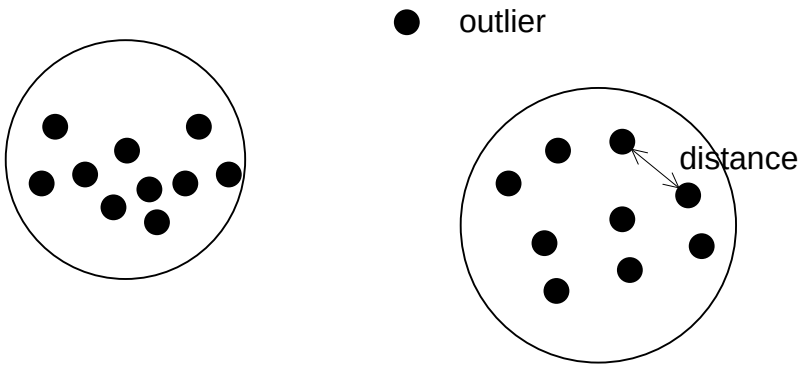

Figure 6. A typical clustering-based outlier detection technique

Figure 6 demonstrates the overall idea of a clustering based outlier detection approach. In this Figure 6 the isolated data point is called an outlier and the clusters are formed based on the distances among the data points. Some techniques cluster the subsequences of data points instead of clustering individual data [72, 73]; these techniques are particularly popular for time series data.

6.2. AnyOut: Anytime Outlier Detection on Streaming Data

AnyOut is a clustering based outlier detection scheme [26]. Unlike other clustering based algorithms where outliers are detected as a byproduct of the underlying clustering algorithm, AnyOut is specifically designed for outlier detection. Moreover, AnyOut can handle dynamic arrival rate effectively. Hence, it only processes a data point until the next data point arrived. AnyOut consists of the following steps:

Step 1: AnyOut creates a hierarchy of clusters from data points of a data stream. AnyOut creates a tree of clusters where the root node represents the entire cluster storing all data points. Each node other than the root node represents a cluster with finer granularity compared to its parent node. Each node including the root node contains the distribution of data points of corresponding cluster.

Step 2: As each data point arrives, AnyOut compares the data point with the root node first and detects the outlier-ness of the data point based on the data distribution in the root node. AnyOut computes the outlier-ness of a data point based on the distance of the data point to the mean of the data points in the cluster. Assent et al. [26] offered an alternative outlier-ness based on the assumption that the data points in a cluster has a multi-dimensional Gaussian distribution. AnyOut keeps track of the distribution of the data points in a cluster and the outlier-ness of a data point is measured in terms of its density.

Step 3: If time permits, AnyOut compares the data point with the appropriate child nodes (defined shortly) of the root node and identifies the outlier-ness of the data point with respect to the child node. Each child node represents a cluster; the cluster which is

closest to the data point is chosen as the appropriate child node. This process continues until the next data point arrives. At any point in time, AnyOut outputs the outliner-ness of a data point based on the most specific cluster with which it is already compared.

Step 4: Based on the application, AnyOut can choose an appropriate cut-off limit for binary decision; if an application does not require binary decision (whether a data point is outlier or inlier) AnyOut is capable of outputting the outlier-ness of every data point.

Although AnyOut addresses dynamic arrival rate, it fails to adjust the transiency of the data points in the data streaming environment. Data points never expired from their hierarchical clusters. Hence an outlier may be detected as an inlier based on the history data points that are irrelevant to the current data point based on the temporal context. AnyOut does not assume any uncertainty in the data points; hence it cannot handle the uncertainty of the data points.

Since clusters never expire, AnyOut is not adaptive to dynamic distribution and, thus, cannot handle concept drift. Moreover, AnyOut computes the outlier-ness of a data point based on the distance of the data point from the mean of the data points in a cluster. If the number of dimensions grows, distance metric is useless as an outlier metric. Similarly, in order to construct an effective data distribution function for high dimensional data points, any algorithm would require an enormous number of data points, which is practically impossible in many applications. Thus the effective-ness of AnyOut's outlier metric is questionable.

6.3. ADMIT: Anomaly-Based Data Mining for Intrusions

ADMIT is a cluster based anomaly detection in a sequence of data [56]. ADMIT has two major phases: the training phase and testing phase. In the training phase, ADMIT learns from the history data, and in the testing phase, ADMIT detects the anomalous sequences.

Step 1: ADMIT assumes that a training dataset is available to train the regular behavior. The entire sequence of data is divided into small subsequences and a dynamic clustering algorithm is used to cluster the small subsequences.

Step 2: The dynamic clustering algorithm is a modified version of the k-mean clustering algorithm. The modified dynamic clustering algorithm does not require any predefined value for k, nor does it choose the initial clusters randomly. It starts with the first subsequence as a cluster and puts all the other subsequences into the cluster if the distance between the subsequence and the cluster center is lower than the user-defined threshold. At every pass it starts with a new subsequence and puts close subsequences into the newly formed cluster. Step 2 completes when each subsequence belongs to a cluster.

Step 3: This step comprises the cluster pruning. In this step the fitness of the clusters are measured and modified. ADMIT splits a cluster into multiple clusters if the cluster contains more than the user-defined number of subsequences. ADMIT splits the cluster using the dynamic clustering algorithm discussed in Step 2, but in this case, the user-defined threshold is increased by 1. ADMIT merges two clusters if the distance between the clusters' centers is lower than the user-defined threshold.

Step 4: Once the training phase is complete, ADMIT is ready for online testing. For each subsequence, ADMIT finds the nearest cluster. If the distance between the subsequence and the nearest cluster's center is greater than the user-defined threshold, the subsequence is identified as a possible outlier.

Step 5: If the last n subsequences are possible outliers, then ADMIT computes the average/weighted/decayed-weighted minimum distances of the last n subsequences and assigns a rating for each subsequence. If the rating is greater than the user-defined threshold, the subsequences are identified as an anomalous behavior. The subsequences are further clustered to determine the type of anomalous behavior.

ADMIT is an online algorithm for detecting anomalous behavior but its training phase is not online. The training phase requires a dataset without outliers which may not be possible in every situation. As each subsequence comes in, ADMIT checks the subsequence and classifies it as either an outlier or inlier, thus it preserves the transiency of the data points. Moreover, ADMIT processes a subsequence in lieu of individual data points; so it implicitly assumes the temporal relationships among the successive data points.

ADMIT clusters the history subsequences; it assumes that the future behaviors of the subsequences are the same as the history subsequences; but this is not true for data streams. Concept drift is a well-known phenomenon in data stream applications; assuming a fixed data distribution is not suitable for data stream applications. Finally, ADMIT does not address the uncertainty of data. Essentially, ADMIT is not appropriate for data stream applications.

6.4. A Machine Leaning Approach to Anomaly Detection (CLAD)

CLAD is a clustering based outlier detection algorithm that finds local and global outliers [57]. CLAD is originally designed for outlier detection unlike other clustering based techniques that produce outliers as a byproduct. CLAD is based on a fixed cluster width which is selected automatically by itself. The technique is discussed step-by-step here.

Step 1: CLAD uses the fixed cluster width for data point selection of a cluster. CLAD uses a non-deterministic algorithm for cluster width selection. CLAD randomly selects 1% of the data points. It computes the pair-wise distances between two data points among the chosen data points and finds the smallest 1% of the pair-wise distances. The average distance of the minimum 1% is chosen as a cluster width.

Step 2: CLAD creates the clusters on-the-fly. A data point is placed into a cluster if it is within the cluster width of any previously created clusters; otherwise CLAD creates a new cluster for the data point and the data point is selected as a centroid of the newly formed cluster. CLAD uses the Euclidian distance between two data points.

Step 3: Once all the clusters are formed, CLAD computes two metrics for each cluster: the number of data points in a cluster and the average distance of the cluster from all other clusters. CLAD also computes the average inter-cluster distances among all the clusters. A cluster is distant if the average distance of the cluster from all other clusters (the second metric) is greater than the sum of the average inter-cluster distances and the standard deviation of the inter cluster distances.

Step 4: A cluster is sparse if it has fewer data points than the lower limit where the lower limit is computed by subtracting the median absolute deviation of the number of data points in a cluster from the average number of data points in a cluster. A data cluster is dense if it has more data points than the upper limit where the upper limit is computed by adding the average number of data points in a cluster to the mean absolute deviation of the number of data points in a cluster. The entire cluster is identified as an outlier if it is distant and sparse or dense.

CLAD is not designed for data streams. It requires more than one pass over the entire dataset for clustering and outlier detection. As a consequence, it does not address the transiency and the temporal characteristic of data points. It does not tackle the other issues like uncertainty and concept drift for data streams and its implementation is neither online nor incremental. Therefore, CLAD is not applicable for data streams.

6.5. Advantages

Clustering-based outlier detection approaches are appealing because of their power of sorting similar data points into a group. A data point which is grouped with many other data points is less likely to be an outlier; hence an outlier identified by a clustering-based technique is usually far from other data points; therefore it produces fewer false alarms.

An important characteristic of clusters is that cluster formation is incremental [9, 56], hence clustering-based outlier detection techniques are also incremental. In some cases clusters are not only incremental but also dynamically adjustable, and so the techniques

dynamically adjust the clusters based on data and detect outliers based on the obtained clusters.

6.6. Disadvantages

Outliers are the byproduct of the clustering techniques [74, 75]; hence the techniques are not optimized for outlier detection. This is an effective and popular argument against the clustering-based outlier detection techniques [9, 32]. The cluster formation step requires a particular choice of distance function and a threshold which groups the data points into a cluster. The choices of the distance function and threshold affect the outlier detection and require explicit knowledge about the application domains.

An outlier can be an inlier if it is accompanied by a sufficient number of data points; hence clustering-based outlier detection techniques cannot identify the outliers which are close to the regular data points. Besides this, these techniques cannot identify the outliers that form a different cluster with a sufficient number of data points in it. Moreover, the time series clustering based outlier detection techniques that are optimized for outlier detection [72, 76] fail to address concept drift. They require a training phase, build a model and compare each temporal sequence with the captured model. The techniques assume that the trend in the dataset is fixed, which is not true for data streams. Furthermore, looking from the data streams' perspective, the clustering based outlier detection algorithms do not deal with the uncertainty and the temporal characteristics of data stream applications.

The biggest flaw of clustering based algorithms is similarity measurement. The popular distance functions, such as Euclidian distance and Manhattan distance, fail to portray

the similarity among the data points. This is because as the number of dimensions grows, all data points become equidistance. Thus the distance between an outlier and an inlier is no different than that between two inliers. Thus, clustering-based techniques fail to point out the outliers.

7. Feature Comparison of Existing Outlier Detection Techniques

We have presented the state-of-the-art outlier detection techniques in the previous sections. None of the discussed techniques deal with all the characteristics of data streams: online incremental processing, uncertainty, temporal relations among the data points, concept drift, transiency, multi-dimensionality, asynchronous arrival rate and heterogeneous schema of data points. To fill this gap, in this dissertation, two innovative outlier detection algorithms, Orion and Wadjet, which take all the data streams' characteristics into consideration are presented. Orion is designed for applications where data are single streams which are not related with each other. In order to detect outlier efficiently for multi-dimensional data points, Orion looks for a projected dimension that reveals the outlier nature of data points with the help of an evolutionary algorithm, and identifies a data point as an outlier if it resides in a low density region in that dimension. Wadjet is designed for applications where data are heterogeneous and asynchronous streams which may or may not be related with each other. It has two phases: in the first phase, it processes each data point independently like Orion and detects the outliers based the temporal correlation of the data points; and in the second phase, it captures and continuously evaluates the cross-correlations, if any, among the data points from multiple streams, and identifies a data point as an

outlier if its value does not conform to the captured cross-correlation. A data point is identified as an outlier in Wadjet if it is identified as an outlier in any of its two phases.

In Table 1 we present a feature comparison study of the techniques discussed in this chapter in contrast to the data streams' issues discussed in Chapter I, Section 5. The explanation for each cell is in the corresponding algorithm discussion section. 'Yes' in a cell means the algorithm listed at the left addresses the issue listed at the top, and 'No' means the algorithm ignores the issue.

Table 1. Feature comparison of the outlier detection techniques

	Transiency	Temporal relationship	Online	Incremental	Arrival rate	Uncertainty	Concept drifting	Multi-dimensionality	Cross-correlation	Asynchronous	Dynamic relationship	Heterogeneous Schema
DB-Outlier **[32]**	Yes	No	Yes	Yes	Fixed	No	Yes	No	No	No	No	No
EAMO **[51]**	No	No	No	No	No	No	No	No	No	No	No	No
DBOD-DS **[47]**	Yes	Yes	Yes	Yes	Fixed	Yes	Yes	No	No	No	No	No
LOCI **[59]**	No	No	No	No	No	No	No	No	No	No	No	No
LOF **[24]**	No	No	No	No	No	No	No	No	No	No	No	No
A-ODDS **[60]**	Yes	Yes	Yes	Yes	Fixed	Yes	Yes	No	No	No	No	No
STORM **[54]**	Yes	Yes	No	Yes	Fixed	No	No	No	No	No	No	No
ODTS **[13]**	Yes	Yes	Yes	Yes	Fixed	No	No	No	No	No	No	No
ODSD **[36]**	Yes	Yes	Yes	Yes	Fixed	Yes	No	No	No	No	No	No
Stream LOCI **[62]**	Yes	Yes	Yes	Yes	Fixed	No	No	No	No	No	No	No
KFOD **[63]**	Yes	Yes	Yes	Yes	Fixed	No	No	No	No	No	No	No
ART **[64]**	Yes	Yes	Yes	Yes	Fixed	No	No	No	No	No	No	No
AMSD **[65]**	Yes	Yes	Yes	Yes	Fixed	No	Yes	No	No	No	No	No
IDMD **[70]**	No	No	No	No	Fixed	No	No	No	No	No	No	No
Horn's Algorithm **[71]**	No	No	No	No	Fixed	Yes	No	No	No	No	No	No
Eskin's Algorithm **[27]**	No	No	No	No	Fixed	Yes	No	No	No	No	No	No
AnyOut **[26]**	No	No	Yes	Yes	Dynamic	No	No	No	No	No	No	No
ADMIT **[56]**	Yes	Yes	No	No	No	No	No	No	No	No	No	No
CLAD **[57]**	No	No	No	No	No	No	No	No	No	No	No	No
Orion	Yes	Yes	Yes	Yes	Fixed	Yes	Yes	Yes	No	No	No	No
Wadjet	Yes	Yes	Yes	Yes	Fixed	Yes	Yes	Yes	Yes	Yes	Yes	Yes

CHAPTER III
THE PROPOSED TECHNIQUES: ORION AND WADJET

This chapter presents both of our outlier detection techniques, Orion and Wadjet, for single and multiple data streams.

1. Outlier Detection for Single Streams

In the single data streams model, data points from one single stream are independent of those from another single stream. They have no cross-correlation and therefore comparing them to each other is not meaningful. Hence in case of single data streams we identify outliers in a stream based on the data points from that stream only. In the single data streams model, a Data Stream Ψ is an infinite set of data points, $\Psi = \{D_t^i | 0 \leq t\}$ where a data point D_t^i is a set of attribute values with an explicit or implicit timestamp t from data stream i. Formally a data point is $D_t^i = (\boldsymbol{V}, t)$ where $\boldsymbol{V}$ is a p-tuple, each value of which corresponds to one attribute, and t is the timestamp for the data point. A data point D_t^i is an outlier if it is significantly different from other data points $\{D_p^i | p \neq t\}$. Figure 7 shows a set of data streams where a sequence of data points is produced from an independent source (represented by a double circle) and the outlier detection component receives one data point (D_t^i) at a time and marks it as an outlier (O_t^i) if it is deviated from the other data points, or an inlier (I_t^i) otherwise, and that outlier/inlier is continued for further processing.

The general idea of our outlier detection method is as follows. We summarize the set of data points received so far. The summary will capture the overall trend of the data points from the same stream. Once a new data point arrives, it is compared to the summary and the value(s) of our outlier metric(s) are computed. If the value(s) for the outlier metric(s) does not fit the notion of normality, then the newly arrived data point is identified as an outlier, otherwise an inlier. The summary of the obtained data points is updated incrementally for detecting outliers for future data points. We call our outlier detection technique for independent streams, Orion.

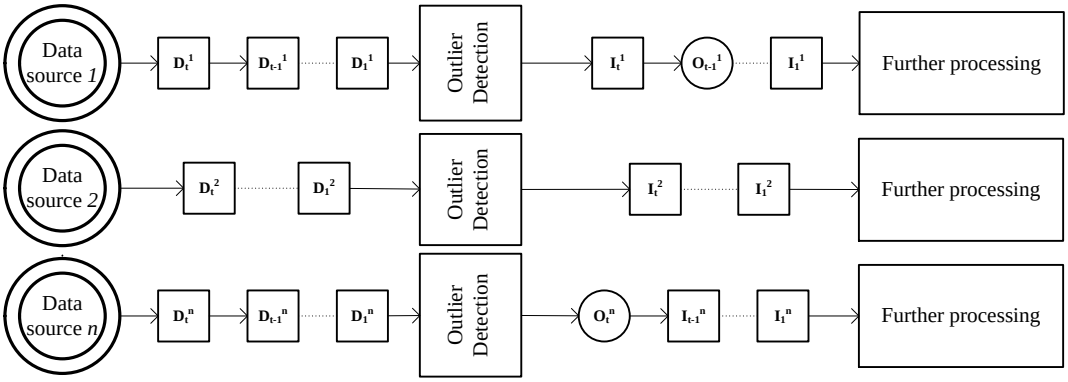

Figure 7. Single data stream model

1.1. Motivation of Orion

Before going into the workflow of Orion, we would like to discuss the motivation of Orion in this section. We start with the motivation of multi-dimensional outlier detection which is the primary motivation of Orion and move forward with other motivations for Orion. We have already developed outlier detection algorithms for data streams, DBOD-DS [47] and A-ODDS [60], but those algorithms are very specific for single dimensional streams. If we apply a single dimensional outlier detection algorithm for each individual data dimension separately in multidimensional data streams, we can identify outliers that have considerably different values in that dimension compared to

those of other data points, but we will not be able to capture the outliers that have considerably different combinations of attribute (dimensional) values.

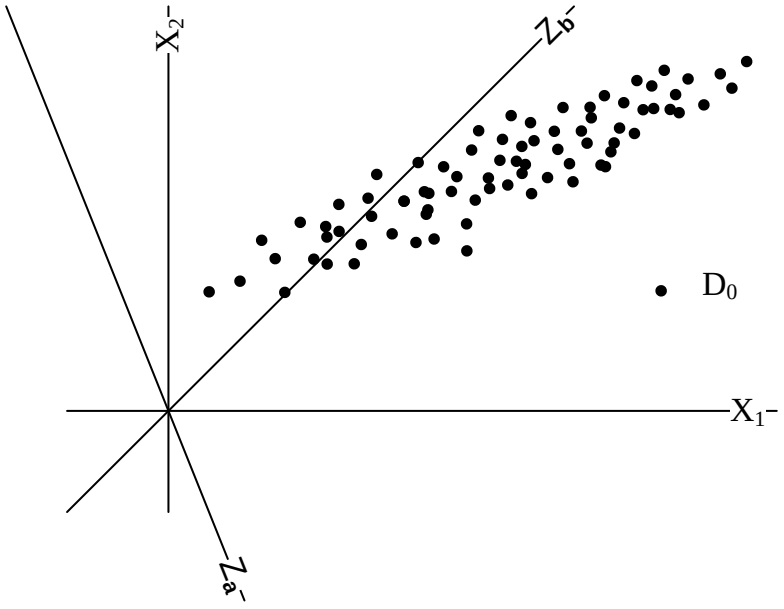

Figure 8. Multi-dimensional outliers

Figure 8 illustrates the idea of multi-dimensional outliers. In this figure, each dot represents a two dimensional data point. Consider the data point D_0 which looks as it is a distant one from the rest of the data points. However, if we consider two dimensions X_1 and X_2 separately, the value of D_0 along X_1 is not much different from the values of other data points along X_1, and the same is true along X_2 as well. Therefore, considering one single dimension alone like X_1 or X_2 may not produce any meaningful result. Consequently, outlier detection algorithms like DBOD-DS and A-ODDS would not be capable of detecting outliers like D_0. Therefore those two techniques cannot detect outliers in multi-dimensional streams - a gap that our proposed algorithm, Orion, aims to fill. Orion detects a data point as an outlier if the data point has a drastically different value or combination of attribute values compared to other data points.

One straightforward solution for multi-dimensional outlier detection would be designing a multi-dimensional data density function. However the density space grows exponentially with the number of dimensions and thus such function is not scalable. Interestingly, we can create an artificial dimension, which we call projected dimension (p-dimension), by linearly combining data dimensions and detect outliers by applying a single dimensional data density function along that p-dimension, computed from the data points projected on that p-dimension. Since the p-dimension considers all data dimensions, it can detect outliers that would have been missed if only individual data dimensions were used. As an example, Figure 8 shows Z_a and Z_b as two p-dimensions each of which is a linear combination of two data dimensions, X_1 and X_2. Now, although D_0 may not look like an outlier if we consider the value of D_0 on X_1 or on X_2 only, it certainly looks like an outlier if we consider the value of D_0 projected on the p-dimension Z_a. This is because Z_a captures the correlation between X_1 and X_2, and any combination of attribute values that violates the correlation of X_1 and X_2 can be observed by looking at the projected dimension Z_a. Theoretically there are infinitely many such p-dimensions, but Orion needs to find only one p-dimension specific to a particular data point D_t along which we can identify whether D_t is considerably different from other data points.

So the first motivation for Orion is to find the p-dimenison specific to D_t that reveals the outlier-ness of D_t. The second motivation of Orion is to avoid using a distance metric to measure the similarity among the data points. If we avoid using a distance metric and use a data point specific p-dimension to determine the outlier-ness of D_t, the p-dimension that reveals the outlier-ness of a data point varies for each data point and

also varies over time. Thus the third motivation is to find the appropriate p-dimension adaptively for a particular D_t that reveals the outlier-ness of D_t. The outlier-ness of D_t is captured by the outlier metric which leads to the last but not the least motivation of avoiding using any specific threshold for the outlier metric to distinguish D_t from inliers. The details of Orion are presented in the next section.

1.2. Overview of Orion

Orion processes each data point from one stream individually. To identify whether a data point D_t is an outlier, Orion goes through three phases: (1) finding an appropriate p-dimension for D_t , (2) computing the outlier metrics for D_t, and (3) determining if D_t is an outlier, based on the outlier metrics. To find an appropriate p-dimension, Orion uses an Evolutionary Algorithm (EA) because EA can effectively optimize any objective function and is adaptive to the change of environment [77]. Every data point D_t has a value along a p-dimension. If this value has much fewer neighbors than the values of other data points, then D_t has either a considerably different value or combination of attribute values compared to other data points and, thus, is very likely an outlier. Therefore the goal of the evolutionary algorithm is to search for the p-dimensions along one of which D_t has the minimum number of neighbors. Orion starts with an initial set of p-dimensions; however this set may not include the p-dimension that would incur the minimum number of neighbors for D_t. Therefore, Orion gradually modifies this set by adding new p-dimensions and removing old p-dimensions. The new ones will most likely incur fewer neighbors for D_t compared to the old ones. This

process is also adaptive to concept drift as the set is also modified if a concept drift occurs.

The appropriateness of a p-dimension is measured by its ability to incur a minimum neighbor count for a data point along that p-dimension. However, the neighbor computation requires storing the entire set of data points which is impossible for data streams. To solve this problem, Orion approximates the neighbor count with the help of a neighbor density function as proposed in Section 1.3.2 (Chapter III). As each data point D_t arrives, Orion picks the p-dimension that has the smallest neighbor density from the existing p-dimensions as it would better reveal the outlier-ness of D_t compared to other p-dimensions.

Orion then computes the outlier metrics for D_t along that p-dimension in order to determine whether it is an outlier or inlier. Two popular metrics are neighbor density (ND) [47, 49] and k-distance [24]. If a data point is considerably different from other data points, it would have much fewer neighbor data points, and thus a smaller ND, compared to other data points. The k-distance is the minimum distance that includes the k number of neighbors [24]. If a data point is considerably different from other data points, it would require a larger distance to include the same number of neighbors. Consequently an outlier would have a smaller ND and a larger k-distance compared to an inlier. Orion uses both ND and k-distance in order to detect outliers. However, ND and k-distance computations require random access to the entire dataset (to compute pair-wise distances between the data points) and do not consider the temporal relationship and uncertainty among the data points. Orion solves all three problems by computing ND and k-distance using our proposed data density function that explicitly

addresses the temporal relationship and uncertainty in data streams without random access to the entire dataset. The modified definitions of ND and k-distance for data streams are discussed in Definition 4 and Definition 7.

After computing the ND and k-distance of D_t, Orion uses co-clustering [78] to identify whether the ND is considerably low and the k-distance is considerably high, compared to other data points. Co-clustering clusters a set of data points based on multiple attributes where each attribute is clustered into a specific number of groups based on the values of that attribute only. Orion clusters the data points based on the NDs and k-distances into three groups: small, average, and large. The data points that belong to the clusters with a small ND and large k-distance are identified as outliers. Table 2 presents the list of symbols used in this paper and the rest of this section discusses Orion in details.

Table 2. List of symbols

Symbol	Interpretation
A_{in}, A_{out}	A set of p-dimensions for data points that are most likely to be inliers (A_{in}) and outliers (A_{out})
A, A_T	a set of p-dimensions, ($A = A_{in} \cup A_{out}$), a working set of p-dimension at T, ($A_T = A_{in}$ or $A_T = A_{out}$)
$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a}', \boldsymbol{b}'$ $\boldsymbol{c}, \boldsymbol{c}'$	m-dimensional vectors, the transpose of the vector $\boldsymbol{a}$ is written as $\boldsymbol{a}'$, the transpose of $\boldsymbol{b}$ **and** $\boldsymbol{c}$ is $\boldsymbol{b}'$ and c'
$a[i]$	i-th component of vector $\boldsymbol{a}$. The same for $\boldsymbol{b}$ and $\boldsymbol{c}$
D_t	a data point at time t
$D_t.\boldsymbol{v}$	m dimensional data value vector for data point D_t
$D_t.v[\boldsymbol{a}]$	the value of D_t along p-dimension $Z_{\boldsymbol{a}}$, $D_t.v[\boldsymbol{a}] = \boldsymbol{a}' D_t.\boldsymbol{v}$
$f_{Z_{\boldsymbol{a}},T}(z_{\boldsymbol{a}})$	data density function along p-dimension $Z_{\boldsymbol{a}}$ up to time T
k	user defined neighbor density for k-distance
m	the number of dimensions
r	user defined neighbor distance
$r_{Z_{\boldsymbol{a}}}$	scaled neighbor distance for dimension $Z_{\boldsymbol{a}}$, $r_{Z_{\boldsymbol{a}}} = 2\delta_{Z_{\boldsymbol{a}}} r$
T	Current timestamp; timestamp starts at 0, increased by 1
$Z_{\boldsymbol{a}}$	a p-dimension along vector $\boldsymbol{a}$

$z_{\boldsymbol{a}}$	a random variable along p-dimension $Z_{\boldsymbol{a}}$
$\boldsymbol{\mu}_T$	the mean vector up to time T. $\boldsymbol{\mu}_T = \frac{1}{T+1}\sum_{t=0}^{T} D_t.\boldsymbol{v}$
$\boldsymbol{\Sigma}_T$	the covariance matrix of attribute values up to time T. $\boldsymbol{\Sigma}_T = \frac{1}{T}\sum_{t=0}^{T}(D_t.\boldsymbol{v} - \boldsymbol{\mu}_t)(D_t.\boldsymbol{v} - \boldsymbol{\mu}_t)'$
$\sigma_{\boldsymbol{a}}$	standard deviation along p-dimension $Z_{\boldsymbol{a}}$
$2\delta_{Z_{\boldsymbol{a}}}$	bin width along p-dimension $Z_{\boldsymbol{a}}$

1.3. Evolutionary Algorithm

Every evolutionary algorithm has two major parts in it: (1) the objective of the evolutionary algorithm and (2) the model of evolution [79].

1.3.1. Objective of Evolutionary Algorithm

The objective of Orion's evolutionary algorithm is to find a set of p-dimensions, one of which would incur the minimum Neighbor Density (ND) for a data point. The ND of a data point in a data stream is analogous to the neighbor count of a data point in a finite set of data points [49 , 54]. For a finite set of data points, the neighbor count represents the number of data points occurring within a given distance from the data point D_t. ND represents the percentage of data points occurring within a given distance from D_t. ND is our approximation of neighbor count for a data point in a data stream. We calculate the ND of D_t from the data density function (defined in Section 1.3.2).

Definition 4. The neighbor density of a data point D_t along a p-dimension Z_a is defined as the percentage of neighbors of D_t within the scaled neighbor distance r_{Z_a}. Formally,

$$nDen_{Z_{\boldsymbol{a}}}(D_t, r_{Z_{\boldsymbol{a}}}) = \int_{D_t.v[\boldsymbol{a}]-r_{Z_{\boldsymbol{a}}}}^{D_t.v[\boldsymbol{a}]+r_{Z_{\boldsymbol{a}}}} f_{Z_{\boldsymbol{a}},T}(z_{\boldsymbol{a}})dz_{\boldsymbol{a}} \quad (1)$$

where $f_{Z_a,T}(z_a)$ is the data density function discussed in Section 1.3.2.

The small ND of a data point along any p-dimension is a good indicator of the outlier-ness for that data point [77] because it indicates that the data point is isolated from other data points in that p-dimension. In order to estimate whether a data point is an outlier, it is sufficient to find a p-dimension Z_a along which it has a low ND [77] compared to other data points. The p-dimension with the minimum neighbor density reveals the most outlier nature of a data point.

Definition 5. The Minimum Density Dimension (MDD) for a data point D_t is the p-dimension along which D_t has the minimum neighbor density compared to all other p-dimensions. Formally,

$$MDD(D_t) = Z_a | \, nDen_{Z_a}(D_t, r_{Z_a}) \leq nDen_{Z_b}(D_t, r_{Z_b}) \; \forall \, b \tag{2}$$

Theoretically each data point has an MDD, but practically many data points may share the same MDD (data points with the same value have the same MDD). Computing such p-dimension for every data point is a difficult optimization problem. The objective of the evolutionary algorithm is to find a set of p-dimensions one of which is MDD for D_t. We choose evolutionary algorithm over deterministic algorithm for two reasons: (1) no close form exists for the optimum p-dimension and the co-efficients of each data dimension vary for each optimal p-dimension, which makes greedy or simulated annealing algorithm very challenging for this optimization, and (2) evolutionary algorithm is very adaptive to the change of data distribution [79]. Moreover, given a proper model of evaluation, an evolutionary algorithm can optimize any function without its closed form [79].

Other component analysis techniques like PCA and MCA do not guarantee the resultant p-dimension to have minimum neighbor density compared to other p-dimensions. Figure 9 illustrates the idea with an example where each dot represents a data point and D_0 to D_5 are the outliers. Each of these outliers is far from other data points in a two dimensional space.

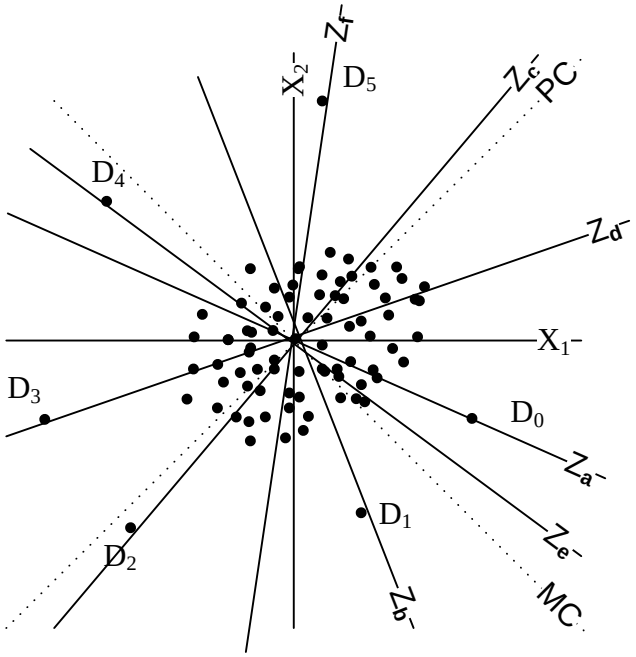

Figure 9. PCA and MCA with other p-dimensions

The principal component (PC) and the minor component (MC) of the data points are shown as dotted lines in Figure 9. Now consider the outlier D_0. The minimum neighbor density of D_0 lies along p-dimension Z_a, but not on the PC or MC. Similarly, we can consider D_1, D_2, D_3, D_4, and D_5; the minimum neighbor density of each of them lies along Z_b, Z_c, Z_d, Z_e, and Z_f. Neither PC nor MC can yield the minimum neighbor density for all of them; this is because the principal component and minor component are fixed for a dataset, where the MDD is different for each of data point. Hence,

finding the appropriate p-dimension with the evolutionary algorithm is more appropriate than using the other component analysis model like PCA or MCA.

1.3.2. Model of Evolution

The model of evaluation of the evolutionary algorithm has three components: (1) the population set, (2) the fitness function, and (3) the population modification.

1.3.2.1. Population Set

Orion starts with an initial set of p-dimensions A called the population set (it can be chosen randomly [79]; we use eigenvectors of covariance matrix as the initial set and the size of the population set is the same as the number of data dimensions). Each p-dimension in A is called a population. A is divided into two sets A_{in} and A_{out}. Two sets are necessary for applications like outlier detection because inliers occur more often than outliers [12], and the p-dimensions that have low NDs for inliers may not have low NDs for outliers. Since the number of inliers is much higher than the number of outliers, the p-dimensions that produce low NDs for inliers will dominate the entire population; thus Orion would fail to find a p-dimension that incurs low ND for an outlier. To avoid such situation Orion randomly divides the initial set of p-dimensions A into two separate sets, A_{in} and A_{out}. The p-dimensions in A_{in} produce the minimum neighbor density for the data points that are surrounded by many other data points and thus are likely inliers, and the p-dimensions in A_{out} produce the minimum neighbor density for the data points that are not surrounded by other data points and thus are likely outliers. In order to find the data points that are surrounded by many other data points, we propose the concept of absolute normalized deviation.

Definition 6. The Absolute normalized Deviation (AD) of a data point D_t along a p-dimension Z_a is defined by the absolute distance between the value of D_t and the mean in scale of standard deviation along Z_a. Formally,

$$AD_{Z_a}(D_t) = \frac{|\boldsymbol{a}' D_t.\boldsymbol{v} - \boldsymbol{a}'\boldsymbol{\mu}_T|}{\sigma_{\boldsymbol{a}}} = \frac{|\boldsymbol{a}'(D_t.\boldsymbol{v} - \boldsymbol{\mu}_T)|}{\sigma_{\boldsymbol{a}}} \quad (3)$$

Lemma 1. For any arbitrary p-dimension $Z_{\boldsymbol{a}}$ and data point D_t, the maximum value of $AD_{Z_{\boldsymbol{a}}}(D_t)$ is $\max\{AD_{Z_{\boldsymbol{a}}}(D_t)\} = \sqrt{(D_t.\boldsymbol{v} - \boldsymbol{\mu}_T)'\boldsymbol{\Sigma}_T^{-1}(D_t.\boldsymbol{v} - \boldsymbol{\mu}_T)}$ where $\boldsymbol{a} = c\boldsymbol{\Sigma}_T^{-1}(D_t.\boldsymbol{v} - \boldsymbol{\mu}_T)$ and c is a constant.

Proof. $AD_{Z_{\boldsymbol{a}}}(D_t)$ would be maximized if $AD_{Z_{\boldsymbol{a}}}(D_t)^2$ is maximized since $AD_{Z_{\boldsymbol{a}}}(D_t) \geq 0$. It is computed as:

$$AD_{Z_{\boldsymbol{a}}}(D_t)^2 = \left(\frac{|\boldsymbol{a}'(D_t.\boldsymbol{v} - \boldsymbol{\mu}_T)|}{\sigma_{\boldsymbol{a}}}\right)^2$$

$$= \frac{\left(\boldsymbol{a}'(D_t.\boldsymbol{v} - \boldsymbol{\mu}_T)\right)^2}{\boldsymbol{a}'\boldsymbol{\Sigma}_T\,\boldsymbol{a}} = \frac{(\boldsymbol{a}'\boldsymbol{d})^2}{\boldsymbol{a}'\boldsymbol{\Sigma}_T\boldsymbol{a}} \text{ where } \boldsymbol{d} = D_t.\boldsymbol{v} - \boldsymbol{\mu}_T$$

$= \boldsymbol{d}'\boldsymbol{\Sigma}_T^{-1}\boldsymbol{d}$ according to Extended Cauchy-Schwarz inequality $(\boldsymbol{a}'\boldsymbol{d})^2 \leq (\boldsymbol{a}'\boldsymbol{\Sigma}_T\boldsymbol{a})(\boldsymbol{d}'\boldsymbol{\Sigma}_T^{-1}\boldsymbol{d})$ and $AD_{Z_{\boldsymbol{a}}}(D_t)^2$ is maximized when equality is achieved [19]

$$AD_{Z_{\boldsymbol{a}}}(D_t) = \sqrt{(D_t.\boldsymbol{v} - \boldsymbol{\mu}_T)'\boldsymbol{\Sigma}_T^{-1}(D_t.\boldsymbol{v} - \boldsymbol{\mu}_T)}$$

and the maximum value of $AD_{Z_{\boldsymbol{a}}}(D_t)$ is attained if $\boldsymbol{a} = c\boldsymbol{\Sigma}_T^{-1}(D_t.\boldsymbol{v} - \boldsymbol{\mu}_T)$.

Heuristically, if the value of a data point D_t is far from the mean value along a p-dimension, D_t will most likely have a smaller neighbor density along that p-dimension compared to other p-dimensions. The p-dimensions in A_{in} and A_{out} are modified based on the data points the maximum ADs of which are lower than the average maximum AD and greater than the average maximum AD. Once a data point arrives, the evolutionary algorithm chooses either A_{in} or A_{out} based on the value of the maximum AD of D_t and loads it into the working set A_T.

1.3.2.2. The Fitness Function

The fitness of an evolutionary algorithm decides which set of populations will be chosen or discarded for future steps. We calculate the neighbor density of D_t for each p-dimension and sort the p-dimensions into descending order; and the rank of a p-dimension is its fitness value for that particular D_t. We do so because the minimum value of ND of D_t along any possible p-dimension is unknown; hence it is difficult to measure the goodness of an ND along any existing p-dimension with respect to the minimum ND. According to Lemma 2, the smaller the angle between a p-dimension $Z_{\boldsymbol{a}}$ and the optimal p-dimension, the smaller ND $Z_{\boldsymbol{a}}$ produces compared to other p-dimensions and vice versa. Therefore the $Z_{\boldsymbol{a}}$ that induces the smallest ND must have the smallest angle with the optimal p-dimension, hence, has the highest fitness. So for the current working set A_T, the p-dimension which has the smallest ND has fitness $|A_T|$, the p-dimension which has the largest ND has fitness 1, and the remaining p-dimensions have fitness between $|A_T|$ and 1. The overall fitness of a p-dimension is the average fitness value of that p-dimension for all history data points. The fitness values

of all p-dimensions of A_T are updated at every time period and a new dimension is created and an old one is deleted (called the evolutionary step) randomly.

Lemma 2. The neighbor density along a p-dimension increases with the increase of the angle between the p-dimension and optimal p-dimension.

Proof. Let, $MDD(D_T) = Z_{opt}$ meaning $nDen_{Z_{opt}}\left(D_T, r_{Z_{opt}}\right) \leq nDen_{Z_b}(D_T, r_{Z_b})\ \forall\ Z_b$. Let Z_a be a p-dimension and $Z_a \neq Z_{opt}$.

$$nDen_{Z_a}(D_T, r_{Z_a}) = \int_{D_T.v[a]-r_{Z_a}}^{D_T.v[a]+r_{Z_a}} f_{Z_a,T}(z_a)dz_a$$

$$= \int_{D_T.v[a]-r_{Z_a}}^{D_T.v[a]+r_{Z_a}} \frac{\sum_{t=\tau}^{T} \lambda^{T-t-\tau} k_{h_{Z_a,t}}(D_t.v[a]-z_a)}{\sum_{t=\tau}^{T} \lambda^{T-t-\tau}} dz_a$$ [Expanding DDF]

$$= \int_{D_T.v[a]-r_{Z_a}}^{D_T.v[a]+r_{Z_a}} \frac{\sum_{t=\tau}^{T} k_{h_{Z_a,t}}(D_t.v[a]-z_a)}{T-\tau+1} dz_a$$ [for the sake of simplicity let us assume that all history data points have equal weight, later we will argue why the result is extendable to variable weights]

$$= \frac{1}{T-\tau+1} \sum_{t=\tau}^{T} \int_{D_T.v[a]-r_{Z_a}}^{D_T.v[a]+r_{Z_a}} k_{h_{Z_a,t}}(D_t.v[a] - z_a)dz_a$$

$$= \frac{1}{T-\tau+1} \sum_{t=\tau}^{T} \int_{D_T.v[a]-r_{Z_a}}^{D_T.v[a]+r_{Z_a}} \frac{3}{4}\left(1 - \left(\frac{D_t.v[a]-z_a}{h_{Z_a,t}}\right)^2\right) dz_a$$ [using Epanechnikov Kernel]

$$= \frac{1}{T-\tau+1} \sum_{t=\tau}^{T} \frac{3}{4} h_{Z_a,t} \int_{\frac{a'(D_T.v-D_t.v)+r_{Z_a}}{h_{Z_a,t}}}^{\frac{a'(D_T.v-D_t.v)-r_{Z_a}}{h_{Z_a,t}}} (z_a'^2 - 1)\, dz_a' \quad [z_a' = \frac{a' D_t.v - z_a}{h_{Z_a,t}}]$$

$$= \frac{1}{T-\tau+1}\sum_{t=\tau}^{T}\frac{3}{4}h_{Z_{\boldsymbol{a}},t}\left[\frac{{z'_a}^3}{3}-z'_a\right]_{\frac{\boldsymbol{a'd_T}-r_{Z_{\boldsymbol{a}}}}{h_{Z_{\boldsymbol{a}},t}}}^{\frac{\boldsymbol{a'd_T}+r_{Z_{\boldsymbol{a}}}}{h_{Z_{\boldsymbol{a}},t}}} \text{ [where } d_T = D_T.v - D_t.v]$$

$$= \frac{1}{T-\tau+1}\sum_{t=\tau}^{T}\frac{3}{4}h_{Z_{\boldsymbol{a}},t}\left[\frac{{z'_a}^3}{3}-z'_a\right]_{\alpha_{Z_a.t}}^{\beta_{Z_a.t}} \quad \text{[where } \beta_{Z_a.t} = \frac{\boldsymbol{a'd_T}-r_{Z_{\boldsymbol{a}}}}{h_{Z_{\boldsymbol{a}},t}} \quad \&$$

$$\alpha_{Z_a.t} = \frac{\boldsymbol{a'd_T}+r_{Z_{\boldsymbol{a}}}}{h_{Z_{\boldsymbol{a}},t}}]$$

$$= \frac{r_{Z_{\boldsymbol{a}}}}{2(T-\tau+1)}\sum_{t=\tau}^{T}[3 - \alpha_{Z_a.t}^2 - \beta_{Z_a.t}^2 - \alpha_{Z_a.t}\beta_{Z_a.t}]$$

The neighbor density would be minimum if $[3 - \alpha_{Z_a.t}^2 - \beta_{Z_a.t}^2 - \alpha_{Z_a.t}\beta_{Z_a.t}]$ is minimum for each data point; and so $[3 - \alpha_{Z_a.t}^2 - \beta_{Z_a.t}^2 - \alpha_{Z_a.t}\beta_{Z_a.t}]$ would be minimum if $\alpha_{Z_a.t}$ and $\beta_{Z_a.t}$ are maximum. According to the definition of Epanechnikov Kernel the maximum value of $\alpha_{Z_a.t}$ and $\beta_{Z_a.t}$ is 1; in that case the minimum neighbor density would be 0.

$\beta_{Z_a.t} = \frac{\boldsymbol{a'd_T}-r_{Z_{\boldsymbol{a}}}}{h_{Z_{\boldsymbol{a}},t}} = \frac{(|a||d_T|cos\theta_a - r_{Z_{\boldsymbol{a}}})}{h_{Z_{\boldsymbol{a}},t}}$ [θ_a is the angle of $\boldsymbol{a}$ & $\boldsymbol{d_T}$**]**

$= \frac{(|d_T|cos\theta_a - r)}{h}$ [since $\boldsymbol{a}$ is a unit vector and neighbor distance and bandwidth are equal for all p-dimensions**]**

$\beta_{Z_a.t}$ would be maximum if $cos\theta_a$ is maximum. Thus, the optimal p-dimension has maximum $cos\theta_a$, thus Z_{opt} is parallel to $\boldsymbol{d_T}$; and as the angle between $\boldsymbol{d_T}$ and $\boldsymbol{a}$ increases $\beta_{Z_a.t}$ decreases. So, the $\beta_{Z_a.t}$ decreases as the angle between Z_{opt} and Z_a increases. The same is also true for $\alpha_{Z_a.t}$. So, the neighbor density increases as the angle

between Z_{opt} and Z_a increases. Since the total neighbor density is the sum of the neighbor density induced by each data point, the weight does not influence the neighbor density induced by a data point. Moreover, the weight of a data points does not play any role for dimension selection since the weight is equal along all dimensions for one data point.

1.3.2.3. The Population Modification

The next components of the evolutionary algorithm are the selection of existing populations with high fitness (p-dimensions) and the design of new populations from them. Orion creates new populations using crossover and cultivation and removes existing populations with low fitness using deletion.

Crossover. Crossover finds two populations (two p-dimensions) called parent populations with high fitness and creates a new population that performs even better than the parent populations. Two populations are selected based on the rank selection scheme [18] which sorts the p-dimensions based on their fitness values in descending order. The first p-dimension has the highest probability of being selected and the last p-dimension has the lowest probability of being selected. Once two p-dimensions are selected, a new p-dimension ($Z_{\boldsymbol{c}}$) is produced from the combination of the two parent p-dimensions. According to Lemma 3, the linear combination of two p-dimensions produces a new p-dimension which produces smaller neighbor density compared to the parents. Since $Z_{\boldsymbol{a}}$ and $Z_{\boldsymbol{b}}$ are two parents p-dimensions where $\boldsymbol{a}$ and $\boldsymbol{b}$ are two unit vectors along them, the new $Z_{\boldsymbol{c}}$ will have a unit vector $\boldsymbol{c}$ which is computed using the following Equation (4):

$$c[i] = \begin{cases} \gamma a[i] + (1-\gamma)b[i] \text{ if } 1 \le i \le m \text{ and } i \ne l \\ q \text{ if } i = l \text{ where } q \in [-1, 1] \end{cases} \quad (4)$$

q, γ and l are randomly chosen values for each crossover. The random initialization of the k-th component is often called mutation. Mutation is necessary because a simple linear combination of $Z_{\boldsymbol{a}}$ and $Z_{\boldsymbol{b}}$ searches for MDD globally. Mutation introduces a new value to one dimension and helps the evolutionary algorithm get out of local optimum.

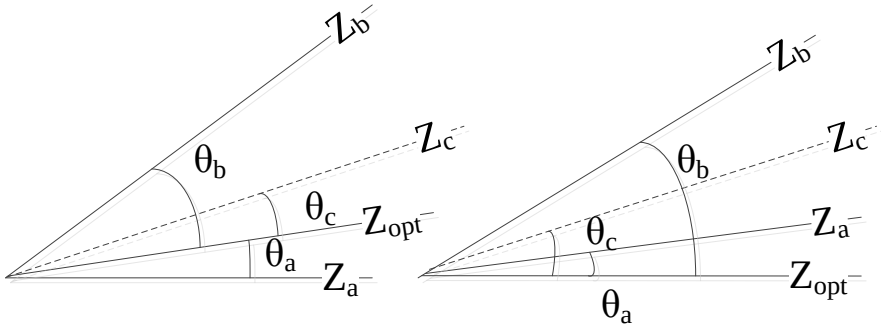

Figure 10. Linear combination of two p-dimensions

Lemma 3. A linear combination of two parents p-dimensions produces a new p-dimension better than at least one parent p-dimension.

Proof. Let Z_a and Z_b be two p-dimensions and Z_{opt} be the optimal p-dimension for D_T (Figure 10). The new p-dimension Z_c is created from the linear combination of Z_a and Z_b. Let a new p-dimension Z_c be created from Z_a and Z_b. Z_c would be in the vector space created by Z_a and Z_b. If Z_a and Z_b are in two different sides of Z_{opt} then Z_{opt} is also in the same vector space. The dimension that produces the maximum angle with Z_{opt} and lies in that same vector space is either Z_b or Z_a and all other p-dimensions must produce a smaller angle with Z_{opt}; hence $\theta_c < \theta_a$ and/or $\theta_c < \theta_b$. If Z_a and Z_b lie on the same side of Z_{opt}, let the p-dimension that creates the minimum (maximum)

angle with Z_{opt} be Z_a (Z_b). Any p-dimension created by the linear combination of Z_a and Z_b will produce a smaller angle with Z_{opt} compared with the angle between Z_b and Z_{opt}, hence $\theta_c < \theta_b$ (this is also true if Z_{opt} is not in the same plane). Thus in both cases Z_c creates a smaller angle with Z_{opt} compared to its parents $Z_a$and Z_b and thus produces smaller neighbor density compared to its parents (according to Lemma 2). Hence the resultant p-dimension produced from the linear combination of two parent p-dimensions induces smaller neighbor density than its parents.

Figure 11 shows the crossover algorithm. This algorithm selects two p-dimensions randomly from the working set A_T using the *rankSelect* function (lines 1-2). After two parent populations $Z_{\boldsymbol{a}}$and $Z_{\boldsymbol{b}}$ are selected, the new p-dimension $Z_{\boldsymbol{c}}$ is created in lines 4-7. Finally the new p-dimension $Z_{\boldsymbol{c}}$ is added to A_T.

Cultivation: In practical applications many data points have similar values and hence similar MDDs; therefore we can calculate the MDD of the current data point and keep that MDD for future use. In this way Orion has a higher chance of finding MDD for future data points. According to Lemma 1, $\boldsymbol{a} \propto \boldsymbol{\Sigma}_T^{-1}(D_t.\boldsymbol{v} - \boldsymbol{\mu}_T)$ is the vector along which the data point D_t has maximum AD. If we create a p-dimension along $\boldsymbol{a}$ and insert it into the p-dimension set A_T, it might induce minimum ND for future data points; otherwise it will eventually be deleted.

Deletion: As we create new p-dimensions using crossover or cultivation and insert them into A_T, we also delete old p-dimensions from A_T using the rank selection scheme (Figure 11). The ones with lower fitness will have a higher chance of being removed compared to the ones with higher fitness.

```
procedure Crossover(Population Set A_T)
1     Z_a ← rankSelect (A_T)
2     Z_b ← rankSelect (A_T − Z_a)
3     γ ← rand(0, 1)
4     c ← γa + (1 − γ)b
5     k ← rand(0, |c| − 1)
6     c[k] ← rand(-1, 1) // mutation
7     c ← c/L_c
8     A_T ← A_T ∪ Z_c
end procedure
function rankSelect(A_T)
9     list ← empty
10    for all Z_a ∈ A_T
11      list.add(Z_a, fitness(Z_a))
12    list ← sort(list) // sort dsc order
13    wheel ← rand(0, (|A_T|(|A_T|−1))/2)// a random value
14    index ← rand(0, |list|-1)
15    while(wheel > 0)
16      wheel ← wheel − (|list| - index)
17      index ← (index + 1) mod |list|;
18    select ← list[index]
end function
```

Figure 11. Crossover and Selection

1.4. Computation of Outlier Metrics

Orion uses the versions of ND and k-distance [24] which we have modified for data streams where they are computed from our data density function. The use of k-distance along with ND plays an important role in outlier identification: if the ND of an outlier is accidentally big (because of masking effect [24]), k-distance does not shorten proportionally and, hence, the outlier can be identified with the help of k-distance. However k-distance adds additional error as well: a true outlier might not have a small k-distance because of error and ND can be used to detect it. Thus in order to classify a data point as an outlier, Orion analyzes both ND and k-distance. Since we already

discussed the modified ND in Definition 1, here we only discuss the modified k-distance for data streams followed by the proposed data density function.

1.4.1. k-Distance

We adopt the concept of k-distance [24] but modify it for data streams. In [24] the k-distance of a data point is the minimum distance from a data point to its k-th nearest neighbor. So for a meaningful choice of k, the k-distance of a data point reveals how different it is compared to the majority of the data points in terms of distance. Since we deal with data streams, we use neighbor density, instead of neighbor count, as k. We calculate the k-distance of a data point along one p-dimension. Since each p-dimension has a different dispersion, k-distance becomes very p-dimension specific. In order to make our k-distance independent of the p-dimension, we scale the obtained distance with respect to the maximum dispersion along the p-dimension.

Definition 7. The k-distance of a data point D_t along $Z_{\boldsymbol{a}}$ is the minimum relative distance that has k neighbor density along $Z_{\boldsymbol{a}}$. Formally,

$$\begin{aligned} kDist_{Z_a}(D_t, k) &= kDist_{Z_a}(D_t.v[\boldsymbol{a}], k) \\ &= \frac{1}{\Delta_{Z_{\boldsymbol{a}}}} \min\{r_{Z_{\boldsymbol{a}}}\} |\, nDen\big(D_t.v[\boldsymbol{a}], r_{Z_{\boldsymbol{a}}}\big) \geq k) \end{aligned} \tag{5}$$

where $\Delta_{Z_{\boldsymbol{a}}}$ is the dispersion of values along $Z_{\boldsymbol{a}}$ and calculated as data points arrive.

A data point would require a large area to cover the same k if it is considerably different from the rest of the data points and vice versa. Thus, the k-distance of an outlier is large compared to that of an inlier.

1.4.2. Data Density Function

Our proposed data density function is based on a kernel probability density estimator. Several techniques exist in the literature to estimate data density function like histogram [55, 80], wavelet [81] and kernel estimation [36]. Among those techniques we chose the kernel probability density estimator (kernel estimator in short) for our approach. We shall justify our choice in the next few paragraphs.

The kernel estimator estimates the data density function based on the data values. For each data value v the kernel estimator increases the frequency of occurrence of v by p_v and increases the frequency of occurrence of each of the other values by a fraction of $1 - p_v$ which fits our requirements excellently. Due to data uncertainty, when we receive a data point d with value v, we cannot assert the data value with full confidence; therefore we cannot increase the frequency of occurrence of v by 1. Since the value v is uncertain, the data value v might be induced by data values other than v. Thus to address the uncertainty of data streams, we do not increase the frequency of occurrence of v by 1. The kernel estimator increases the frequency of occurrence of v by p_v and distributes the rest of the frequency of occurrence $(1 - p_v)$ into the other data values which are close to the value v. Formally, if $(x_1, x_2, \dots, x_n)$ are n sample one dimensional data points, their respective values are $(v_1, v_2, \dots, v_n)$ and the data density function $f(x)$ is defined by Equation (6) where $k(x)$ is called the *kernel function*. v_i can be a scalar or vector.

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} k(v_i - x) \tag{6}$$

The kernel function is responsible for distributing the frequency of occurrence induced by the data value v_i. Various researchers have proposed various kernel functions (e.g., Uniform kernel function, Triangle kernel function, Epanechnikov kernel function, Normal kernel function etc. [82]). Different kernel functions distribute the frequency of occurrence differently. Interestingly, the choice of a kernel function does not affect the data distribution function very much [36, 82]. Typically a kernel function distributes the frequency of occurrence into the neighbor data values which reside within a range called bandwidth (h) (Normal kernel function distributes the probability of occurrence from $-\infty$ to $+\infty$ [82]). A kernel function along with the bandwith (h) is denoted by $k_h(x)$ where $k(x) = hk_h(x)$. Although the choice of the kernel function is not very significant, the choice of the bandwidth is very important for data distribution estimation. A detailed discussion about the choice of kernel function and bandwidth selection can be found in [82]. In our approach we chose a data-based approach for bandwidth selection. Scott's rule provides a data-based bandwidth selection where $h = \sqrt{5}\sigma n^{-1/5}$ where σ is the standard deviation and n is the number of data points used for data distribution estimation [36].

In a kernel estimator the frequency of occurrence is distributed into an equal number of neighbor values for each data point, but in a variable kernel estimator the frequency of occurrence is distributed into different numbers of neighbor values for each data point. Hence at any specific point of time, if the data points are close to each other (in terms of value), the bandwidth becomes small, and if the data points are far (in terms of value) from each other, the bandwidth becomes large. Let $(x_1, x_2, \dots, x_n)$ be our data points with values $(v_1, v_2, \dots, v_n)$ at times $(T-n, T-n+1, \dots, T)$, and our corresponding

bandwidths be $(h_1, h_2 \ldots, h_n)$. The data distribution function $f(x)$ at time T becomes Equation (7) where $f_T(x)$ is the data distribution function at time T. In our approach we use the variable kernel estimator.

$$f_T(x) = \frac{1}{n}\sum_{i=1}^{n} k_{h_i}(v_i - x) \tag{7}$$

The use of the variable kernel estimator is twofold: first, the variable kernel estimator offers a variable bandwidth for each data point and the bandwidth can be computed on-the-fly using Scott's rule for each data point, and second, the variable kernel selects the bandwidth based on the recent data values only.

We modify the variable kernel estimator to address the temporal characteristic of a data stream. Recent data points are more interesting than old data points; therefore, when we estimate the data distribution function we need to consider the freshness of data points. Heuristically, the recent data items should have more *weight* than the old data points [65, 66, 83]. Here weight is defined as how a data point contributes to the data distribution function; thus, in our data distribution function, instead of giving all data points the same weight, we weight them according to their freshness. The most recent data point receives the highest weight while the oldest one receives the lowest weight. Exponential forgetting is a weight assigning scheme which gives more weight to the recent data points and less weight to the old data points, and the weight is decreased exponentially from present to past [84]. According to exponential forgetting, the relative weights among two consecutive data points are constant, called forgetting factor λ where $0 < \lambda \leq 1$. Among the two consecutive data points, the recent data point receives weight 1 and the old one receives weight λ. In case of a series of data points, at

any particular time the most recent data point receives weight 1 and all other data points receive the weights according to their relative positions to the most recent data point. If $(x_1, x_2, \dots, x_n)$ are the data points with data values $(v_1, v_2, \dots, v_n)$ at times $(T-n, T-n+1, \dots, T)$, respectively, the corresponding weights are $(\lambda^{n-1}, \lambda^{n-2}, \dots, 1)$. We weight the kernel function with an exponential forgetting factor. Adding the exponential forgetting factor λ to Equation (2), the data distribution function becomes Equation (8) where $\sum_{i=1}^{n} \lambda^{n-i}$ is the total weight. Another advantage of using the exponential forgetting factor is that it can be computed incrementally [84] which eases the on-the-fly implementation for data streams.

$$f_T(x) = \frac{\sum_{i=1}^{n} \lambda^{n-i} k_{h_i}(v_i - x)}{\sum_{i=1}^{n} \lambda^{n-i}} \tag{8}$$

$f_T(x)$ is the data density function we have available at any time T which represents the current data distribution of a data stream [5].

Orion computes the data density function (DDF) of the data point D_t along every p-dimension in the set A. When a data point arrives, Orion updates the DDFs of all p-dimensions. Each p-dimension has a DDF based on the data points arrived after its creation. The DDF of a p-dimension is the DDF proposed in Equation (8). It is based on the projection of the data values on a p-dimension. For any p-dimension $Z_{\boldsymbol{a}}$ the proposed DDF is defined by Equation (9).

$$f_{Z_{\boldsymbol{a}},T}(z_{\boldsymbol{a}}) = \frac{\sum_{t=\tau}^{T} \lambda^{T-t-\tau} k_{h_{Z_{\boldsymbol{a}},t}}(D_t.v[\boldsymbol{a}] - z_{\boldsymbol{a}})}{\sum_{t=\tau}^{T} \lambda^{T-t-\tau}} \tag{9}$$

where τ is the timestamp when $Z_{\boldsymbol{a}}$ is created and $h_{Z_{\boldsymbol{a}},t}$ is the bandwidth along $Z_{\boldsymbol{a}}$. The DDF does not assume any particular fixed/standard data distribution, but is adjusted on-the-fly. In addition, it uses a variable bandwidth for the kernel estimator which eases the online incremental implementation and makes the DDF adaptive to the dispersion of the data points.

1.4.2.1. Data Density Function Implementation

The kernel estimator requires a large amount of computation. Binned implementation is a popular, fast implementation for the kernel estimator [85]. In this approach the entire range of data points is divided into some equally spaced bins and data are distributed into bins according to their data values. Each bin has a representing value and all the data points in a bin are represented by the representing value. The key idea is that most values are naturally close to each other and binned implementation reduces the number of evaluations; but this popular binned implementation still requires multiple passes of the data points and cannot be computed incrementally.

In our approach we divide the entire range of data values into equally spaced bins. A representing value is selected for each bin ($b_0, b_1, b_2,$ ... in Figure 12). Instead of binning the data points for each bin, we store the value of the data density function $f(b_i)$ of the representing value b_i, cumulative data density function $F(b_i)$ and the derivative of the data density function $f'(b_i)$. $f(b_i)$, $F(b_i)$ and $f'(b_i)$ are stored for each representing value b_i. $f(b_i)$, $F(b_i)$ and $f'(b_i)$ are the sum of the kernel estimations, sum of the cumulative kernel estimations and the sum of the derivative of the kernel estimations for all the data points received, respectively. The kernel function, the cumulative kernel function and the derivative of the kernel function for each representing value are

computed on-the-fly and added to the previous sums; hence this is an online incremental implementation.

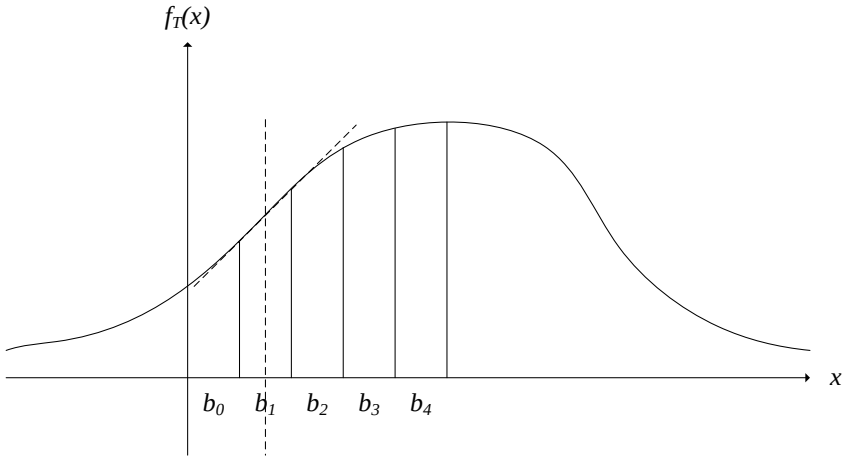

Figure 12. Binned implementation of kernel estimator

Figure 12 shows the binned implementation of our proposed probability density function. By carefully selecting the bin width we can assume each bin to be a trapezoid as shown in Figure 12 and we can approximate the probability of occurrence of a data value within a bin. The top of the trapezoid is a straight line (shown in Figure 12 as the dotted line touching the probability density function) and we store the passing point as well as the derivative; hence using the straight line equation of the line we can estimate the probability of occurrence of any data value within a bin.

Practically it is impossible to know the entire range of values for all possible p-dimensions, hence we do not assume to have the entire range of data values for each possible dimension. Therefore, instead of creating a static set of bins, we create a bin whenever necessary, that is whenever the DDF of that particular bin is updated. In order to create bins dynamically we approximate the value of the bin width for each p-dimension $Z_{\boldsymbol{a}}$ from its variance $\sigma_{\boldsymbol{a}}$. The bin width should be such that the average error

value is minimum; Section 1.7.2 provides a detailed discussion about bin width selection for our approach.

```
1  procedure update(dataItem d, timestamp t)
2    s_1 ← λs_1 + d; // s_1 is the sum of data value and λ
     is our forgetting factor
3    s_2 ← λs_2 + d^2; // s_2 is the sum of the square of the
     data value
4    ω ← λω + 1; // ω is the total data weight
5    μ_1 ← s_1/ω; // μ_1 the first moment
6    μ_2 ← s_2/ω; // μ_2 the second moment
7    σ ← √(μ_2 − μ_1^2); // σ is the standard deviation
8    h ← √5 σ ω^(−1/5); // h is the bandwidth
9    c ← h/binWidth; // c is the cell count
10   b ← indexLookup(d); // b is the middle cell
11   for i = b − c to b + c, // i is the index of the
     cell, where i ≥ 0 and i ≤ maximum index.
     // b_i is the representing value of the bin/cell(c_i)
     and α_i and β_i are the starting value and the end
     value of the bin.
     // distance between two consecutive time stamp is 1.
12     c_i[f(b_i)] ← λ^(t − c_i[timestamp]) c_i[f(b_i)] + k_h(d − b_i);
13     if (k_h(d − x_i) is not discontinuous at x_i
14       c_i[f'(b_i)] ← λ^(t − c_i[timestamp]) c_i[f'(b_i)] − k'_h(d − b_i);
15     else
16     c_i[f'(b_i)] ← λ^(t − c_i[timestamp]) c_i[p'(b_i)] − (k_h(d − β_i) −
       k_h(d − α_i))/binWidth;
17     c_i[timestamp] ← t;
18     c_i[F(b_i)] ← c_i[F(b_{i−1})] + c_i[f(b_i)]
19   end for
20 end procedure
```

Figure 13. Update data density function

Figure 13 shows the online incremental update algorithms for our proposed data density function. The update algorithm updates the DDF as each data point comes for every p-dimension. The update algorithm takes a data point and its timestamp as input. It starts by updating the weighted summation (lines 2-3), where s_1 is the weighted summation of

the data values and s_2 is the weighted summation of the square of the data values. The ω in line 4 is the total weight of the data. s_1 and s_2 are required to calculate the current standard deviation σ and hence the bandwidth h. In line 9 we calculate the number of cells we need to update. Some kernel functions update the values in the range from $-\infty$ to $+\infty$ (e.g., Normal kernel function [82]); in that case we restrict it to $minValue$ and $maxValue$, which represent the minimum and maximum allowable values for a data point, respectively. Now for each bin we update the sum of the kernel function and the latest timestamp when the bin is updated. If the kernel function is continuous at the representing point b_i then we store the derivative of the kernel function at b_i; otherwise we store the gradient from the starting point α_i to the end point β_i of the bin. The probability lookup algorithm is fairly simple; it finds the appropriate bin which contains the sum of the kernel function values. Finally the probability is achieved by dividing the sum of the kernel function values by the sum of the weights.

1.5. Outlier Detection

A data point is an outlier if it has a considerably small neighbor density (ND) or a large k-distance compared to the normal data points. However the definition of "considerably small/large" is application-dependent and no well accepted definition of "considerably small/large" exists in literature [10, 12]. Orion solves the problem by quantizing the values of ND (and also of k-distance) into three groups: (1) small, (2) average and (3) large where they represent the three sets of data points with small, average and large neighbor density (k-distance), respectively. Instead of using user-defined thresholds for small, average and large for ND or k-distance, Orion quantizes them using the concept of co-clustering [78].

Co-clustering clusters the data points into three clusters based on their NDs and three clusters based on the k-distances; hence the total number of clusters with a unique combination of ND and k-distance is nine (three times three). We initialize the cluster centers with the minimum (for small), average (for average) and maximum (for large) values, and the cluster centers are updated with the arrival of every new data point. In this way we avoid using any user-specified threshold for the boundaries of small, average and large and let the boundaries emerge from co-clustering.

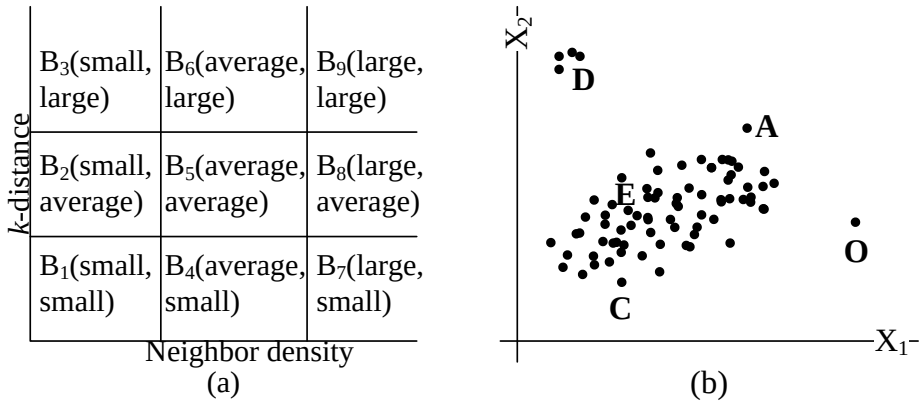

Figure 14. Neighbor density/k-distance space

Orion creates nine blocks for nine clusters in a two-dimensional space (Figure 14a) where the ND goes along the horizontal axis and k-distance goes along the vertical axis. Each block is a cluster and has a cluster center defined by a tuple (ND, k-distance). Blocks B_1, B_2 and B_3 share a common ND and different k-distances as their centers, and B_1, B_4 and B_7 share different NDs and a common k-distance (other blocks are defined similarly). As a data point arrives and its ND and k-distance are calculated, Orion finds the appropriate block based on its ND and k-distance, updates the corresponding centers, and determines whether it is an outlier or inlier based on the block properties.

B_1 and B_2 correspond to the blocks where every data point has a small ND and small/average k-distance compared to other data points. If the ND is small, the data point has very few neighbor data points and can be an outlier; however the k-distance is also small/average which shows most of the data points can be reached within a small/average distance. These data points are on the periphery of a group of data points (A and C in Figure 14b). Every data point belonging to this block will be identified as an outlier if it is more inclined to the vertical axis compared to the horizontal axis since those data points have large k-distance and small ND. B_3 is an ideal case of outliers where every data point has a small ND and a large k-distance value, hence it is identified as an outlier (O in Figure 14b). Blocks B_4 and B_5 have average ND and small/average k-distance. The NDs of the data points belonging to these blocks represent the normal behavior of the data and the k-distances represent a close (B_4)/typical (B_5) proximity to other data points. Data points belonging to B_4 or B_5 are identified as inliers (E in Figure 14b). B_6 shows a large k-distance with average ND; this may happen because of the masking effect [24] where a group of outliers cluster together and far from the rest of the data points. Thus B_6 is considered as an outlier block (D in Figure 14b). Blocks B_7 and B_8 consist of ideal inliers that have a large ND and small/average k-distance. Finally B_9 is an invalid block because a data point that simultaneously has a large ND and large k-distance is impossible to exist. If a data point has large ND which means it has plenty of data points in its close proximity, thus it would require a small k-distance to incorporate k percents of the data points. Any data point belonging to B_9 is invalid and therefore can be considered as an inlier or outlier.

```
function detectOutlier(nDen, kDist, hCenters, vCenters)
   1  dist ← MAX
   2  (h, v) ← (-1, -1)
   3  for i = 1 to 3
   4    for j = 1 to 3
   5      lDist ← |nDen-hCenters[i]|+|kDist-vCenters[j]|
   6      if(lDist < dist)
   7        dist ← lDist
   8        (h, v) ← (i, j)
   9  update(hCenters[h], nDen)
  10  update(vCenters[v], kDist)
  11  if((h,v) = B4 or B5 or B7 or B8)
  12    detectOutlier ← false
  13  else if((h,v) = B3 or B6 or B9)
  14    detectOutlier ← true
  15  else if((h,v) = B1 or B2)
  16    if(cosine((0, 1), (nDen, kDist)) > cosine((0, 1),
      (hCenters[1], vCenters[3]))
  17      detectOutlier ← true
  18    else
  19      detectOutlier ← false
end function
function cosine((x1, y1), (x2, y2))
  20  cosine ← (x1x2 + y1y2) / (sqrt(x1^2 + y1^2) sqrt(x2^2 + y2^2))
end function
```

Figure 15. Outlier detection algorithm

Figure 15 shows the outlier detection algorithm *detectOutlier*. Orion invokes this algorithm once it computes the ND and k-distance for the newly arrived data point. *detectOutlier* compares the ND and k-distance with all cluster centers (lines 3-8) and finds the closest cluster center. Once the closest cluster center is found, *detectOutlier* updates it and identifies whether the data point is an outlier or not based on which block the data point belongs to. If the data point belongs to any of the blocks B_4, B_5, B_7 or B_8, it is identified as an inlier (lines 11-12); if it belongs to any of B_3, B_6 or B_9, it is identified as an outlier. If the data point belongs to B_1 or B_2 it is identified as an outlier

if it is more inclined to the vertical axis (lines 16-17); otherwise it is identified as an inlier.

1.6. The Orion Algorithm

Orion consists of two stages: the initialization stage to initialize the data structures and learn the parameters, and the incremental stage to determine whether a newly arrived data point is an inlier or outlier.

1.6.1. Initialization Stage

Orion initializes all of its data structures (mean $\boldsymbol{\mu_T}$ and covariance $\mathbf{\Sigma_T}$) and learns the required parameters for the data density function from the bootstrapping rounds which are the first few rounds of data specified by the user. The first step of initialization is the forgetting factor selection. Brailsford et al. [84] proposed a method for forgetting factor selection which is discussed in Section 1.7.1. In the second step Orion creates a set of p-dimensions A as the eigenvectors of the covariance matrix $\mathbf{\Sigma}_T$ and partitions A into two sets A_{in} and A_{out} randomly. The reason Orion chooses eigenvectors because they are along the maximum variance of the attributes. Once the dimension set A is populated, Orion updates the data density functions along all the p-dimensions in A and calculates the ND and k-distance for all bootstrapping rounds. Finally Orion initializes the cluster centers for co-clustering: small as 0 (minimum value), large as 1 (maximum value) and average as mean (average ND/k-distance value of the data points in the bootstrapping rounds). Orion then moves to the incremental stage.

1.6.2. Incremental Stage

Orion processes every data point online and incrementally. Once a new data point D_T arrives, the algorithm ProcessData shown in Figure 16 is invoked. It takes D_T, user-defined neighbor distance r and k for k-distance as input, identifies whether D_T is an outlier or inlier and updates its internal data structures for future use. Figure 17 accompanies Figure 16 to show the flow diagram of Orion.

```
procedure ProcessData(Data point D_T, neighbor distance r, k)
1    μ_T ← (T/(T+1)) μ_{T-1} + (1/(T+1)) D_T.v  // update mean
2    Σ_T ← ((T-1)/T) Σ_{T-1} + (D_T.v − μ_T)(D_T.v − μ_T)'
3    AD_{Z_a}(D_T)^2 ← (D_T.v − μ_T)' Σ_T^{-1} (D_T.v − μ_T)  // Lemma 1
4    if(AD_{Z_a}(D_T)^2 > meanAD_{T-1})      A_T ← A_out
5    else A_T ← A_in  // find appropriate set
6    meanAD_T ← ((T-1)/T) meanAD_{T-1} + (1/T) AD_{Z_a}(D_T)^2
7    densityList ← {nDen_{Z_a}(D_T, r_{Z_a}) | Z_a ∈ A_T}//r_{Z_a} = 2δ_{Z_a} r
8    sortedList←sort(densityList)//Asc-order
9    Z_a ← sortedList.top()//choose item within ND
10   nDen ← nDen_{Z_a}(D_T, r_{Z_a})
11   kDist ← kDist_{Z_a}(D_T, k)
12   if(detectOutlier(nDen, kDist, hCenters, vCenters))
     isOutlier ← true
13   else isOutlier ← false
14   for all Z_a ∈ A_T updateFitness(Z_a)
15   A_T ← evolve(A_T, D_T, Σ_T)
16   for all Z_a ∈ A updateDensity(Z_a, D_T)
17   return isOutlier
end procedure
```

Figure 16 ProcessData Procedure

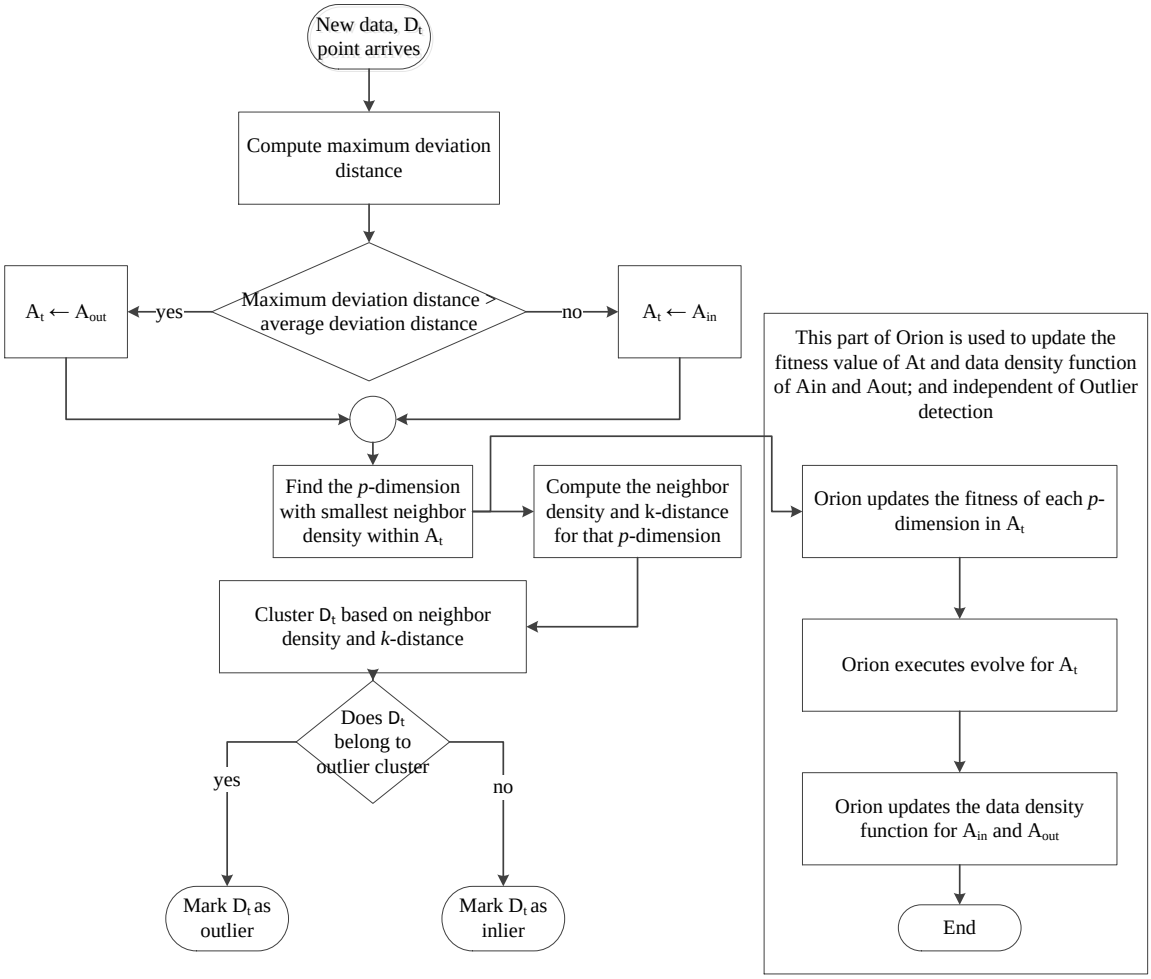

Figure 17. Work flow of Orion

At first, Orion computes the maximum deviation distance for the newly arrived data point (Figure 17). The corresponding operation can be found is lines 1-3 in Figure 16 where ProcessData starts with updating the mean $\boldsymbol{\mu}_T$ and covariance matrix $\boldsymbol{\Sigma}_T$ (lines 1-2) and computes the maximum AD (line 3). If the maximum AD is smaller than the average maximum AD, $meanAD_{T-1}$, ProcessData selects the set A_{in} and stores it in the working set A_T; otherwise it selects A_{out} for the working set A_T (lines 4-7, the top decision box in Figure 17). ProcessData selects one p-dimension from A_T which incurs the smallest ND (lines 6-9) and calculates the ND and the k-distance of D_T along that p-

dimension (lines 10-11), which is also shown in the flow diagram of Orion. D_T is identified as an outlier if it has a considerably low neighbor density and considerably high k-distance compared to the normal data points using the detectOutlier function in line 12. The detectOutlier function examines the cluster to which D_T belongs; if it is an outlier cluster, detectOutlier identifies the D_T as an outlier; otherwise inlier (the bottom decision box in Figure 17). In the next step, Orion updates the existing data structures for future processing (the shaded region in Figure 17). ProcessData then sorts the p-dimensions into ascending order based on the NDs of the data point D_T along those dimensions. Orion updates the fitness (rank) of the selected p-dimensions and performs the evolutionary steps by adding a new dimension to A_T and removing an old one from A_T (line 15). Finally ProcessData updates the data density function along every p-dimension in A which is a union of the sets A_{in} and A_{out} (line 16) for future use. At this point the processing of D_T is complete and Orion moves to the next data point.

1.7. Parameter Selection

In the course of the description of Orion we have proposed the use of the parameters forgetting factor λ, bin width and initial population without giving an idea about how those parameters are selected. In this section we describe the parameter selection strategy for each parameter.

1.7.1. λ Selection

Data points have a temporal characteristic in a data stream. Data points are interesting for a specific amount of time. Moreover, data points have a temporal dimension and they are correlated with respect to time. The data points which are close to a given data

point x are more correlated to x than the data points which are far from x. Heuristically, we weight the data items with an exponential forgetting factor λ ($0 < \lambda \leq 1$), which implies that the recent data points will receive higher weights than the old data points. If the value selected is close to 1, the data distribution function will remember more history than the data distribution functions where the value is close to 0. The value 0 implies that only the current data point is used for the data distribution function while the value 1 implies a complete history where all previous data points will be used for the data distribution function. Intuitively, the more history the user wants, the larger the λ should be. One important factor for an appropriate λ selection is the correlation between consecutive data points. Intuitively, data collected more frequently tend to be more correlated to each other. Above all, it is a highly application-dependent parameter in time series analysis or data streams. To select an appropriate forgetting factor, we adopt the static forgetting factor selection method proposed by [84]. The λ selection is based on a bootstrapping method; therefore, it requires an initial dataset to select an appropriate λ. This model portrays the time series as an auto-regression model with a forgetting factor. A part of the initial dataset is used to train the auto-regression model and the rest is used for error estimation. According to the method, the λ which gives the minimum error is the right choice for λ. Madsen [66] pointed out that for an auto-regressive model, λ^c should be greater than 0.75 where c is the number of components in the model; therefore it is sufficient to consider the values which are greater than 0.75. Brailsford et al. use a first order linear model for their approximation, and so do we. The next section discusses the selection method for another important parameter for our approach named bin width.

1.7.2. Bin Width Selection

Bin width is an important parameter for the correctness of the data density function. Instead of exactly representing the data density function, our binned implementation approximates the data density function. The bin width should be such that the average error is minimum. This section presents the maximum error bound for a specific choice of bin width and provides a guideline for bin width selection.

Let us say $f(x)$ is the actual data density function and $f_a(x)$ is the approximate data density function for a dimension by our binned implementation where x is a random variable. At point x our approximate frequency of occurrence is $f_a(x)$ and true frequency of occurrence is $f(x)$; therefore the error induced at point x is $\varepsilon(x) = |f(x) - f_a(x)|$. Here, $f(x) = \frac{1}{N}\sum_{i=0}^{N} k_{h_i}(v_i - x)$ and $f_a(x) = f(b) + (x - b)f'^{(b)} = \frac{1}{N}\sum_{i=0}^{N}\left[k_{h_i}(v_i - b) - (x-b)k'_{h_i}(v_i - b)\right]$ where b is the representing value for the nearest bin and v_i is the data value. For the sake of simplicity we omit the exponential forgetting factor from the following error analysis.

$$\varepsilon(x) = \left|\frac{1}{N}\sum_{i=0}^{N} k_{h_i}(v_i - x) - \frac{1}{N}\sum_{i=0}^{N}\left[k_{h_i}(v_i - b) - (x-b)k'_{h_i}(v_i - b)\right]\right|$$

$$= \left|\frac{1}{N}\sum_{i=0}^{N}\left[k_{h_i}(v_i - x) - k_{h_i}(v_i - b) + (x-b)k'_{h_i}(v_i - b)\right]\right|$$

$$\leq \frac{1}{N}\sum_{i=0}^{N}\left|k_{h_i}(v_i - x) - k_{h_i}(v_i - b) + (x-b)k'_{h_i}(v_i - b)\right|$$

$$= \frac{1}{N}\sum_{i=0}^{N}\left|\varepsilon_{v_i}(x)\right|$$

Here $\varepsilon_{v_i}(x)$ is the error induced by the data value v_i, where $\varepsilon_{v_i}(x) = k_{h_i}(v_i - x) - k_{h_i}(v_i - b) + (x - b)k'_{h_i}(v_i - \alpha)$. The total error can be minimized by minimizing $\varepsilon_{v_i}(x)$ [5].

$$\varepsilon_{v_i}(x) = k_{h_i}(v_i - x) - k_{h_i}(v_i - b) + (x - b)k'_{h_i}(v_i - b)$$

$$= \frac{3}{4h_i}\left[1 - \left(\frac{v_i - x}{h_i}\right)^2 - 1 + \left(\frac{v_i - b}{h_i}\right)^2 + (x - b)\left[-\frac{2(v_i - b)}{{h_i}^2}\right]\right]$$

$$= \frac{3}{4{h_i}^3}[(v_i - b)^2 - (v_i - x)^2 - 2(x - b)(v_i - b)]$$

$$= \frac{3}{4{h_i}^3}[(x - b)(2v_i - x - b) - 2(x - b)(v_i - b)]$$

$$= \frac{3}{4{h_i}^3}(x - b)(2v_i - x - b - 2v_i + 2b)$$

$$= -\frac{3}{4{h_i}^3}(x - b)^2$$

If the random variable x goes to the bin where b is the representing value, the average error in the bin is defined by $\varepsilon_{v_i}^{(b)}$ for data value v_i as follows:

$$\varepsilon_{v_i}^{(b)} = \frac{1}{2\delta}\int_{b-\delta}^{b+\delta} -\frac{3}{4{h_i}^3}(x - b)^2 dx$$

$$= \left(\frac{1}{2\delta}\right)\left(-\frac{3}{4{h_i}^3}\right)\int_{-\delta}^{\delta} y^2 dy$$

$$= \left(\frac{1}{2\delta}\right)\left(-\frac{3}{4{h_i}^3}\right)\left[\frac{y^3}{3}\right]_{-\delta}^{\delta}$$

$$= -\left(\frac{\delta^2}{4{h_i}^3}\right) \text{ where } 2\delta \text{ is the bin width}$$

and therefore,

$$\varepsilon_{v_i}^{(b)} = \left(\frac{\delta^2}{4{h_i}^3}\right) \tag{10}$$

$\varepsilon_{x_i}^{(b)}$ is the average error in a bin induced due to approximation. From Equation (10) we can see the induced error is independent of the bin location. Therefore, each bin would have an equal average error. The error will be minimized if we minimize the average error. Hence, the minimum error occurs when $\frac{d}{d\delta}\left(\varepsilon_{v_i}^{(b)}\right) = 0$. $\frac{d}{d\delta}\left(\varepsilon_{v_i}^{(b)}\right) = \frac{d}{d\delta}\left(-\frac{\delta^2}{4{h_i}^3}\right) = -\frac{\delta}{2{h_i}^3}$; therefore, $-\frac{b}{2{h_i}^3} = 0$ if $\delta = 0$ or $h_i \longrightarrow \infty$. Obviously, the average error is minimum when the bin width is zero. We can reduce the error by choosing a bin width much smaller than the bandwidth [5]. If the bin width is greater than the bandwidth, only one bin's frequency of occurrence will be updated. In that case if we consider one bin as one histogram, then the kernel estimation is turned into the histogram based approach [5].

Minimizing the bin width increases the number of bins; hence the greater the number of bins the lower the error. Fan and Marron [85] mentioned that four hundred bins is often optimal, fewer than four hundred bins often deteriorates the quality of the results and

more than four hundred bins offers very little improvement. In our technique we use the optimal four hundred bins.

In our implementation we do not assume to have the entire range of data values for each possible dimension. Therefore, instead of creating a static set of bins, we create a bin whenever necessary, that is whenever the DDF of that particular bin is updated. In order to create bins dynamically we approximate the value of the bin width for each dimension $Z_{\boldsymbol{a}}$ from the variance $\sigma_{\boldsymbol{a}}$. According to Chebyshev's inequality [59], regardless of the data distribution, more than 90% of data points are within six standard deviations away from the mean. Hence we choose the optimal four hundred bins for six standard deviations (bin width $2\delta_{Z_a} = 6\sqrt{\boldsymbol{a}'\boldsymbol{\Sigma}_{\boldsymbol{T}}\boldsymbol{a}}/400$) for any p-dimension.

We embed the λ and bin width selections in our approach. Hence our approach does not require the user to select these two parameters; rather our approach automatically chooses the appropriate values for them.

1.7.3. Initial Population Selection

The strength of the evolutionary algorithm is that it can start from any random set of solutions [79]. So theoretically we can start from any random set of p-dimensions. However, in our case we compute the covariance matrix using the bootstrapping dataset and compute the eigenvectors from the covariance matrix and initialize our initial population set with eigenvectors. We randomly divide the set of eigenvectors into two sets and put them in A_{in} and A_{out}. The use of eigenvectors has one advantage over random population initialization in that it ensures all the starting p-dimensions to be independent of one another and there would be at least one p-dimension along which

the data points are far from their mean (because according to Lemma 1, a data point is furthest from its mean along the eigenvector corresponding to the minimum eigenvalue). Therefore, if future data points are similar to bootstrapping data points, then they might incur minimum neighbor density along that p-dimension.

2. Outlier Detection for Multiple Data Streams

Multiple data streams consist of a set of data streams where each data stream produces an infinite sequence of data points accompanied with explicit or implicit timestamps and data points from different streams may or may not be correlated. These data points may have two kinds of correlations: one is called temporal correlation where data points from the same stream are correlated and one is called cross correlation where data points from different streams are correlated. Outlier detection for single streams compares a data point in a stream with respect to the history data points from that same stream in order to identify whether the data point is an outlier. In case of multiple data streams, such identification can be done either by (1) comparing the data point with the history data points from the same stream that carries the data point, (2) comparing the data point with the data points from the other correlated streams, or (3) using a combination of both (1) and (2). The opportunity of having multiple data streams to compare allows richer semantics across the data streams to be taken into consideration which would lead to better detection accuracy. Our algorithm, which is discussed in detail in Section 2, uses Option (3) so that it can take advantage of both Options (1) and (2).

Figure 18 shows a generic outlier detection methodology with multiple streams. Each data source produces a sequence of data points and sends them to the outlier detection component. The streams produced by multiple data sources can be homogeneous or heterogeneous, synchronous or asynchronous, and some or all of them may be correlated. For each data point received, the outlier detection component identifies whether it is an outlier or inlier and releases it for further processing (Figure 18).

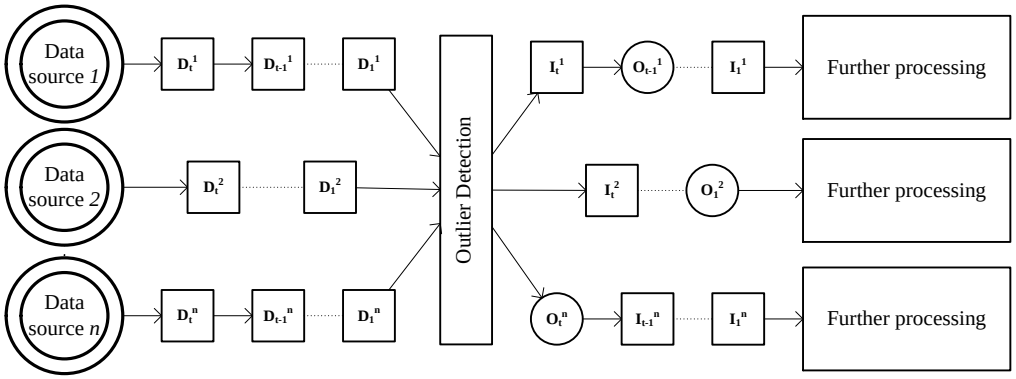

Figure 18. A generic multiple data streams model with outlier detection

We propose the idea of two-phase outlier detection that exploits the temporal and cross correlations among data points. In the first phase, outliers would be detected based on the data points from only one stream using Orion for single streams; and in the second phase, outliers would be detected based on the data points from all correlated streams.

Figure 19 depicts the idea of two-phase outlier detection. In the first phase, a data point D_t^i (the t-th data point from the i-th stream) is identified as an outlier if it is considerably different from other data points from the same stream or an inlier otherwise. For the data points that were not found to be outliers in the first phase, they will be sent to the second phase for further detection. In this phase, a data point is identified to be an outlier if it violates the expected cross correlations among the data

points in the streams that have cross-correlation with the stream from which the data point comes. In the second phase, we process the group of correlated data points together that arrive at the same point in time in order to detect outliers among them. In summary, a data point is said to be an outlier if it violates any of the correlations (temporal correlation in the first phase or cross correlation in the second phase).

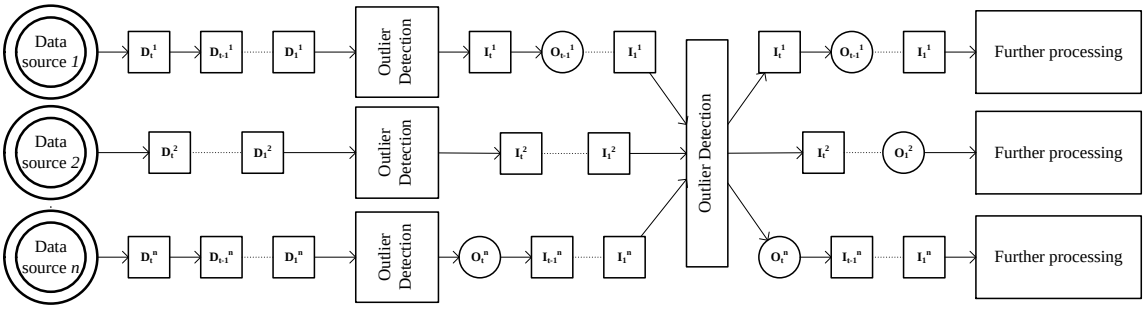

Figure 19. Two-phase outlier detection for multiple data streams

The second phase requires a novel idea of outlier detection that can compare data points from multiple streams to identify outliers. This is because data points from multiple streams could be very different from one another and their values may not be directly comparable. Moreover, the asynchronous and dynamic nature of multiple streams make it difficult to learn the cross correlation. On top of that, the notion of outlier in heterogeneous streams is yet to be defined. Since the second phase tries to exploit cross correlation among the data points from multiple streams and we may or may not have more than one correlated data points at any point in time, instead of processing one data point only, we have to process a set of correlated data points where each of them may be originated from multiple streams. We called our algorithm Wadjet.

2.1. Overview of Wadjet

Wadjet is composed of two phases. In the first phase, Wadjet uses Orion to detect outliers based on individual data streams. The data points that are identified as outliers are stored in the set of outliers; otherwise they are stored in the set of inliers. In the second phase, Wadjet further processes the set of inliers. It captures the cross correlations, if any, among the data points (that are in the set of inliers) from multiple streams. Each stream produces a sequence of data points, and two random data points which are picked from two different streams may not have the same cross-correlation; thus we need to compute the cross-correlation among the data points. However, theoretically we can compute the cross-correlation between any two data points chosen from two different streams. Imagine two sensors S_1 and S_2 producing temperature readings at every hour, such as $D_1^1, D_2^1, D_3^1, \dots, D_T^1$ and $D_1^2, D_2^2, D_3^2, \dots, D_T^2$. We can compute the cross-correlation between any pair of their data points (D_i^1, D_j^2) where i and j can be any value. Hence, the theoretical possible cross-correlation pair could be infinite (because each data stream can produce an infinite amount of data points). Practically, it is impossible to compute an infinite number of possible cross-correlations; and therefore, it is necessary to define the context for each stream under which the cross-correlation would be captured. Consider n data streams, each of which produces a sequence of data points. $\{D_1^1, D_2^1, \dots, D_6^1\}$, $\{D_1^2, D_2^2, \dots, D_5^2\}$ and $\{D_1^n, D_2^n, \dots, D_5^n\}$ are data points from streams 1, 2 and n, respectively, where D_i^j is the data point at time *i* from stream *j*. The cross-correlation between D_1^1 and D_1^2 is not the same as the cross-correlation between D_1^1 and D_2^2. Therefore, we need to identify the data points from the streams whose cross-correlation would be captured. Ideally, we

should select the data points in such a way that the cross-correlation is maximized, which raises the problem of time series alignment [86]. A time series alignment algorithm aligns two series in such a way that they exhibit maximum correlation; however, existing time series alignment algorithms require the availability of the entire set or a large subset of data points [86, 87], which is not possible in data streams.

Wadjet captures the cross-correlations among the data points that arrive simultaneously. Figure 20 shows the flowchart of Wadjet. Wadjet starts collecting all available data points from the streams and starts detecting outliers among them. Tracking cross-correlations among time series is an active research area [48, 86]. A number of outlier detection techniques also track multiple streams for outlier detection [32, 39, 88]; however, they assume that all the data points from multiple streams have the same value (a form of cross-correlation), and hence, any data point which breaks that assumption is identified as an outlier. We believe this assumption is too restrictive and the existence of correlation is unknown and dynamic. In order to compute the cross-correlations among the data points from multiple streams, we compute the correlation matrix that has the pair-wise correlations of all attributes of all data points from all streams.

Wadjet compares one attribute value to the other correlated attribute values to detect outliers. However, not all attributes are significantly correlated with other attributes. Therefore, comparing one attribute value to an uncorrelated attribute value would not produce any meaningful result. Thus, Wadjet needs to find the set of correlated attributes and compare their values to measure their similarity and detect outliers. Each set of correlated attributes is called a cluster.

Once Wadjet computes the cross-correlation between the attributes, Wadjet groups the significantly correlated attributes of the same set of data points into clusters. Two attributes are called correlated if their coefficient of determination (coefficient of determination between two attributes is the square of Pearson correlation between them [89]) is not significantly different from the perfect coefficient of determination (which is 1). We use coefficient of determination because if two attributes have the perfect coefficient of determination, the variation of one attribute can perfectly be explained by the second attribute. The perfect coefficient of determination also implies that there exists a linear function between them. If two attributes x and y have perfect coefficient of determination, they can be represented as a linear function of one another such as $y = px + q$, where p and q are two regression coefficients [90].

In order to group the attributes into clusters, Wadjet starts with an empty cluster. One attribute is chosen randomly and placed in the empty cluster. This attribute is called the cluster head. Wadjet chooses the first attribute randomly so that every attribute has an equal chance of being a cluster head. Once the cluster head is chosen, for each attribute other than the one already in the cluster, Wadjet checks whether the attribute has a significant correlation with the cluster head; if yes, then the attribute is added to the newly formed cluster. This process continues until no more attribute can be added to the newly formed cluster. Wadjet continues the same cluster formation process with the remaining set of attributes that do not belong to any cluster already formed.

The objective of the cluster formation step of Wadjet is to identify a set of attributes that are correlated. Once Wadjet can identify the set of correlated attributes, it can compare the attribute values with each other to identify outliers. Our clustering scheme ensures

that each attribute in a cluster is significantly correlated with the cluster head. This is necessary because outlier detection relies on similarity measurement between two data points. However, computing similarity between two attributes is very difficult if their values are not equal but correlated. Thus, we need to make them equal in order to measure the similarity between two attributes. In order to make them equal, all the attributes in a cluster must be correlated to one attribute, so that their values can be made equal (we call this *"equating the values").* In our clustering scheme, all the attributes in one cluster is significantly correlated with the cluster head, thus we can equate the attributes of a cluster to the cluster head using a linear regression function.

By construction, each attribute in a cluster has a significant correlation with the cluster head. By definition, if two attributes are significantly correlated, there exists a linear regression function between them. If a cluster has five attributes, X_1, X_2, X_3, X_4, and X_5 and X_5 is the cluster head, then X_5 can be represented as a function of X_1; similarly it can be represented as a function of X_2, X_3, and X_4 as well. We can compute four different values of X_5, each computed using one of X_1, X_2, X_3, and X_4. Each of these values is called the expected value of X_5 obtained from X_i where $i \in [1,4]$; we call this "*equating the value of X_i to X_5."* Theoretically, X_5 has five different values four of them are computed by equating other attributes and one of them is the value of X_5; and all of them must be the same. Therefore, all the values from X_1 to X_5 are now equal and Wadjet can easily measure the similarity between them. Thus, we solve the problem of similarity measurement among the attributes of a cluster by equating them to the cluster head.

For each cluster, Wadjet computes the expected values of the cluster head by equating each of the attributes in that cluster with the cluster head. If any of the expected values of the cluster head is significantly different from other expected values, the attribute is most likely an outlier. This is because based on our obtained correlation, all of these values (expected values and the obtained value) should be equal. Since, one value is not equal, it is non-conformist to our obtained correlation and thus it is very likely an outlier. Wadjet identifies the data point which contains that attribute as an outlier.

Wadjet identifies the outliers for each cluster separately because the attributes in a cluster is significantly correlated with the cluster head. Thus each attribute is compared with only its correlated set of attributes. Two attributes from two different clusters may not be correlated at all and, therefore, comparing them to detect outliers would not produce any meaningful result. Therefore, Wadjet compares each attribute with the other attributes in the same cluster only.

Figure 20 shows the flow diagram of Wadjet. In the first step, Wadjet executes Orion for each data point and identifies the sets of outliers and inliers. Wadjet further processes the set of inliers to update the cross-correlation matrix. In the following step, Wadjet clusters the attributes and equates the attribute values to their respective cluster head values. Finally, Wadjet identifies the outliers from each cluster by finding significantly different attribute values (compared to other attribute values in the same cluster) and adds the outliers to the previously constructed set of outliers. Wadjet continues this process until no cluster left for processing. The details of Wadjet are explained in the next few sections.

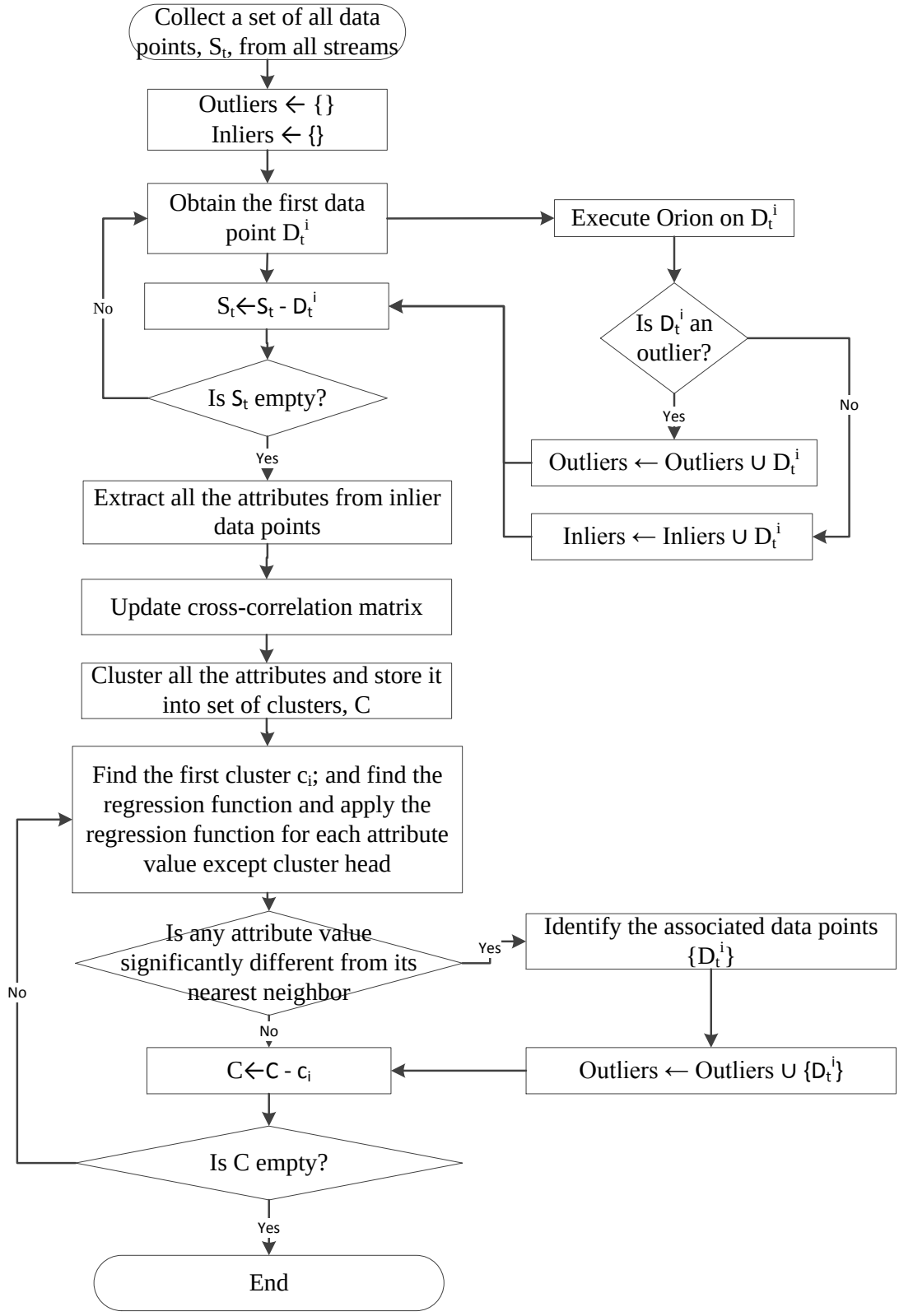

Figure 20. Flowchart of Wadjet

2.2. Cross-correlation Computation

In order to detect outliers from multiple streams, Wadjet makes use of the cross-correlation among the data points from multiple streams. Each stream is an infinite sequence of data points; hence, computing the cross-correlation between all pairs of data points is impossible. Selecting an appropriate context is a major challenge of computing the cross-correlation among the data points. The second problem associated with cross-correlation computation is the representation of cross-correlation. We will discuss both of them in the next two sections.

2.2.1. Context Selection

We select an appropriate context for the data points, among which we compute the cross-correlation. Then, we propose the selection of data points based on time. In the time-based approach, time is quantized and each quantum is separated by a Δt amount of time (the vertical lines in Figure 21). The data points that arrive within one quantum of time are grouped together. We compute the cross-correlation among the streams based on the data points arriving at the same quantum of time. We choose Δt as the smallest time difference between two consecutive data points from one stream because such choice of Δt ensures that there will be no two data points from the same stream within Δt.

Although this model is very restrictive, it is effective in tracking dynamically changing cross-correlations among the data points, and a suitable data structure can easily be established. Of course this approach would miss some cross-correlations among the data points with any time difference greater than Δt; however, if we want to capture those cross-correlations and compare the data points based on them, the processing of at least

one of the data points has to be delayed for a time duration greater than Δt as well. In this model, the processing of any data point is delayed no more than Δt.

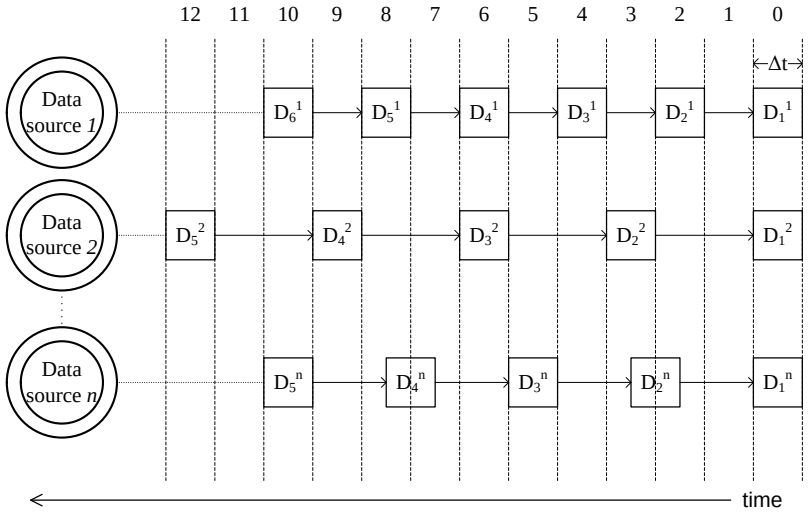

Figure 21. Temporal context for cross correlation

2.2.2. Cross-Correlation Representation

Wadjet captures the cross-correlations between the data points arriving within Δt from all streams. Since Δt is the minimum time between two consecutive data points from any stream, each stream can have at most one data point within Δ. Wadjet captures the cross-correlation between the data points by capturing the cross-correlation between their attributes. Each data point can have multiple attributes; hence Wadjet captures the pair-wise cross-correlation between all pair of attributes. Wadjet uses the Pearson correlation matrix [91] that has the pair-wise correlations of all attributes of all data points from all streams in order to compute the cross-correlations among the data points. As each data point consists of multiple attributes, computing the Pearson correlation between two data points is impossible since the Pearson correlation is defined between two random variables. Instead of computing the Pearson correlation

between two data points, Wadjet computes the Pearson correlation between a pair of attributes. This correlation matrix has $\frac{m\ x\ N}{2}$ entries, where m is the number of dimensions and N is the number of streams. Depending on the values of these two parameters, this matrix can be large; however, this matrix can be computed incrementally and in parallel.

Wadjet uses the West algorithm [92] to compute the covariance matrix, Σ, and computes the Pearson correlation matrix from the covariance matrix. The West algorithm computes the covariance matrix online. As a set of data points arrives, Wadjet updates the covariance matrix and the Pearson correlation matrix using Equation (11).

$$corr(X_1, X_2) = \frac{cov(X_1, X_2)}{\sqrt{Var(X_1)\ \times Var(X_2)}} \tag{11}$$

where X_1 and X_2 are two attributes, $cov(X_1, X_2)$ is the covariance between X_1 and X_2, $corr(X_1, X_2)$ is the Pearson correlation between X_1 and X_2 and $Var(X_1)$ and $Var(X_2)$ are the variances of X_1 and X_2, respectively.

2.3. Attribute Value Equating

Wadjet identifies the outliers based on the cross-correlations of attributes among the data points. Practically, not all attributes are correlated with each other. Therefore, Wadjet needs to find attributes that are correlated. Wadjet clusters the attributes based on Pearson correlation and equates the values of attributes within the cluster. Moreover, comparing two cross-correlated values is very difficult unless they are equal. Wadjet equates two attribute values using linear regression and later compares them to each

other to test whether they are different or not. In this section we discuss both of them in details.

2.3.1. Attribute Clustering

In order to find the correlated attributes, Wadjet assembles the attributes into clusters. For a multi-dimensional data point, each of its attributes are treated individually for clustering purposes. The attribute clustering is necessary because not all attributes are correlated to each other. The attribute clustering step groups the correlated set of attributes into one cluster. Later, Wadjet can compare the attribute values with each other inside a cluster to detect outliers. In the first step, Wadjet creates an empty cluster and chooses an attribute randomly and makes the attribute as the cluster head of a cluster; at this point this cluster has only one attribute, which is the cluster head, in it. In the second step, for each remaining attribute, Wadjet puts it into the same cluster created in the first step if it has a strong Pearson correlation with the cluster head. For those attributes that do not belong to that cluster, Wadjet repeats the two steps to form clusters for them until no attribute is left for clustering. In order to measure whether a Pearson correlation between two attributes is strong or not, Wadjet computes their coefficient of determination (often called χ^2). Given two random variables, the coefficient of determination is the square of Pearson correlation between them [89]. The coefficient of determination illustrates how much of the variance of a random variable can be explained using another random variable. So if two random variables, X_1 and X_2, have the coefficient of determination as 0.80, then 80% of the variance of X_1 can be explained using X_2. If X_1 and X_2 are perfectly correlated, then the coefficient of determination would be 1 and there would be a linear regression function ($X_1 = pX_2 +$

q where p and q are regression coefficients) between X_1 and X_2 [89]. If the coefficient of determination is smaller than 1, there still exists a linear regression between X_1 and X_2 but some data points would be deviated from the regression line. Wadjet calls two attributes strongly correlated if their coefficient of determination is not significantly different from 1 with a given confidence level.

2.3.2. Regression Function Computation

Once the cluster formation is complete, Wadjet equates each of the attribute values that belong to same cluster to the cluster head value. This is because all the attributes in a cluster are correlated but their values may not be equal. Comparing two unequal values to measure their similarity is impossible unless their values are equated to a common value. Wadjet uses a regression function to equate the attribute values to the cluster head in a cluster. By design, all attributes in a cluster are strongly correlated with the cluster head. If a cluster consists of three attributes, X_1, X_2 and X_3 where X_1 is the cluster head, there exist two linear regression functions between (X_1, X_2) and (X_1, X_3). The strong correlation between X_1 and X_2 ensures that there exists a linear regression function between X_1 and X_2; thus X_1 and X_2 can be represented as $X_1 = f_{X_1, X_2}(X_2) = p_{X_1,X_2}X_2 + q_{X_1,X_2}$ where p_{X_1,X_2}, q_{X_1,X_2} are called regression coefficients and $f_{X_1, X_2}(X_2)$ is the linear regression function that equates X_1 and X_2. Once we apply the linear regression function, each $f_{X_1, X_i}(X_i)$ becomes an independent observation of X_1 calculated from X_i. The values of the regression coefficients are given in Equations (12) and (13) [93].

$$p_{X_1,X_2} = \frac{Cov(X_1, X_2)}{Var(X_1)} \tag{12}$$

$$q_{X_1,X_2} = mean(X_2) - p_{X_1,X_2} * mean(X_1) \tag{13}$$

where $mean(X_2)$ and $mean(X_1)$ represent the mean values of X_2 and X_1, respectively. The variance and covariance values can be readily found from the covariance matrix Σ. Wadjet computes the regression coefficients p_{X_h,X_i} and q_{X_h,X_i} for the regression function between each attribute, X_i, of a cluster and its cluster head X_h.

2.4. Outlier Detection

Once Wadjet equates all the attribute values in each cluster to the cluster head value, all the attribute values including the cluster head value are expected to be equal. The set of values obtained after equating the attribute values of a cluster to their cluster head is called equivalent values. Consider a cluster consisting of four attributes X_1, X_2, X_3, and X_4 with the attribute values x_1, x_2, x_3, and x_4 , respectively, and X_1 being the cluster head. After equating the attribute values of X_2, X_3 and X_4, Wadjet would have four different values $f_{X_1,X_2}(x_2)$, $f_{X_1,X_3}(x_3)$, $f_{X_1,X_4}(x_4)$ and x_1 in that cluster; these values are called equivalent values. These are four independent observations of x_1 obtained from X_2, X_3, X_4 and X_1. Hence, each cluster of attributes has a similar set of equivalent values. By design, equivalent values in a cluster are supposed to be equal; so if one of them is significantly different from the rest of the equivalent values, then that equivalent value is most likely an outlier; and therefore, the originating data point containing that attribute value is called an outlier. In order to detect whether or not an equivalent value is significantly different from other equivalent values in the same cluster, Wadjet uses a

statistical significance test. Theoretically, once the regression function is found, for X_1, X_2, X_3 and X_4 after equating, we can expect $x_1 = f_{X_1, X_2}(x_2) = f_{X_1, X_3}(x_3) = f_{X_1, X_4}(x_4)$. If $x_1 = f_{X_1, X_2}(x_2) = f_{X_1, X_3}(x_3) \neq f_{X_1, X_4}(x_4)$ and $f_{X_1, X_4}(x_4)$ is significantly different from $x_1, f_{X_1, X_2}(x_2)$, and $f_{X_1, X_3}(x_3)$ then we call x_4 an outlying value for the attribute X_4 and if $x_4 \in D_t^2$, then data point D_t^2 is called an outlier.

In order to detect whether an equivalent value is significantly different from the other equivalent values in the same cluster, Wadjet computes the nearest neighbor for each equivalent value in that cluster. According to Lemma 4, if an equivalent value is significantly different from its nearest neighbor (which is also an equivalent value), it would be significantly different from the rest of the equivalent values; thus it is sufficient to prove that a value is significantly different from its nearest neighbor.

Lemma 4. If an equivalent attribute value is significantly different from its nearest neighbor, it is significantly different from the rest of the equivalent values.

Proof. Let a cluster be composed of three attribute values x_1, x_2 and x_3 with x_2 being the nearest neighbor of x_1. Here we prove that if x_1 is significantly different from x_2, then it is also significantly different from x_3. x_1 and x_2 would be significantly different if $\sqrt{df} * \frac{|x_1 - x_2|}{\sigma_{X_1}} > t_{\alpha,df}$, where σ_{X_1} is the estimated standard deviation of attribute X_1, df the degree of freedom (number of parameters that can be independently varied), $t_{\alpha,df} = \frac{1}{2} + \alpha\Gamma(\frac{df+1}{2})$ the critical value obtained from the student-t distribution where $\alpha = 1 - \frac{confidenceInterval}{100}$, $\Gamma(.)$ is the Gamma function; x_1 would be significantly different from x_3 if $\sqrt{df} * \frac{|x_1 - x_3|}{\sigma_{X_1}} > t_{\alpha,df}$. Since both x_2 and x_3 are obtained from a

linear regression of the same order, their degree of freedom is equal. With x_2 being the nearest neighbor of x_1, we have $|x_1 - x_2| < |x_1 - x_3|$, hence $\sqrt{df} * \frac{|x_1 - x_3|}{\sigma_{x_1}} > \sqrt{df} * \frac{|x_1 - x_2|}{\sigma_{x_1}} > t_{\alpha,df}$, therefore x_1 is significantly different from x_3. Hence, if an equivalent attribute is significantly different from its nearest neighbor, it is also significantly different from the rest of the equivalent values.

A data point is identified as an outlier if it is identified as an outlier in the first phase or in the second phase of Wadjet. This is because the data point is nonconformist to either other data points from the same stream carrying the data point or other data points from the multiple streams arriving at the same time point.

If any cluster contains fewer than three attributes, then Wadjet skips that cluster. This is because if a cluster has two attributes and they are significantly different from one another, it is impossible to identify which one of them is an outlier and which one of them is not. In that case the correlation between any of these two attributes with other attributes is not strong enough for one attribute to be compared with the others to detect the outlier-ness of that attribute. Thus Wadjet cannot identify the outlier-ness of those attributes based on the cross-correlation, but the outlier-ness of the associated data point can still be identified using Orion.

2.5. The Wadjet Algorithm

Figure 22 shows our algorithm Wadjet. The algorithm is invoked once a set of data points S_T arrive at any point in time *T*. Wadjet then executes Orion for each data point

D_T^i in S_T (lines 4-11). If a data point is identified as an outlier by Orion, Wadjet adds it to the *outliers* set (line 7).

```
detectOutlier(Set of Data Points S_T, neighbor distance r,
percentage of neighbors k, cross-correlation matrix Σ_{T-1})
      1  notOutlier ← nil
      2  outliers ← nil
      3  attributes ← nil
      4  for each D_t^i in S_t // D_t^i is t-th data point from i-th
         stream
      5    isOutlier ← ProcessData(D_t^i, r, k)// execute Orion
      6    if (isOutlier)
      7      outliers.add(D_T^i)
      8    else
      9      notOutlier.add(D_T^i)
     10      attributes.addAll(D_T^i)
     11  end for
     12  Σ_T ← updatePearsonCorrelation(attributes, Σ_{T-1})
     13  clusters ← nil // start with empty set of clusters
     14  do
     15    cluster ← ϵ // begin with empty cluster
     16    cluster.head ← random(attributes) // choose
         cluster head
     17    attributes.remove(cluster.head)
     18    for each attr in attributes
     19      if (attr is correlated with cluster.head)
     20        attributes.remove(attr)
     21    end for
     22    clusters.add(cluster)
     23  until (attributes is empty)
     24  for each cluster in clusters
     25    for each attr in cluster equate (attr,
         cluster.head)
           for each attr in cluster
     26  findNearestNeighbor(attr) // find the nearest
         neighbor
     27    if (isDifferent(attr, nearestNeighbor(attr)))
     28      D_t^i ← findDataPoint (attr)
     29      if(!outliers.contains(D_t^i)) outliers.add(D_t^i)
     30  end for
     31  return outliers
end procedure
```

Figure 22. The Wadjet Algorithm

Once Wadjet completes executing Orion for each data point, Wadjet collects all the attributes of the data points that are not outliers (line 10). In the next step Wadjet updates the Pearson correlation matrix in line 12 and clusters the attributes based on their coefficient of determination (lines 19-21). In the next step for each cluster, Wadjet equates all the attribute values in the cluster and finds the nearest neighbor of each attribute value. If an attribute value is significantly different from its nearest neighbor then Wadjet finds the associated data point with that attribute (lines 27-29). If the data point is not already identified as an outlier, it is added to the set of outlier, *outliers*. Finally Wadjet returns the set of outliers in line 31.

2.6. Confidence Interval Parameter Selection

Wadjet needs one additional parameter, called confidence interval, besides those required by Orion. The confidence interval parameter is used for the significance test when Wadjet decides whether an attribute is significantly different from its nearest neighbor or whether an attribute is significantly correlated with a cluster head or not. In this section we discuss the value selection for this parameter.

Confidence interval is the measure of reliability of the comparison. It is a user defined parameter. Wadjet compares one attribute value to its nearest neighbor attribute value. If the attribute value is significantly different from its nearest neighbor, Wadjet identifies the associated data point as an outlier. If the confidence interval is large the two attribute values has to be very far from one another. If the confidence interval is small, a small distance between two attribute values can make the attribute values different from one another. 95% confidence interval is very popular in literature and is

considered reliable enough for many applications [91]. In case of Wadjet, if the confidence interval is high, in order to detect a data point as an outlier, the attribute value of that data point has to be significantly different from those of the other data points and Wadjet would only consider the attributes that are highly correlated. Thus for a high confidence interval, it is very unlikely that Wadjet would create any false alarm, thus the precision of Wadjet should be high. However, the high precision comes with the cost of low recall. In that case Wadjet may miss some outliers that are not too far from their related attributes. Confidence interval can be seen as a tradeoff between precision and recall for Wadjet.

Since Wadjet is a two-phase outlier detection technique, we recommend using a high confidence interval. This is because a data point is identified as an outlier if it is identified as an outlier in any of the two phases. Hence, we want to minimize the false alarms (maximize precision) in both phases so that the total number of false alarms can be minimized.

CHAPTER IV
PERFORMANCE ANALYSIS

This chapter presents the theoretical and empirical analysis of our techniques, Orion and Wadjet, evaluating their performance in terms of accuracy and execution time. The empirical analysis is conducted using both real and synthetic datasets. We present the theoretical analysis first followed by the details of our simulation model and experimental results.

1. Theoretical Analysis

In this section we discuss the time and space complexity of Orion and Wadjet.

1.1. Complexity Analysis for Orion

1.1.1. Time Complexity of Orion

The complexity of Orion is divided into two parts: 1) outlier detection and 2) update. The outlier detection part includes everything starting from an arrival of a data point to the final decision about the outlier-ness of that data point (which includes appropriate p-dimension finding, outlier metrics computation and co-clustering to classify the data point). In this section, we present the time complexity of each part individually and the total complexity of Orion, which is the sum of the complexity of all three parts. The complexity of Orion is analyzed based on the amount of time Orion would take to execute for one data point with respect to the number of dimensions and number of bins for each dimension. Since Orion is executed for every data point and the number of data

is potentially infinite, we do not analyze Orion with respect to the number of data points.

1.1.1.1. Time Complexity of the Outlier Detection Part

On the arrival of a new data point D_t, Orion chooses the set of p-dimensions along which the neighbor density and k-distance of D_t would be computed, which is in either A_{in} or A_{out}. Orion computes the maximum absolute normalized deviation ($\max\{AD_{Z_a}(D_t)\}$ in Section 1.3.2) of D_t; the maximum absolute normalized deviation of D_t later is compared with the average maximum absolute normalized deviation of all history data points. The maximum absolute normalized deviation requires inverting the covariance matrix. If the number of data dimensions is m, the time complexity of the matrix inversion is $O(m^3)$ using a simple matrix inversion algorithm. There exists a complex algorithm that reduces the time complexity to $O(m^{2.373})$ [94]. We use the simple matrix inversion algorithm because of its ease of implementation and for the purpose of complexity analysis here. Thus Orion needs $O(m^3)$ to compute the maximum deviation and choose between A_{in} and A_{out}. Once A_{in} or A_{out} is chosen, Orion finds the appropriate p-dimensions in it. Finding the appropriate p-dimension requires neighbor density computation for every existing p-dimension. Hence, we discuss the time complexity of neighbor density computation first, followed by the time complexity of finding appropriate p-dimensions.

In order to identify whether D_t is an outlier, Orion computes its neighbor density of for each existing p-dimension in A_T. The neighbor density is computed from the binned implementation of the data density function where each bin stores the cumulative data

density function $F_{Z_a,T}(Z_a)$. Orion needs to find the bordering bins and computes the neighbor density using Equation (14)

$$\begin{aligned} nDen_{Z_a}(D_t, r_{Z_a}) &= \int_{D_t.v[a]-r_{Z_a}}^{D_t.v[a]+r_{Z_a}} f_{Z_a,T}(z_a)dz_a \\ &= F_{Z_a,T}(D_t.v[a] + r_{Z_a}) - F_{Z_a,T}(D_t.v[a] - r_{Z_a}) \end{aligned} \quad (14)$$

where $f_{Z_a,T}(z_a)$ is the data density function, $F_{Z_a,T}$ is the cumulative data density function, and r_{Z_a} is the neighbor distance along p-dimension Z_a at time T. Since the neighbor density computation requires to lookup two cumulative data density functions and subtracts one from another, the computation complexity of computing the neighbor density is $O(1)$. However, we have to compute the neighbor density for m p-dimensions (here we are assuming the number of p-dimensions is the same as the number of data dimensions because in our experiments, we choose the number of population to be the same as the number of data dimensions). Hence, in order to find the p-dimension with the smallest neighbor density we have to iterate over all p-dimensions once. Thus the total time complexity of finding the p-dimension with the smallest neighbor density is $O(m)$. Wadjet needs to find the minimum neighbor densities obtained from m p-dimensions which Wadjet finds along with the computation of neighbor density with the same $O(m)$ time complexity.

Once Orion finds the p-dimension, it computes the k-distance along that dimension. The k-distance is a monotonically increasing function of k because if k increases, the distance that includes k percent of the data points also increases. Hence Orion uses binary search to find the appropriate k-distance for a given k. If $binCount$ is the total

number of bins along a p-dimension, the time complexity of k-distance computation is $O(log(binCount))$.

Finally, when both the neighbor density (ND) and k-distance of a data point are obtained, they are used to classify D_t. The classification task consists of computing the distances from the data point represented by ND and k-distance to the cluster centers of the co-clusters (lines 3-5, Figure 16, Chapter 3, Section 1.6.2). Since we have a constant nine clusters, the classification task takes $O(1)$ time. Thus the total time complexity of detecting the outlier-ness of a data point is $O(m^3 + log(binCount))$.

1.1.1.2. Time Complexity of the Update Part

In the update algorithm, Orion updates the set of p-dimensions and their fitness, creates a new p-dimension and discards an old p-dimension. Once Orion computes the neighbor density for all p-dimensions, Orion sorts them in ascending order. If we have $\frac{m}{2}$ p-dimensions in each of A_{in} and A_{out} the time complexity of sorting them is $O(mlog(m))$. Once the p-dimensions are sorted, Orion updates the fitness of each p-dimension. Updating the fitness of a dimension takes a constant time because it just updates a fixed number of values which is the mean fitness. Hence, the total time to update the fitness values of all p-dimensions is $O(m)$.

In the next step, Orion discards an old dimension and chooses two well fit p-dimensions and creates a new p-dimension. Finding the poorly performing p-dimensions or well performing p-dimension uses the roulette wheel algorithm [79]. The roulette wheel algorithm picks a p-dimension randomly where the p-dimension with the highest fitness has the highest probability of being chosen. The time complexity of the roulette wheel

algorithm is $O(m)$ because it evaluates the fitness of each p-dimension sequentially. Thus, our p-dimension calculation algorithm takes $O(m)$ time where m is the number of p-dimensions. This is because we track the fitness value of each dimension in A_{in} and A_{out}. The new p-dimension calculation also requires $O(m)$ time because Orion needs to calculate m coefficients for m data dimensions.

Orion updates the data density function for all p-dimensions in A_{in} and A_{out}. In order to update one p-dimension Orion has to traverse all the bins of that dimension. Hence updating one p-dimension takes $O(binCount)$ amount of time, where $binCount$ is the number of bins for a p-dimension. Thus the total time to update all bins for all p-dimensions is $O(m \times binCount)$. Totally, the time complexity of the update part is $O(m(log(m) + binCount))$.

Considering both the outlier detection and update parts of the Orion algorithm as shown in Figure 16 in Chapter 3, the total time complexity of Orion to identify whether a data point is an outlier is the sum of the complexity of those two parts, which is $O(m^3 + log(binCount) + m(log(m) + binCount))$

1.1.2. Space Complexity of Orion

Orion stores m p-dimensions. For each dimension Orion stores its number of bins, which is represented by $binCount$. For each bin Orion stores 5 different variables: last update timestamp, data density function, derivative of the data density function, cumulative data density function, and bin's representative value. For each p-dimension Orion also stores its fitness value. Hence the total amount of storage required is $5 \times m \times (binCount + 1)$. Moreover, Orion requires a constant amount of space for co-

clustering that stores a constant number of variables (the 9 cluster centers). Thus the total space complexity is $O(5m \times (binCount + 1) + 9)$, thus the space complexity is $O(m \times binCount)$ (disregarding the lower order terms and constants).

1.2. Complexity Analysis for Wadjet

In this section we present the time and storage complexity of Wadjet. Since Wadjet includes Orion as the first step, we omit the complexity discussion of Orion and only discuss how we derive the time and storage complexity of the second phase of Wadjet. Then, we show the total time and storage complexity of Wadjet that includes both phases. The time complexity of Wadjet is analyzed based on the time the algorithm would take to execute once with respect to the number of dimensions and streams and the number of bins for each dimension.

1.2.1. Time Complexity of Wadjet

In the second phase of Wadjet, to detect whether a data point is an outlier, Wadjet updates the cross-correlation matrix. The cross-correlation matrix contains the Pearson correlation between all pair of attributes. In case of homogeneous data streams, if we have n streams with m dimensions for each stream, then the total number of attributes would be mn and the matrix has m^2n^2 entries. For heterogeneous data streams, if we n streams with m_1, m_2, …, m_n dimensions, the total number of attributes would be $m_1 + m_2 + \cdots + m_n$. Without losing any generality we can assume the average number of dimensions for each streams in case of heterogeneous streams is $m = \frac{m_1+m_2+\cdots+m_n}{n}$; hence the total number of attributes is mn. Thus in both cases the total number of attributes is mn where n is the number of streams and m is the average number of

dimensions for each stream. The cross-correlation matrix requires m^2n^2 entries for pair-wise correlation computation. In the worst case each entry of the matrix needs to be updated. Hence the time complexity of updating the cross-correlation matrix is $O(m^2n^2)$. After updating the cross-correlation matrix, Wadjet clusters the attributes into clusters. Our clustering algorithm finds the set of cross-correlated attributes for each attribute. Hence in the worst case, Wadjet needs to check m^2n^2 cross-correlations. So, the time complexity of the clustering algorithm is also $O(m^2n^2)$.

Once the clusters have been formed, Wadjet computes the regression function and the equivalent value for each attribute in the cluster, which takes a constant time e; hence in the worst case the total number of regression and equivalent values we need to compute is mn. Thus the time complexity for computing the equivalent values for all attributes is $O(mn)$. In the last step, Wadjet computes the nearest neighbor for each attribute's equivalent value which, in the worst case, takes all pair-wise distance computations with a time complexity of $O(m^2n^2)$.

In the following part Wadjet tests whether an attribute value is significantly different from the other attribute values in the cluster. The test requires constant time complexity because it involves critical value lookup and computation of the difference between one attribute value and its nearest neighbor. Moreover, since the previous step is performed for each attribute, the total number of tests performed is mn. Once an attribute is identified as an outlier Wadjet finds the corresponding data point and marks it as an outlier as well. The last step requires a constant time because finding a data point given one attribute is a trivial lookup procedure. Therefore, the total time complexity for the second step of Wadjet is $O(m^2n^2 + mn)$ which is $O(m^2n^2)$.

In the first phase, Wadjet executes Orion for each data point from a stream. Thus the total time complexity of Wadjet is n times the time complexity of Orion where n is the number of streams. The time complexity of the first phase of Wadjet is $O(n(m^3 + log(binCount) + m(log(m) + binCount)))$. Thus, the total time complexity of Wadjet becomes $O(m^3n + nlog(binCount) + mnlog(m) + mnbinCount + m^2n^2)$. Disregarding the lower order terms, the time complexity of Wadjet becomes $O(mn(m^2 + mn + binCount))$.

1.2.2. Space complexity of Wadjet

The space complexity of the second step of Wadjet is very straightforward. The only thing we store is the cross-correlation matrix. The cross-correlation matrix captures the cross-correlation between all pairs of attributes. For n data streams with average m attributes each, the total number of attributes is m. Thus the total number of pairs possible is m^2n^2. Thus the cross-correlation matrix needs to have m^2n^2 entries. However, correlation is symmetric relation, means the correlation between x and y is the same as the correlation between y and x. Thus the cross-correlation matrix needs to have half of m^2n^2 entries. So, the number of entries in this matrix is $\frac{m^2n^2}{2}$. Thus the space complexity for the second step is $O(m^2n^2)$. Including the first step of Wadjet which runs Orion for each stream, the total space complexity of Wadjet is $O(mn(mn + binCount))$.

2. Experimental Analysis

We have conducted an extensive set of simulation experiments to study the performance of our techniques and compare it with that of existing techniques. In this section, we

present the simulation model and the experimental results that we have obtained. We have performed empirical studies for both of our algorithms. We divide our experimental analysis section into two parts (1) experimental analysis for Orion and (2) experimental analysis for Wadjet.

2.1. Experimental Analysis for Orion

2.1.1. Simulation Model

The goal of the simulation model is to demonstrate the effectiveness of Orion. The details of our simulation model are discussed in this section.

2.1.1.1. Software Description

In the simulation model, we mimic a centralized data stream architecture where there is one base station and a number of data sources each of which produces one data stream. . Each data source obtains a data value at a fixed time interval and sends it to the base station. The base station receives one data point at a time and processes it. We execute one outlier detection algorithm at a time at the virtual base station for outlier detection.

The entire simulation model is built on the Java platform (Sun Developer Network 2010) and we run the simulation using JAVA version 1.6.2. JAVA is running on Red Hat Linux Enterprise 5 (OU Supercomputer Resources 2009). The base station processes each data source in an individual thread/core.

2.1.1.2. Hardware Description

We use the cluster supercomputer at the University of Oklahoma to run our simulation model [95]. Each computing core is a 2.0 GHz Pentium4 Xeon E5405/2.66GHz Pentium4 Xeon E5345/2.40 Pentium4 XeonMP E7340 [95]. Each computing node has

16GB of main memory and eight computing cores. The comparison is fair since each technique is run on the same machine.

2.1.1.3. Datasets

We perform our experiments based on both real and synthetic datasets. Our real datasets are collected from UCI Machine learning repository [96]. In this section we will describe each of our datasets.

2.1.1.3.1. KDD Cup 99 Data

Network security is becoming very important. Outlier detection is a popular way for detecting network intrusion. We use KDD'99 data in order to show the efficacy of our algorithm. This dataset is captured in the DARPA'98 IDS evaluation program [97]. The dataset consists of 4 gigabytes of compressed TPC dump data of 7 weeks of network traffic (approximately 5 million connection records of 100 bytes each). A single connection record contains 41 features and is labeled as either normal or an attack. In our experiments, we mark the data points that are labeled as normal as regular data points and those that are marked as attack as outliers.

2.1.1.3.2. Vicon Physical Action Data

The data collection took place in the Essex robotic arena [96]. Seven male and three female subjects were involved in a scenario such as physical fighting in 20 different experiments. Throughout the 20 experiments each subject has to perform 10 normal and 10 aggressive activities in random locations. Human subjects perform normal or attacking activities and the locations of the different body parts are measured at a regular interval. Each data point has a timestamp attached with it and the data points are

temporally correlated. Each data point has 27 attributes. Each physical activity data has been recorded separately. The data points from normal actions are marked as normal and the data points from aggressive activities are marked as outliers. In our experiments we try to find the data points that correspond to aggressive activities.

2.1.1.3.3. Australian Sign Language Data

This dataset consists of a sample of signs of Australian Sign Language [96]. The dataset is the raw measurement from a Nintendo PowerGlove which is interfaced to a Silicon Graphics 4D/35G workstation with PowerGlove Serial Interface. The position information is calculated on the basis of ultra sound emission from the emitters to the microphone. There are two emitters in a glove and three microphone receivers. Total four pieces of information is collected: (1) left/right, (2) up/down, (3) backward/forward and (4) roll of palm. All the positions are calibrated with respect to a fixed calibration point. The measurements from five individual participants have been collected. Since outlier detection is very similar to the task of classification with skewed distribution of the classes, we measure the performance of our technique from the classification task. The dataset consists of a set of signs represented as time series of aforementioned four pieces of data. We choose one sign at random (the data point representing that sign is called inliers) and some data points from a second sign are mixed as outliers. Our goal here is to use an outlier detection technique to separate the data points into two classes.

2.1.1.3.4. EMG Physical Action Data

The EMG Physical Action dataset is similar to the Vicon Physical Action dataset in many respects except that its data points were collected from eight different locations of

a human body unlike the Vicon Physical Action dataset where data is collected from twenty-seven different locations of a human body [96].

2.1.1.3.5. Irrigation Data

The California Irrigation Management Information System (CIMIS) manages a network of over 120 automated weather stations in California [15]. Each weather station collects data every minute and calculates hourly and daily values. The data are analyzed and stored in the CIMIS database and publicly available. The measured attributes are solar radiation, net radiation, air temperature, vapor pressure, wind speed, result wind, precipitation, relative humidity, dew point, and soil temperature. For our experiments, we use the data collected from 1998 to 2011 and implant the random synthesized outliers. On average each station has 50,000 rounds of data (total 5,550,000 data rounds for all stations together). From now on we shall refer to this dataset as *irrigation data*.

2.1.1.3.6. Synthetic Data

We create another dataset synthetically in order to perform a vast array of experiments. Our synthetic dataset resembles real life data. Each data source has a fixed 100 dimensions unless otherwise specified; and some of the dimensions (randomly selected) are linearly correlated. Each dimension has three components in it: (1) trend component, (2) harmonic component and (3) noise component. The trend component changes the trend of the dimension over a long period of time. The harmonic component adds periodicity to the attribute value. Finally the noise component adds random or Gaussian noise to the attribute value. We generate 50 data sources (50,000 data rounds for each source), all of which have the same number of dimensions. We synthetically implant the

outliers by changing a set of attribute values by adding/subtracting a set of random numbers to/from them.

2.1.1.4. Competitive Algorithms

We compare Orion with two existing algorithms. The first algorithm is called Stream LOCI [62]. LOCI was first proposed by Papadimitriou et al. [59]. LOCI computes the deviation of neighbor count, called Multi granularity Deviation Factor (MDEF), of every data point from the average neighbor count of data points which are within a certain radius of the data point. LOCI calculates the MDEF of a data point for multiple radiuses. A data point is identified as an outlier if its MDEF value is three standard deviations apart from its mean. Lu et al. [62] extends the idea for data streams. They assume a sliding window of user defined size *w* and execute the LOCI algorithm for the sliding window. They also optimize the process of inserting a data point into the sliding window and deleting a data point from the sliding window as well. We name the technique Stream LOCI. Stream LOCI works for multi-dimensional data points, but measures the similarity between data points using Euclidian distance.

The second algorithm is A-ODDS [60] which is designed for single dimensional data streams. Still, we can see a multi-dimensional data stream as a collection of some single dimensional data streams and finds the outliers for each dimension individually. In case of multi-dimensional data streams we execute one A-ODDS for each dimension and detect outliers. A data point is identified as an outlier if any of its dimensions is identified as an outlier by A-ODDS. For A-ODDS and Orion, we use the first 100 data points for bootstrapping purposes for all experiments unless otherwise specified.

2.1.1.5. Simulation Parameters

We study the impacts of Orion's parameters on its performance. The range of values and the default value of each parameter is presented in Table 3. The default values are chosen based on the characteristics of the datasets (for the number of rounds and number of dimensions parameters) and existing literature (for the neighbor distance, k, percentage of outliers, and bin count parameters). The default value of the population count is same as the number of data dimensions. For every experiment, when the impact of a parameter is under study, we vary its value within its range and fix the other parameters at their default values.

We only perform the parameter study for two datasets, irrigation data and synthetic data. This is because we cannot vary the parameters such as number of dimensions, percentage of outliers, etc. for other datasets. Thus we choose the irrigation data and synthetic data as the representative datasets and perform all of parameters studies based on them. However, we report the overall performance for all datasets.

Table 3. List of parameters for Orion

Name	Irrigation data		Synthetic data		KDD' 99	
	Range of Values	**Default value**	**Range of Values**	**Default Value**	**Range of Values**	**Default value**
Neighbor distance	0 – 50	10	0 – 50	10	0 – 50	10
k	0 – 0.25	0.05	0 - 0.25	0.05	0 – 0.25	0.05
Percentage of Outliers	1 -10	5	1- 10	5	N/A	N/A
Number of data dimensions	N/A	N/A	10 – 100	100	46	46
Population count	5 – 50	10	5 – 50	25	46	46
Bin count	50 -500	400	50 -500	400	400	400
Bootstrap size	50 – 500	100	100 – 1,000	100	100	100
Data rounds	5,000 – 40,000	40,000	5,000 – 50,000	50,000	400,000	400,000
	Vicon Physical Action		**EMG Physical Action**		**Australian sign language**	
Neighbor distance	0 – 50	10	0 – 50	10	0 – 50	10
k	0 – 0.25	0.05	0 - 0.25	0.05	0 – 0.25	0.05
Percentage of Outliers	5%	5%	5%	5%	5%	5%
Number of data dimensions	27	27	8	8	22	22
Population count	27	27	8	8	22	22
Bin count	400	400	400	400	400	400
Bootstrap size	100	100	100	100	100	100
Data rounds	4,000	4,000	10,000	10,000	1,500	1,500

2.1.1.6. Performance Metrics

Table 4. Confusion matrix

	Actual outliers	**Actual inliers**
Predicted outliers	True positive (TP)	False positive (FP)
Predicted inliers	False negative (FN)	True negative (TN)

We present the accuracy of each studied algorithm based on the four following performance metrics: Precision, Recall, Jaccard Coefficient and Receiver Operator Characteristic Curve. All the performance metrics are based on the confusion matrix shown in Table 4 where a true positive (TP) is a real outlier that is identified as an outlier, a true negative (TN) is an inlier that is identified as an inlier, a false positive (FP) is an inlier that is identified as an outlier, and a false negative (FN) is an outlier that is identified as an inlier. A good outlier detection technique is the one which maximizes the true positives and minimizes the false negatives and false positives. In case of outlier detection we ignore true negative; this is because outliers are significantly outnumbered by inliers. Thus, the number of true negatives is very high compared to the number of true positives. Incorporation of the number of true negatives would deteriorate the quality of the results. Consider the case where we have a dataset with 95% inliers and 5% outliers and an algorithm that identifies everything as an inlier. If we consider the accuracy based on both true negatives and positives, the accuracy of that algorithm would be 95%; but in reality, the algorithm is useless. Hence, we ignore true negatives from our results.

Precision

Precision ($precision = \frac{TP}{TP+FP}$) is a popular performance metric for outlier detection [31, 36, 64]. Precision is a ratio of the correct identifications to the total identifications.

Statistically, it implies the amount of correct identifications i.e., precision resembles the correctness of the classifying task. Intuitively, an optimal classifier is the one which has the highest precision. The highest possible precision is 1 where there is no false classification [5].

Recall

Precision only shows the correctness of the results but it does not reflect the completeness of the results; therefore precision is always accompanied with recall ($recall = \frac{TP}{TP+FN}$) to demonstrate the correctness and completeness of an algorithm. Recall is the ratio between the number of identified outliers and the total number of outliers. Statistically, it represents the completeness of the classification task. An optimal algorithm should be able to identify all the outliers existing in a dataset; hence the optimal recall is 1. The best algorithm should have the highest precision and recall, but practically, many algorithms do a trade-off between precision and recall, i.e., if precision increases, recall decreases and vice versa.

Jaccard Coefficient (JC)

Precision and recall represent two important concepts, but it is often possible to maximize one value by minimizing the other. Hence, it is very difficult to evaluate the performance of an algorithm based on two different metrics. Basu and Meckesheimer [13] proposed the Jaccard Coefficient (JC) to overcome the shortcomings. JC is the ratio of true positives and the summation of false negatives, false positives and true positives ($JC = \frac{TP}{FP+FN+TP}$). Therefore, JC increases with the increase of correct positive classifications and decreases with the increase of wrong classifications. JC does not

consider the true negatives which are not necessary for the domain of outlier detection because the number of inliers is significantly larger than the number of outliers; hence any performance metric that considers only true negatives fails to depict the differences between the competitive algorithms vividly. Jaccard Coefficient is a more appropriate metric to evaluate the performance of outlier detection algorithms.

Execution time

The study of execution time of an outlier detection algorithm is important for data stream applications. Data stream outlier detection algorithms have to be online and the processing of a data point must be finished before the next data point arrives. Thus if the execution time of an algorithm is greater than the time difference between two consecutive data points in a data stream application, the outlier detection algorithm would not be applicable for that application. This is because if an algorithm cannot finish processing a data point before the next one arrives, the result would be flooding. Thus, execution time is one important metric that defines the applicability of an outlier detection algorithm for data streams. Execution time is reported in milliseconds in all our experiments.

2.1.2. Experimental Results

2.1.2.1. Overall Performance Comparison

The overall performance comparison is done by fixing all the parameters' values at their respective default values. The outliers are detected for individual datasets by each algorithm for each stream. The interpretation of one stream is dataset specific: a stream is either a station in the irrigation data, a physical action in the Vicon and EMG data, a sign in the Australian sign language data, or a source of the synthetic data . The

KDD'99 data consists of one stream only. The average performance results are presented in this section. They are computed from all the performance results collected from each stream.

2.1.2.1.1. Precision

Table 5. Precisions of all three algorithms for all datasets

	Orion	Stream LOCI	A-ODDS
KDD'99	0.69	0.64	0.26
Vicon Physical Action	1.00	0.001	0.44
Australian Sign	0.69	0.38	0.22
EMG Physical Action	1.00	0.06	0.78
Irrigation data	0.76	0.73	0.14
Synthetic data	0.99	0.26	0.37

Table 5 shows the average precisions for all datasets for the three studied algorithms. Irrespective of the datasets, Orion always performs better than the other two competing algorithms. Orion shows near perfect precision for the Vicon Physical Action, EMG Physical Action and Synthetic datasets. Data points in these three datasets are strongly correlated and less sparse compared to the data points in the other three datasets. Therefore, no (or almost no) inlier has a significantly small neighbor density or large k-distance and thus is never misidentified as an outlier. But in the KDD'99, Australian sign and Irrigation datasets, the data points are relatively sparse compared to the other three datasets and some inliers have quite small neighbor density and/or large k-distance and are misclassified as outliers.

One interesting thing to point out here is that Stream LOCI completely fails to detect outliers for the Vicon and EMG physical action datasets. Although the data points in these two datasets are not too sparse, Stream LOCI is unable to measure the similarity

among the data points using Euclidian distance for these two datasets and, therefore, fails to identify outliers correctly.

A-ODDS only considers each dimension individually; therefore, it completely fails to incorporate the correlation among the dimensions. If one attribute value of an outlier is different enough to distinguish itself from the rest of the data points, then it can easily be detected as an outlier. A-ODDS shows better precision on the EMG physical action; this is because in this dataset, for some outliers, a data point can be detected as an outlier just by examining one attribute value only.

2.1.2.1.2. Recall

Table 6 shows the recall values of Orion, Stream LOCI and A-ODDS for all six datasets. Orion has better recall value compared to Stream LOCI for all six datasets. A-ODDS shows better recall value compared to Stream LOCI in two datasets: Vicon Physical Action and Synthetic datasets.

Orion starts with an initial set of p-dimensions that may not incur the minimum neighbor density for the current data point. Gradually Orion adds new p-dimensions and removes old p-dimensions so that the current set of p-dimensions incur the minimum neighbor density for the current data point. If Orion fails to find an appropriate p-dimension for a data point that has the minimum neighbor density, the data point may not be identified as an outlier. In the Australian sign language dataset, the data points are not only sparse, but also changing very fast. Hence Orion sometimes fails to find the appropriate p-dimension and therefore fails to identify some outliers, thus, shows a smaller recall value compared to other datasets.

Stream LOCI performs comparatively poor in terms of recall for all six datasets. Since all of these datasets possess multi-dimensional data, Euclidian distance fails to identify the dissimilarity between inliers and outliers and, hence, plenty of outliers are identified as inliers, which results in a poor recall. A-ODDS shows strong recall values for all datasets except KDD'99. This is because the outliers in the KDD'99 datasets cannot be detected just by looking at one single attribute, rather it is the combination of attribute values that makes a data point an outlier. Hence A-ODDS shows a poor recall value for this dataset. In other datasets, the outliers have either an infrequent attribute value or an infrequent combination of attribute values. A-ODDS detects outliers in the former case, but fails to identify any outlier in latter case.

Table 6. Recalls of all three algorithms for all datasets

	Orion	Stream LOCI	A-ODDS
KDD'99	0.92	0.34	0.16
Vicon Physical Action	1.00	0.003	0.98
Australian Sign	0.67	0.34	0.66
EMG Physical Action	0.93	0.02	0.88
Irrigation data	0.70	0.33	0.84
Synthetic data	0.86	0.10	0.96

2.1.2.1.3. Jaccard Coefficient

Table 7 shows the average Jaccard Coefficient (JC) of all the algorithms for all the datasets. Orion shows better JC compared to Stream LOCI and A-ODDS for all six datasets. Orion has perfect JC for the Vicon Physical Action dataset, i.e., Orion identifies all the outliers without any false alarm at all. The JC of Orion for the Australian sign and irrigation datasets is comparatively small compared to the JC of other four datasets due to a smaller recall of Orion for these two datasets.

Stream LOCI performs poorly compared to Orion in all datasets. Stream LOCI shows poor JC for spatial data such as the Vicon and EMG Physical Action data. In case of spatial data, Stream LOCI fails to measure the similarity between two data points using the Euclidian distance, hence shows poor accuracy overall. A-ODDS also shows a smaller JC compared to Orion due to its inapplicability to multi-dimensional datasets. A-ODDS only performs well for the EMG Physical Action dataset where an outlier can be detected just by looking at one attribute value.

Table 7. Jaccard Coefficients for all three algorithms for all datasets

	Orion	Stream LOCI	A-ODDS
KDD'99	0.65	0.28	0.12
Vicon Physical Action	1.00	0.001	0.43
Australian Sign	0.51	0.21	0.20
EMG Physical Action	0.93	0.01	0.70
Irrigation data	0.55	0.30	0.13
Synthetic data	0.86	0.05	0.36

2.1.2.1.4. Execution Time

Table 8 shows the execution time in milliseconds for Orion, Stream LOCI and A-ODDS. Orion has better execution time compared to Stream LOCI for all datasets except the EMG physical action dataset. The EMG physical action dataset has a small number of dimensions; hence Stream LOCI shows superior execution time for the EMG physical action dataset. This is because for a small number dimensions, Euclidean distance computation between two data points takes short time. Moreover, Euclidean distance is very effective for a small number of dimensions; hence, Stream LOCI can identify the outliers while considering small local neighborhood for outlier detection. So, it can skip the search of outliers in large local neighborhood. Therefore, the execution time for Stream LOCI is shorter than that of Orion. However the execution

time increases significantly for Stream LOCI for other datasets. The execution time of Orion for high dimensional data is always better than that of Stream LOCI.

A-ODDS shows better execution time compared to Orion. However, the execution time of both of these algorithms is very competitive and, in the Australian sign, Irrigation and Synthetic datasets, Orion shows better execution time compared to A-ODDS. Moreover, Orion offers much better accuracy compared to A-ODDS.

The maximum execution time we receive in our experiments is 4.34 milliseconds which is practically good enough for many data stream applications including environmental monitoring [6]. In these applications the arrival rate is in the order of seconds and the arrival rate more frequent than that is practically useless [6]. In these applications, we have an adequate amount of time to execute Orion and detect their outliers.

Table 8. Execution time (in ms) of all algorithms for all datasets

	Orion	Stream LOCI	A-ODDS
KDD'99	1.25	2.81	0.40
Vicon Physical Action	4.34	6.85	0.87
Australian Sign	0.15	0.56	0.55
EMG Physical Action	0.78	0.18	0.07
Irrigation data	0.25	3.48	0.65
Synthetic data	1.58	3.27	1.76

We have discussed our overall results based on the real and synthetic datasets. In order to perform further analysis we have to manipulate different parameters of the datasets which is impossible with the four real datasets we obtain from the UCI machine learning repository [96]. Therefore for further analysis we only use two datasets: (1) irrigation data where the data points are real and we synthesize the outliers, and (2) synthetic data where we simulate real world time series data with an appropriate set of outliers.

2.1.2.2. Impact of Neighbor Distance

2.1.2.2.1. Precision

Figures 23 and 24 portray the impact of neighbor distance on precision. Neighbor distance is used to calculate the neighbor density of a data point. If the neighbor distance is small, the obtained neighbor densities of all data points are comparatively small compared to the case if the neighbor density is large. Neighbor distance depicts how far Orion should look to find a neighbor of a data point. Typically, for any given neighbor distance, outliers would have a significantly smaller neighbor density compared to inliers. Therefore, Orion would be able to separate outliers from inliers. For a smaller or larger neighbor distance, the induced neighbor density of both inliers and outliers is shortened or lengthened appropriately. Thus, Orion shows little impact with respect to neighbor distance.

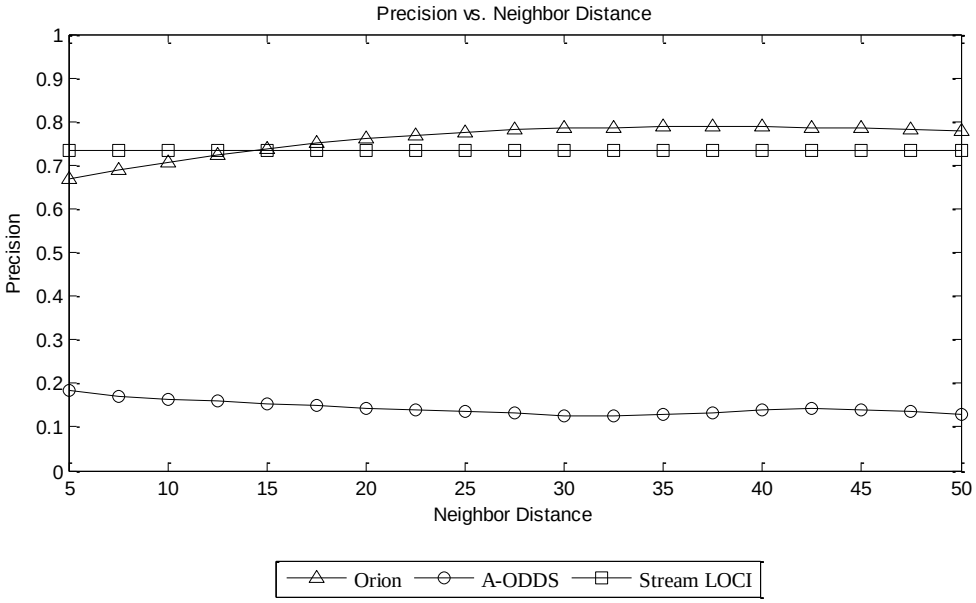

Figure 23. Impact of neighbor distance on precision for the irrigation data

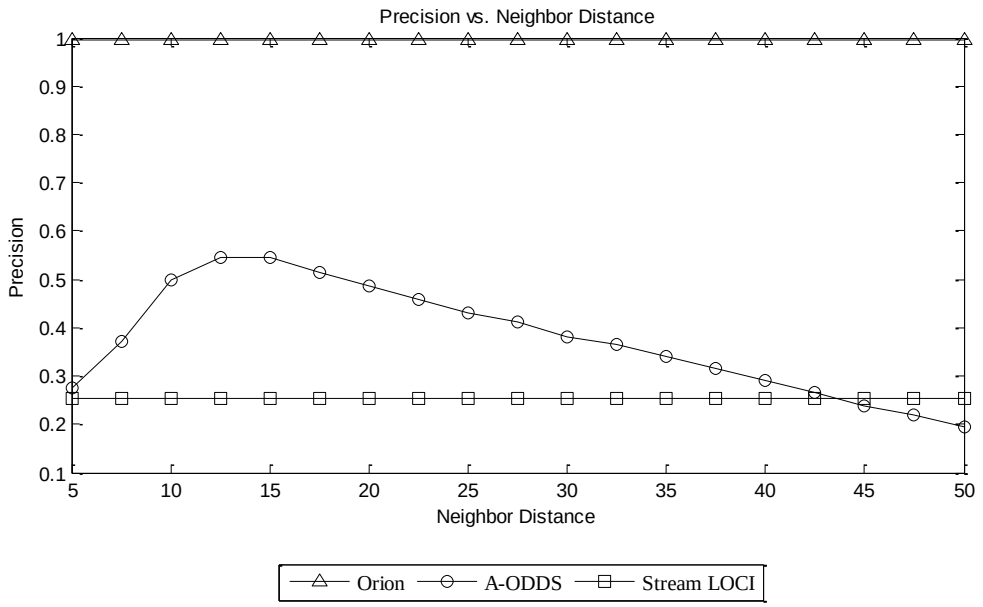

Figure 24. Impact of neighbor distance on precision for the synthetic data

In both irrigation data and synthetic data, neighbor density has very little impact on Orion. This is particularly useful since Orion cannot be fooled by choosing neighbor distance wrongly. Interestingly A-ODDS also uses neighbor distance to compute its global deviation factor (GDF) and local deviation factor (LDF). A-ODDS shows a great deal sensitivity with respect to neighbor distance. The precision is small at the beginning when neighbor distance is small. As neighbor distance increases precision increases as well. This is because for small neighbor distances, some inliers (the data points that are in the periphery of a cluster) also have very small GDF and LDF values and, therefore, are identified as outliers. As neighbor distance increases, those data points blend with regular inliers and the precision reaches its peak value. If we increase the neighbor distance even further, the precision starts to decrease again. This is because in that case outliers are also having large neighbor density. Thus the precision start to decrease again.

Since Stream LOCI does not require neighbor distance, the precision of Stream LOCI shows no sensitivity with respect to neighbor distance.

2.1.2.2.2. Recall

Although the precision of Orion shows no sensitivity with respect to neighbor distance, the recall of Orion shows some sensitivity with respect to neighbor distance (Figures 25 and 26). Larger neighbor distance induces greater neighbor density for both inliers and outliers. Hence the neighbor density of outliers increases with the neighbor distance. As the neighbor density increases some outliers have large enough neighbor density so that they can be considered as inliers. Thus Orion misclassifies them as inliers; hence Orion fails to reveal some outliers and its recall decreases with the increase of neighbor distance.

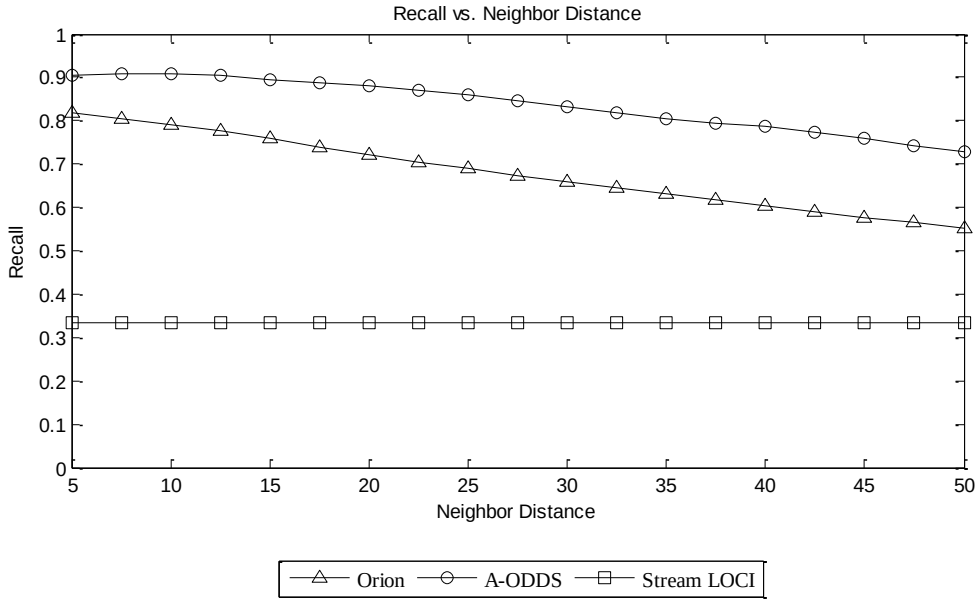

Figure 25. Impact of neighbor distance on recall for the irrigation data

A similar result pattern is also visible with A-ODDS. A-ODDS also computes GDF and LDF based on neighbor distance. If the neighbor distance increases it fails to separate the outliers from inliers and shows the overall drop of recall value.

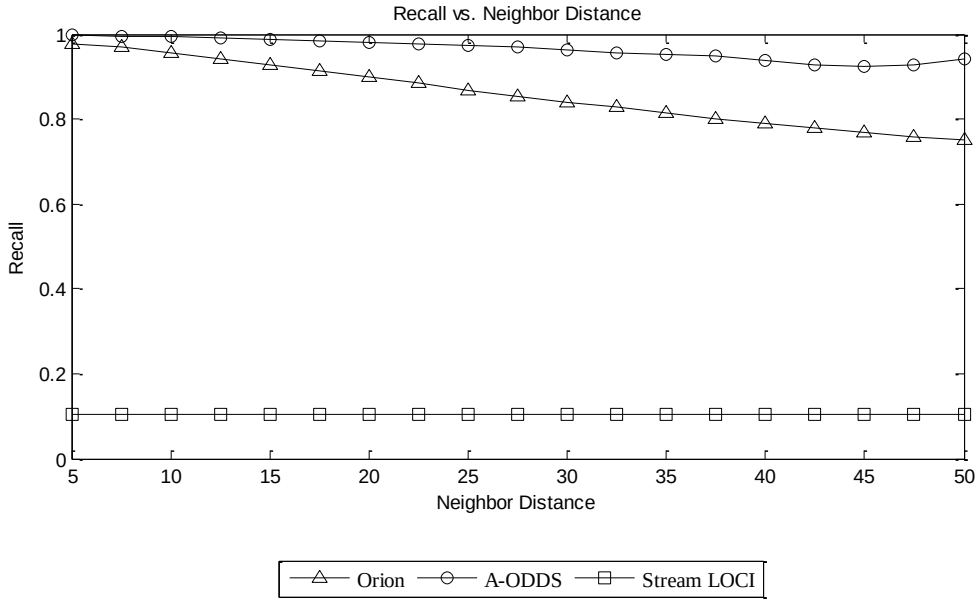

Figure 26. Impact of neighbor distance on recall for the synthetic data

However, since Stream LOCI does not require neighbor distance as input, it does not show any sensitivity with respect to neighbor distance at all. The sensitivity of recall with respect to neighbor distance for both the irrigation data and synthetic data is the same.

2.1.2.2.3. Jaccard Coefficient

Jaccard Coefficient (JC) is the combination of both precision and recall values. Since the precision of Orion remains the same and the recall of Orion decreases with the increase of neighbor distance, the JC of Orion decreases a little as well. The decrease of JC is induced by some false negatives where Orion fails to identify some outliers. However, the JC of Orion is still much better than those of A-ODDS and Stream LOCI.

Orion's worst JC is much better than the average JC of Stream LOCI (51% better for irrigation data and 1110% better for synthetic data). Figures 27 and 28 show the change of JC for all three algorithms for the irrigation and synthetic data.

The JC and recall of A-ODDS show a similar pattern. The JC of A-ODDS starts to increase with the increase of neighbor distance. This increase of JC is induced by the increase of precision of A-ODDS. Since precision increases with the increase of neighbor distance, JC increases as well and then after that, precision starts to decrease with the increase of neighbor distance. The JC of Stream LOCI remains unchanged with respect to the change of neighbor density; this is because it does not require neighbor density as input.

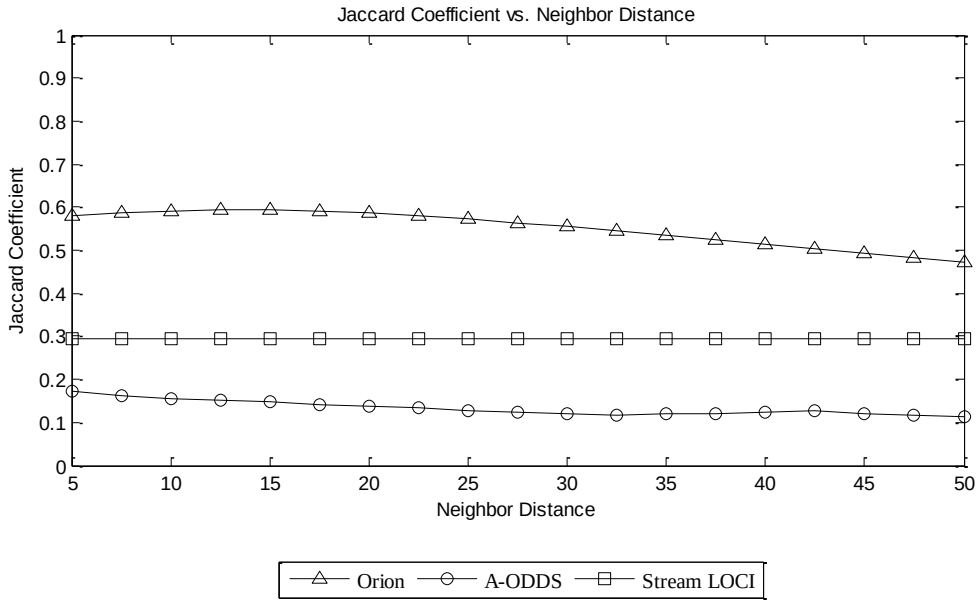

Figure 27. Impact of neighbor distance on Jaccard Coefficient for the irrigation data

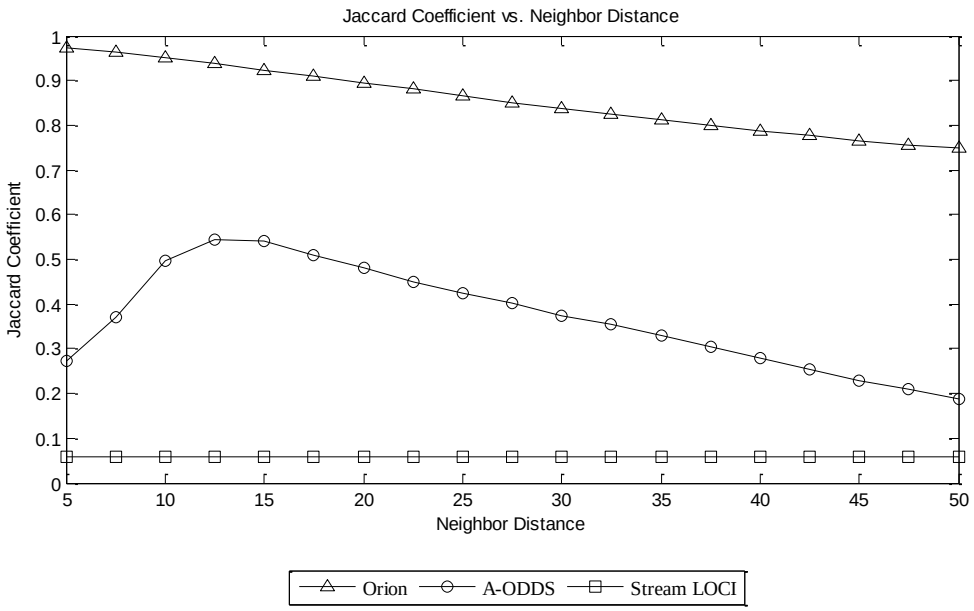

Figure 28. Impact of neighbor distance on Jaccard Coefficient for the synthetic data

2.1.2.2.4. Execution Time

Figure 29 shows the impact of neighbor distance on execution time for the irrigation data. Orion does not show any variation with respect to change of neighbor distance. This is because our neighbor density computation algorithm takes a constant time. Hence the time required to compute the neighbor density from a given neighbor distance is the same regardless of the value of neighbor distance and, thus, the execution time of Orion remains unchanged with respect to neighbor distance. Figure 29 shows the same results for the synthetic data.

However, the execution time of A-ODDS increases linearly with the increase of neighbor distance. This is because the GDF and LDF computation times in A-ODDS increase linearly with the increase of neighbor distance. Hence, as the neighbor distance increases the execution time of A-ODDS also increases.

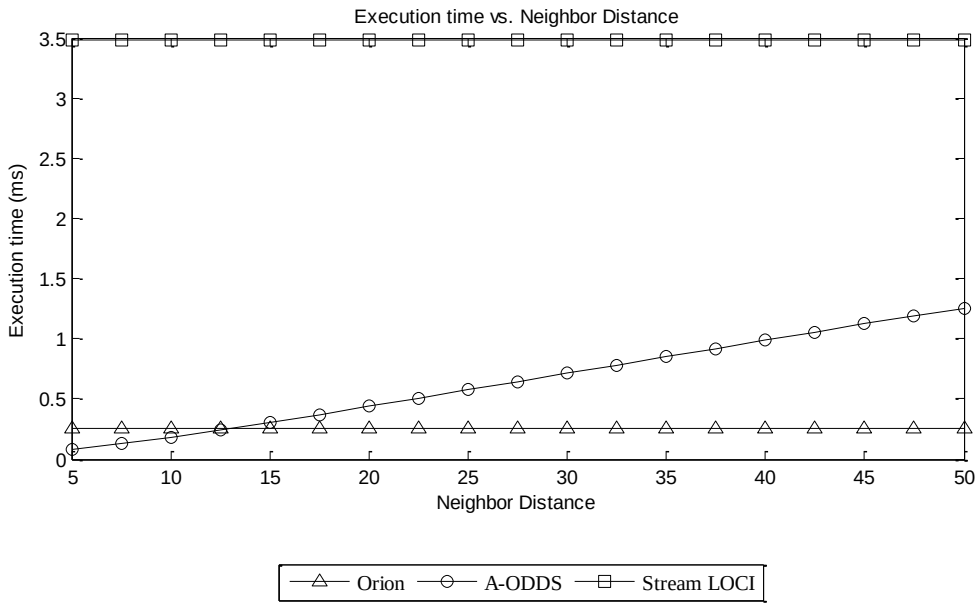

Figure 29. Impact of neighbor distance on execution time for the irrigation data

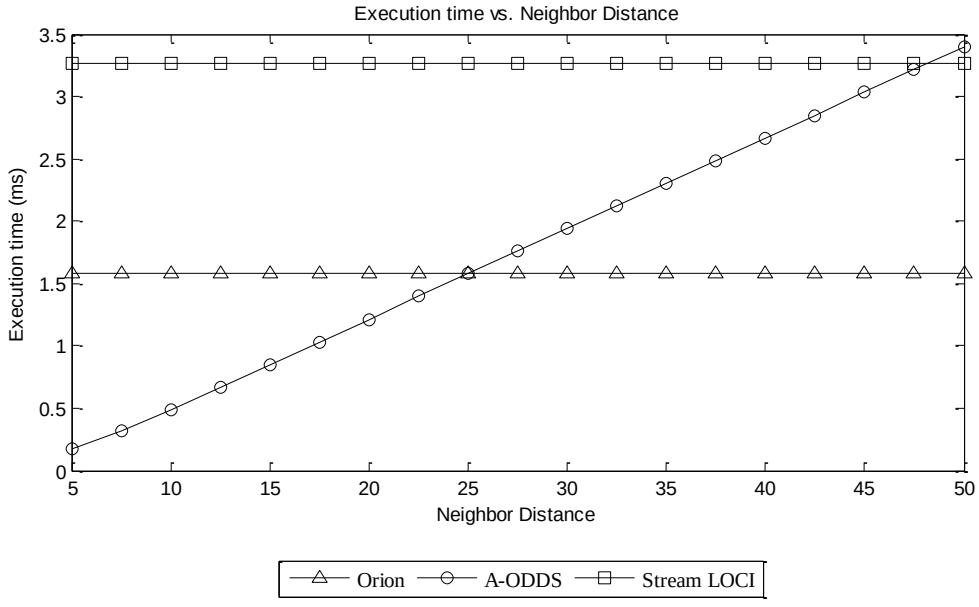

Figure 30. Impact of neighbor distance on execution time for the synthetic data

Like other performance metrics, the execution time of Stream LOCI remains unchanged with respect to change of neighbor distance.

2.1.2.3. Impact of k

The second most important parameter for Orion is k. The parameter k dictates the percentage of neighbors of a data point that we need to look for so that we can tell whether the data point is far from rest of the data points. For each data point, Orion computes the distance that includes k percent of the data points, which we call k-distance. If a data point requires a small k-distance to include k percent of data points it is certainly surrounded by lots of other data points and is unlikely to be an outlier. Conversely, if a data point requires a large k-distance to include k percent of data points, then the data point may be isolated from other data points and, hence, is most likely an outlier. This section presents the study of the sensitivity of the three outlier detection algorithms with respect to k.

2.1.2.3.1. Precision

Figures 31 and 32 show the impact of k on precision for the irrigation and synthetic data. The precision of Orion for the irrigation data decreases with the increase of k. This is because as the value of k increases each data point has to have a longer k-distance to include k percent of the data points. Some inliers that are along the periphery of a group of data points also needs to have a larger distance to include k percent of the data and hence they look like outliers. Thus Orion creates some false alarms and therefore its precision decreases with the increase of k.

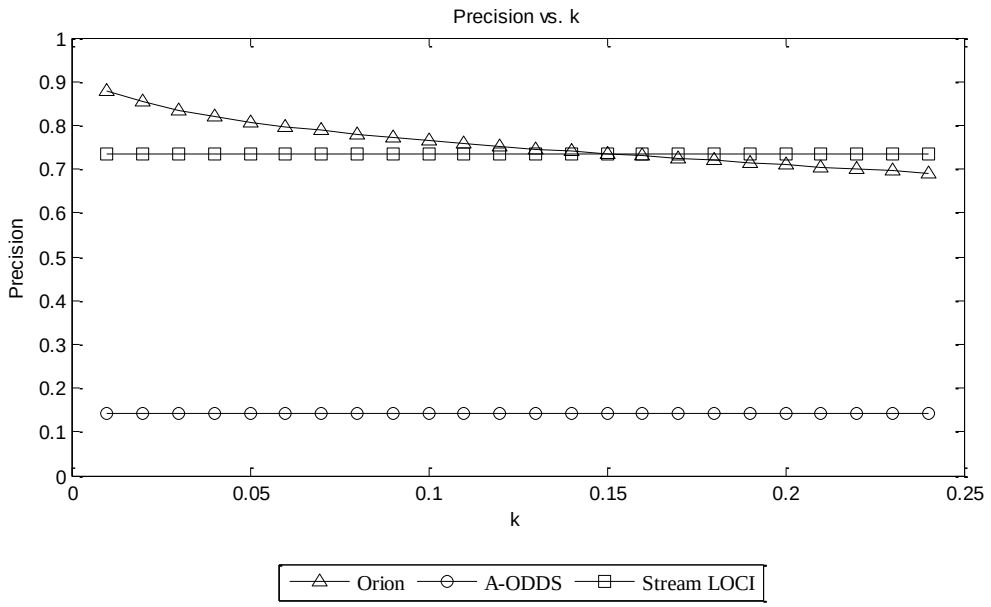

Figure 31. Impact of k on precision for the irrigation data

However, Figure 32 does not show any sensitivity in any algorithm with respect to k at all for the synthetic dataset. This is because in this dataset we have less noise compared to the irrigation data and hence the outliers are easily identified by Orion. Although the k-distance increases with the increase of k, the k-distances of inliers and outliers increase proportionately and thus they have no impact on the precision of Orion. Meanwhile, since A-ODDS and Stream LOCI do not require k as an input parameter, they are not sensitive to k.

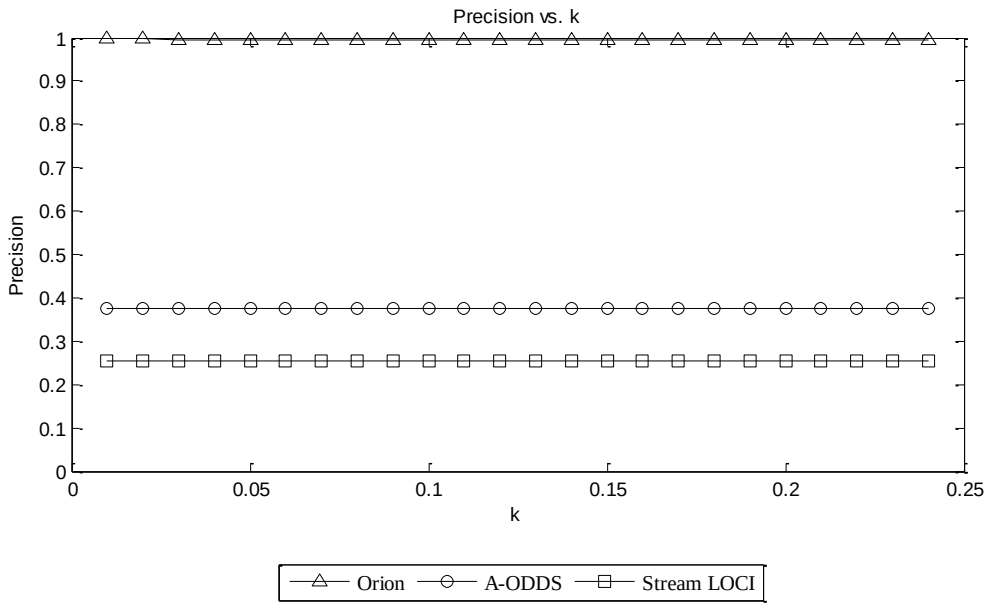

Figure 32. Impact of k on the precision for the synthetic data

2.1.2.3.2. Recall

Conversely, the recall of Orion increases a little bit with the increase of k. Figure 33 shows the impact of k on the recall of Orion. The recall increases with the increase of k because, for a large k, it becomes extremely difficult for outliers to have smaller k-distances and thus outliers cannot blend with inliers. Hence, the recall value increases with the increase of k. The increase of the recall value with the increase of k is consistent for both the irrigation data (Figure 33) and synthetic data (Figure 34). Since A-ODDS and Stream LOCI do not require k as an input parameter, they show no sensitivity in terms of recall for varying k.

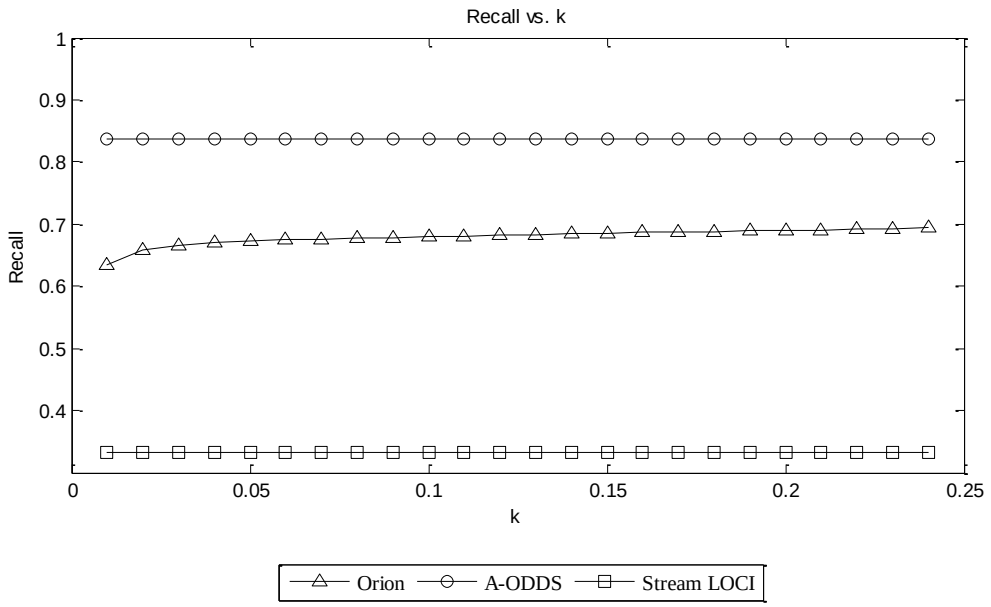

Figure 33. Impact of k on the recall for the irrigation data

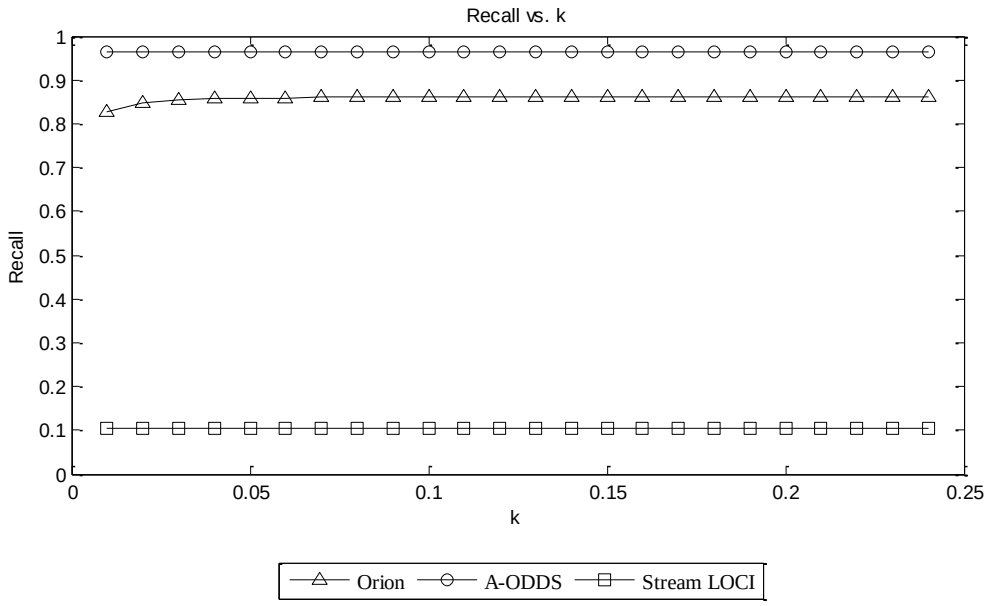

Figure 34. Impact of k on precision for the synthetic data

2.1.2.3.3. Jaccard Coefficient

The Jaccard coefficient (JC) of Orion shows a more interesting trend with respect to the change of k for the irrigation data (Figure 35). The JC almost decreases a little bit with

the increase of k. This is because the precision decreases with the increase of k, which also decreases the JC; however the recall increases with the increase of k, which increases the JC a little bit. Therefore, the overall trend of JC is almost constant at the beginning, and decreases a little bit after that, while the recall stops increasing after $k = 0.05$; but the change is very insignificant.

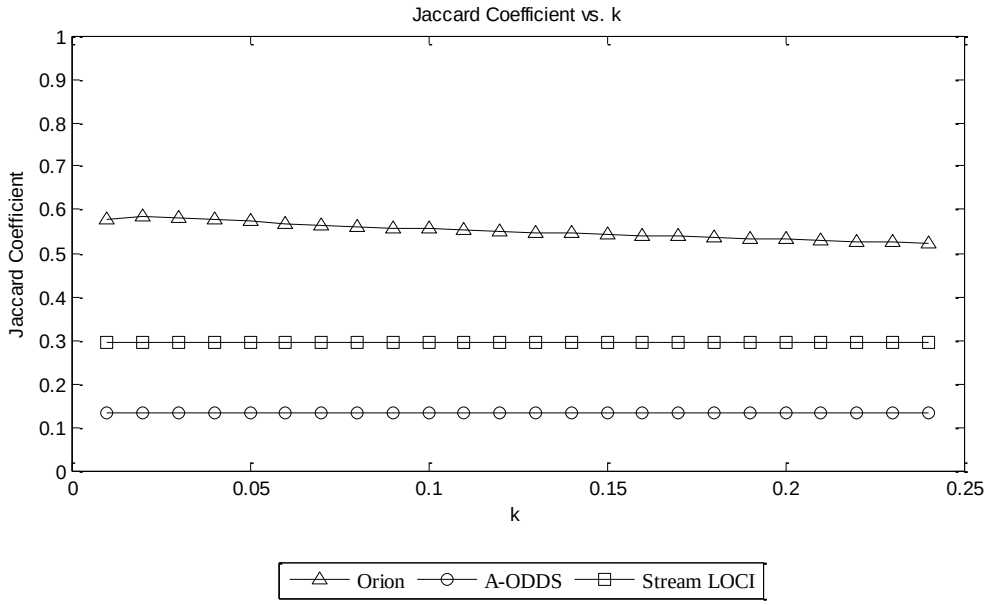

Figure 35. Impact of k on Jaccard Coefficient for the irrigation data

The JC of Orion for the synthetic data shows an opposite trend compared to that for the irrigation data (Figure 35). With the synthetic data, the JC of Orion increases with the increase of k. This is because the precision shows no impact for the synthetic data and the recall increases a little bit with the increase of k, JC increases with the increase of k. However, the increase is very insignificant for the synthetic data as well. Like the precision and recall of A-ODDS and Stream LOCI, the JC of A-ODDS and Stream LOCI show no variation with respect to the change of k.

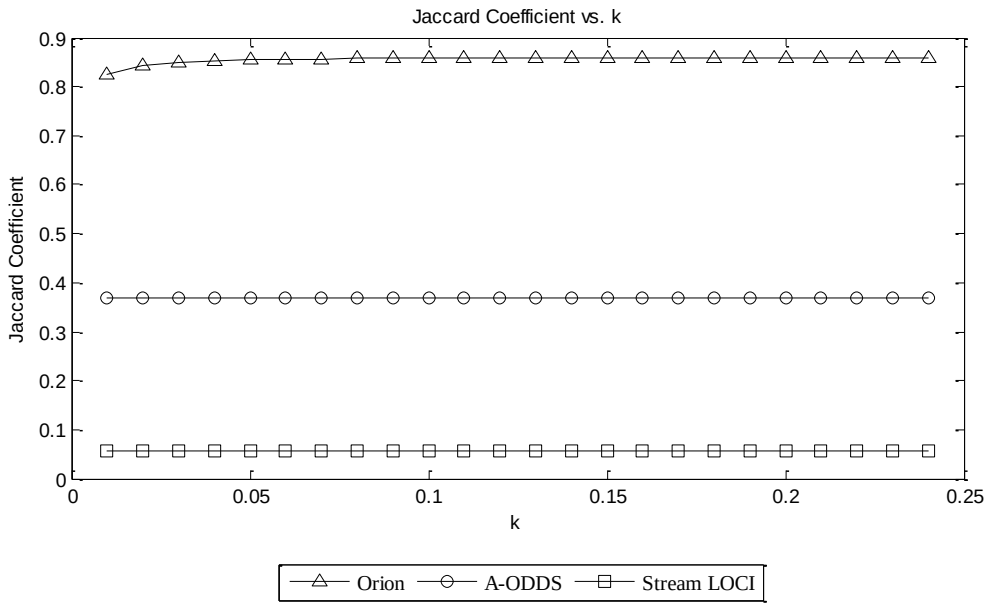

Figure 36. Impact of k on the Jaccard Coefficient for the synthetic data

2.1.2.3.4. Execution Time

The k-distance computation takes $log\ (binCount)$ amount of time, which is a constant with respect to the value of k. Thus the impact of changing k is not visible in the execution time of Orion. Thus, Orion shows almost a constant execution time with respect to the change of k. This trend is persistent for both the irrigation data (Figure 37) and synthetic data (Figure 38). The execution times of A-ODDS and Stream LOCI are not impacted by the change of k either because they do not require k as an input parameter.

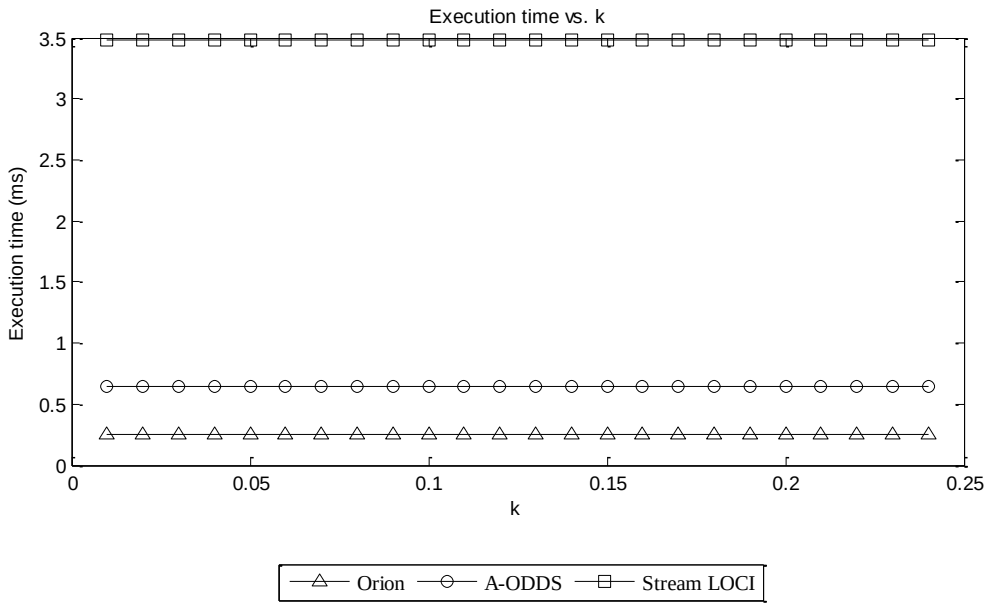

Figure 37. Impact of k on the execution time for the irrigation data

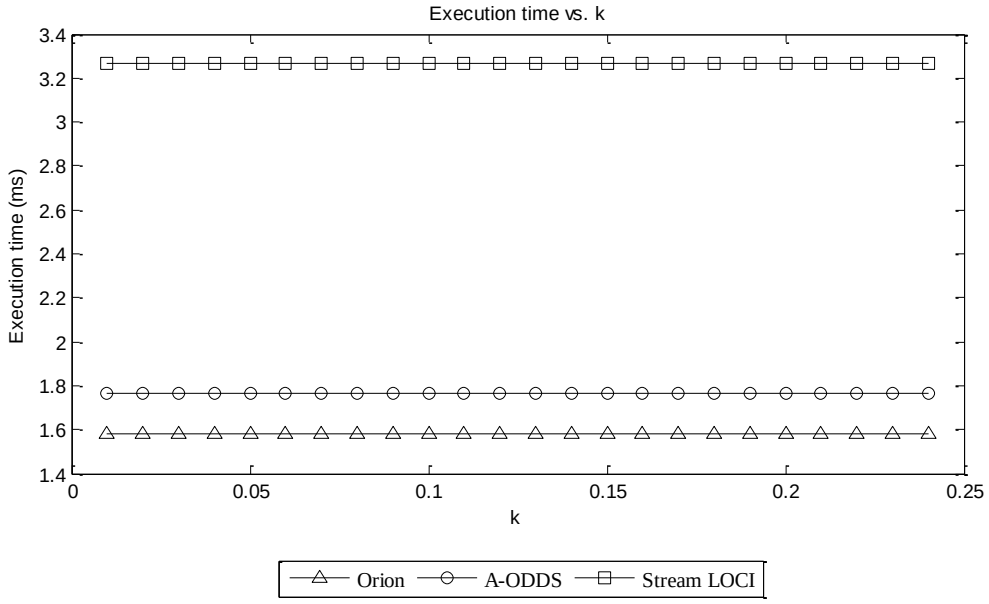

Figure 38. Impact of k on the execution time for the synthetic data

2.1.2.4. Impact of Percentage of Outliers

The impact of percentage of outliers is very important for the performance study. Many algorithms assume the percentage of outliers to be very low compared to the percentage

of inliers [9]. A high percentage of outliers destroys the underlying trends of the data. In this section we study the performance of the competitive algorithms with respect to the percentage of outliers.

2.1.2.4.1. Precision

Figures 39 and 40 show the impacts of the percentage of outliers on the precision of Orion, A-ODDS and Streams LOCI. The precision increases with the increase of the percentage of outliers in Orion and Stream LOCI. This is because both the algorithms create some false alarms. The data points that are a little bit far from the other data points are mistakenly identified as outliers. All the algorithms create these false alarms. The number of false alarms remains the same regardless of the number of outliers. Hence if the number of outliers increases, the ratio of false alarms to true outliers becomes small and, hence, the precision increases with the increase of the percentage of outliers.

Moreover, the percentage of outliers has a secondary impact on our co-clustering approach. As the number of outliers increases, the number of data points that have small neighbor density and large k-distance also increases (this is because outliers have small neighbor density and large k-distance). A large number of data points with small neighbor density and large k-distance shifts the cluster centers toward smaller neighbor density and larger k-distance. Hence, Orion becomes more pessimistic; meaning it would identify a data point as an outlier if it has very small neighbor density and large k-distance and, therefore, the number of false alarms decreases (from 15% to 1%) with the increase of percentage of outliers.

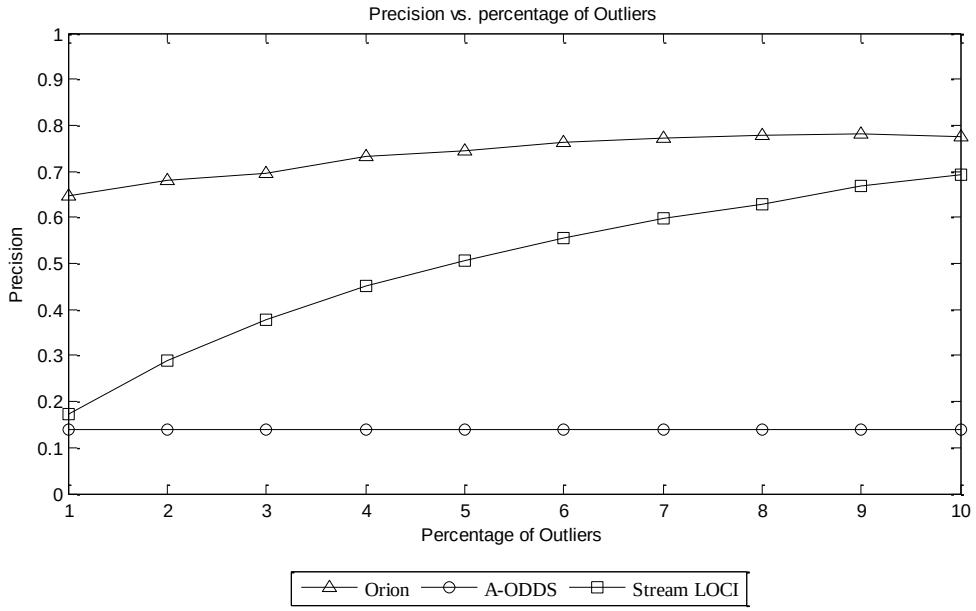

Figure 39. Impact of the percentage of outliers on the precision for the irrigation data

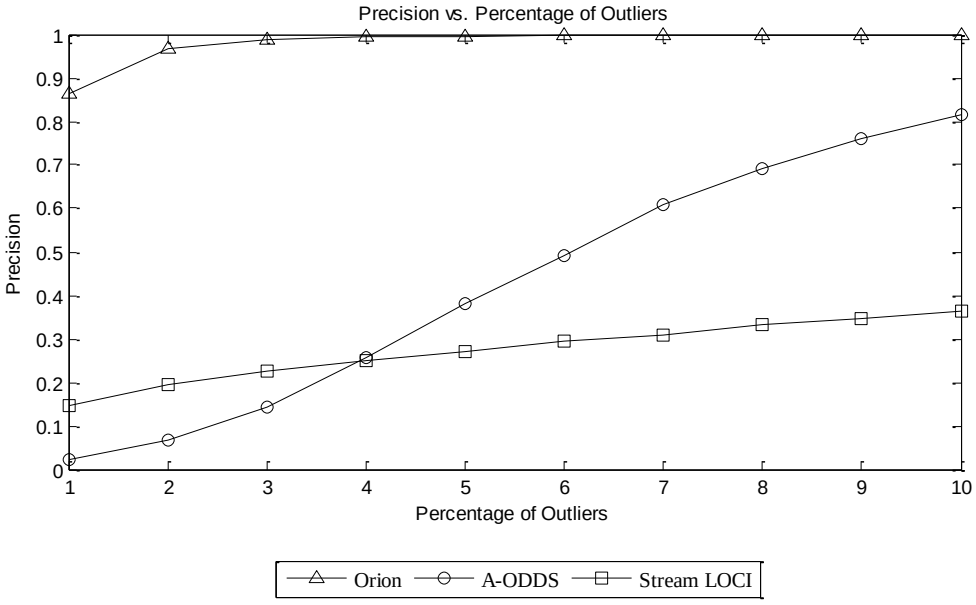

Figure 40. Impact of the percentage of outliers on the precision for the synthetic data

A-ODDS does not show any sensitivity with respect to the change of the percentage of outliers. This is because the rates of true outliers and misclassified outliers of A-ODDS

are approximately the same regardless of the percentage of outliers. However, if it can correctly identify all the outliers, its rate of change of precision should be proportional. But that is not the case we find in Figures 39 and 40. This is because the correctness of detecting new outliers is different in Orion, A-ODDS and Stream LOCI. Thus the precision increases in all three algorithms but the rate of increase is different in all of them.

2.1.2.4.2. Recall

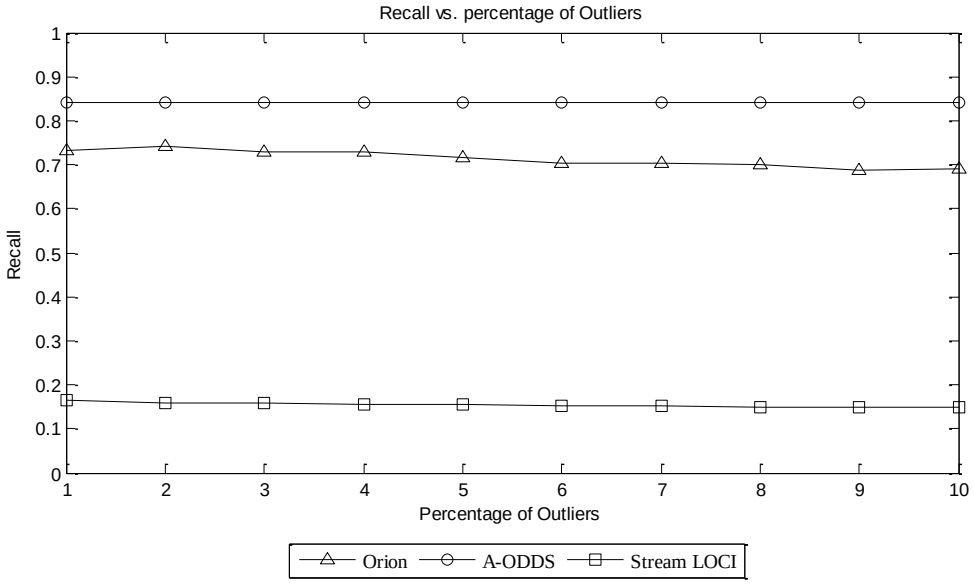

Figure 41. Impact of the percentage of outliers on the recall for the irrigation data

The recall of Orion decreases as the percentage of outliers increases (Figures 41 and 42). As the percentage of outliers increases, the number of outliers increases; and the number of data points with small neighbor density and large k-distance increases as well. The large number of data points with small neighbor density and large k-distance shift the cluster centers and Orion becomes pessimistic, meaning it does not identify a data point as an outlier unless it has very high k-distance and very small neighbor

density. Therefore, it fails to identify those outliers which are not very far from other data points. Hence, we see the decrease of recall in Orion with the increase of percentage of outliers.

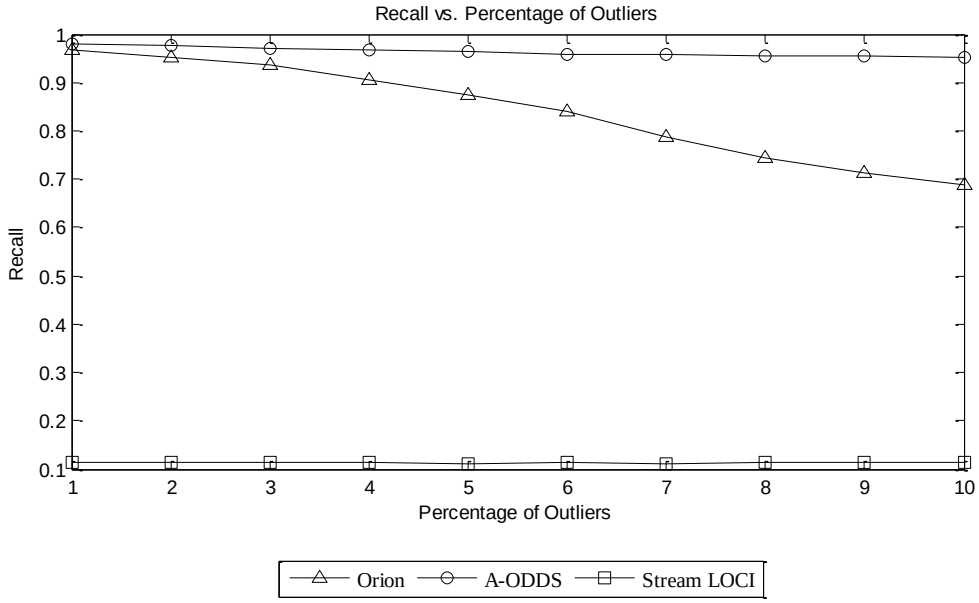

Figure 42. Impact of the percentage of outliers on the recall for the synthetic data

Interestingly, A-ODDS and Stream LOCI show no variation with respect to the change of the percentage of outliers because they do not have the clustering step and hence they do not group the outliers together. However, we conduct our experiments with up to 10% of the data points as outliers, which is a rather high percentage of outliers since a typical percentage of outliers is assumed to be within 0.001 – 5% [98] and the recall of Orion is always better than those of A-ODDS and Stream LOCI.

2.1.2.4.3. Jaccard Coefficient

The impact of the percentage of outliers on the JC of Orion is very interesting. The JC increases a little bit but the change of JC is very insignificant (Figure 43). Since the precision increases and the recall decreases with the increase of the percentage of

outliers, the JC remains almost the same with respect to the change of the percentage of outliers.

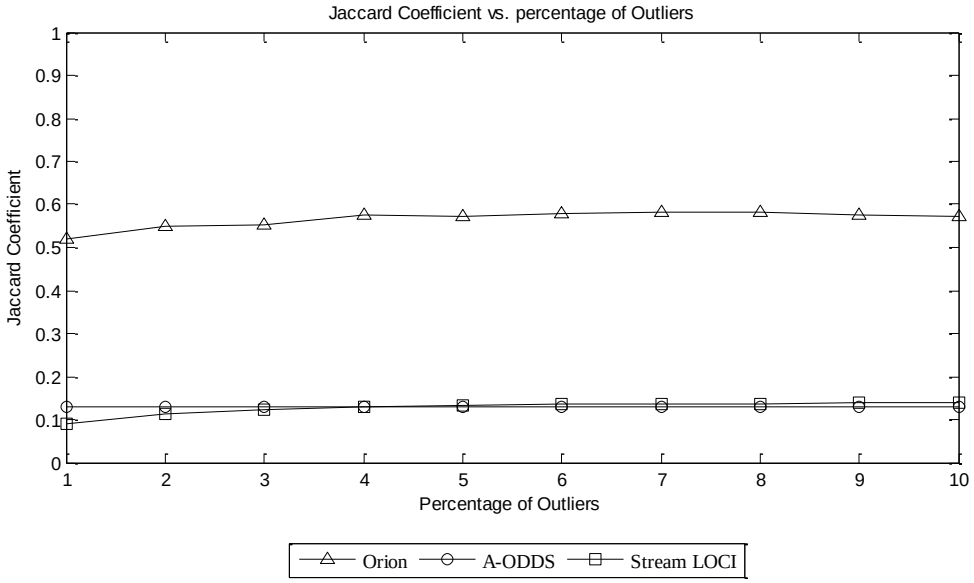

Figure 43. Impact of the percentage of outliers on the Jaccard Coefficient for the irrigation data

The JC of Orion also decreases with the increase of the percentage of outliers for the synthetic data (Figure 44). This decrease is induced by the recall. The JC of Orion for the synthetic data is more vivid compared to that for the irrigation data. In the synthetic data, the recall decrease is more comparable to the increase of the precision and, thus, overall the JC decreases with the increase of the percentage of outliers. A-ODDS show the change of JC with respect the change of percentage of outliers. The JC of A-ODDS increases when the percentage of outliers increases. This is because A-ODDS assumes that data points with GDF and LDF higher than three standard deviations are outliers. Theoretically, approximately 10% of the data points have GDF or LDF higher than three standard deviations [60]. If a dataset has less than 10% of outliers, many inliers satisfy the outlier criteria and are identified as outliers. As the percentage of outliers

approaches 10%, outliers replace those inliers in the identified set of outliers; hence, the JC of A-ODDS increases with the increase of percentage of outliers.

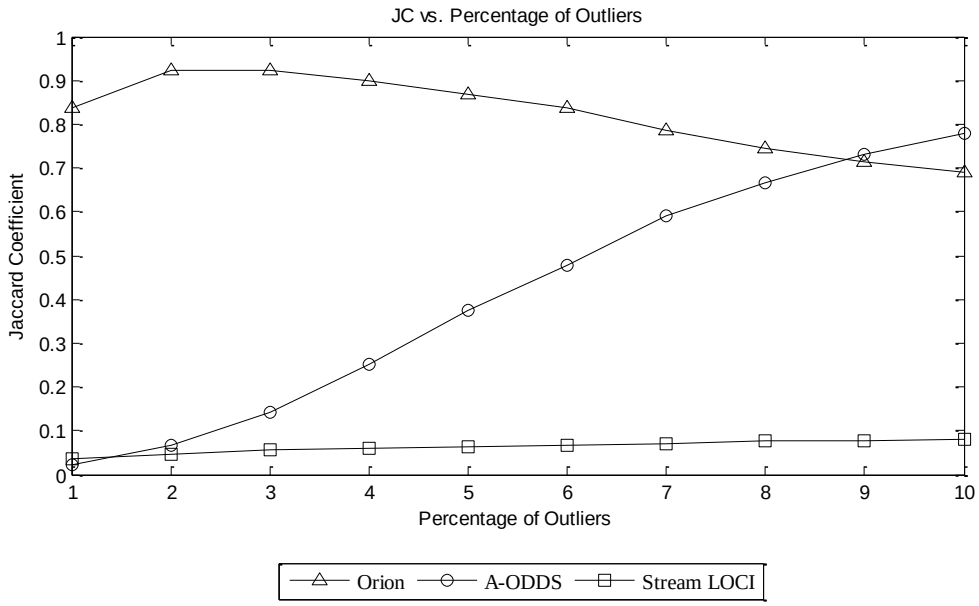

Figure 44. Impact of the percentage of outliers on the Jaccard Coefficient for the synthetic data

The impact of percentage of outliers on JC of Stream LOCI is insignificant. The JC of Stream LOCI remains almost constant with the increase of percentage of outliers. This is because as the number of outliers increases, the ratio between the number of identified outliers by Stream LOCI and the total number of outliers remain same. Hence, the JC of Stream LOCI remains the same with respect to the increase of percentage of outliers.

2.1.2.4.4. Execution Time

Orion shows no variation of execution time with respect to the change of the percentage of outliers. This is because the number of tasks Orion performs (finding appropriate p-dimension, computing outlier metrics, co-clustering data points, updating fitness values

and updating the data density function) do not depend upon the outlier-ness of a data point. Regardless the outlier-ness of a data point, Orion performs these steps and they require an equal amount of time. Thus, the execution time of Orion remains fixed with respect to the change of percentage of outliers. The execution times of A-ODDS and Stream LOCI decrease when the percentage of outliers increases (Figures 45 and 46). Since A-ODDS considers each dimension independently and once it finds an outlier in one dimension, it discards the processing of the other dimensions, and thus its execution time decreases as the percentage of outliers increases (Figure 46). Stream LOCI does not consider each dimension independently but considers multiple radiuses for MDEF. As the percentage of outliers grows, Stream LOCI can detect outliers for a smaller radius, abandons its execution for a larger radius and, therefore, shows a better execution time for a larger percentage of outliers (Figure 46).

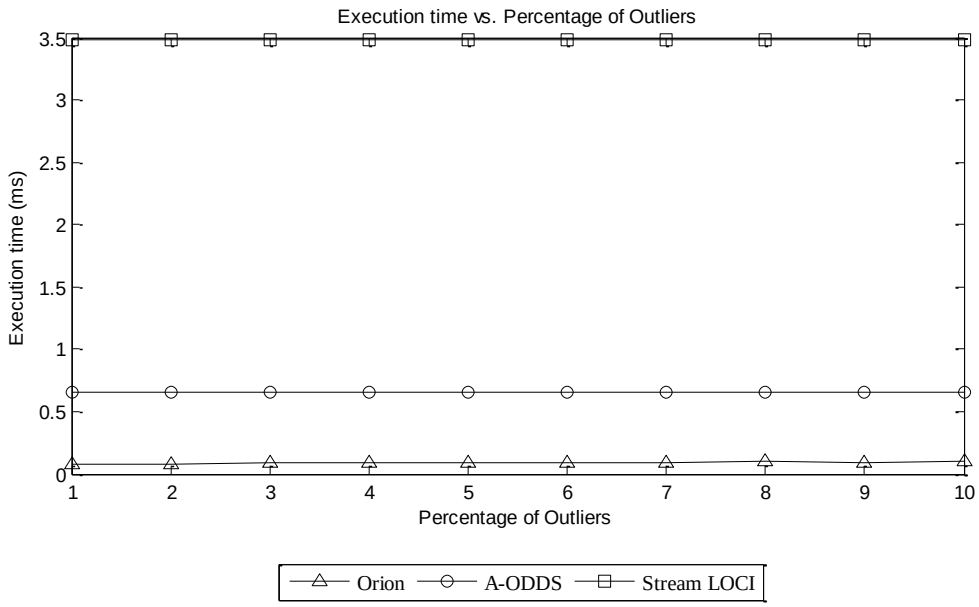

Figure 45. Impact of the percentage of outliers on the execution time for the irrigation data

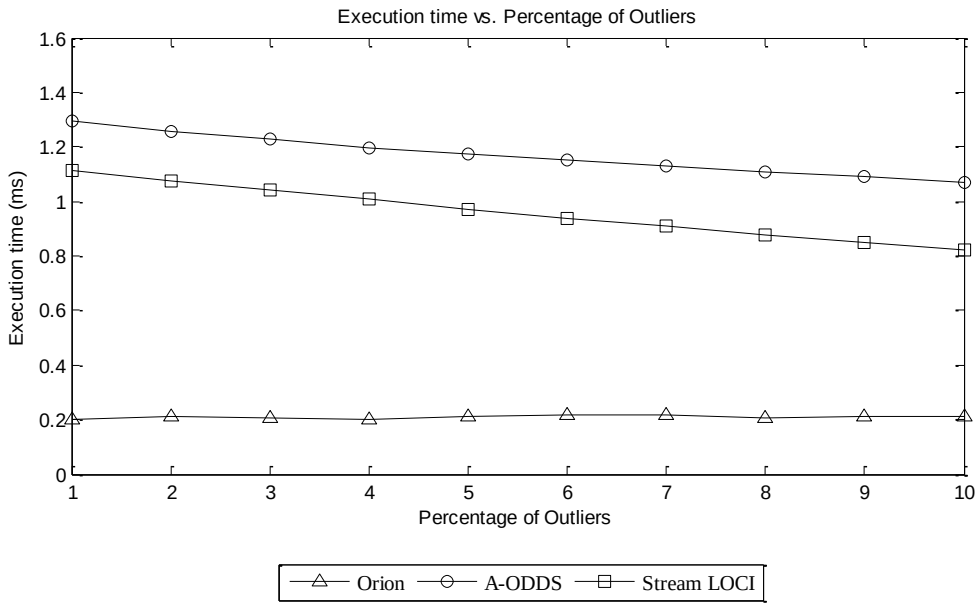

Figure 46. Impact of the percentage of outliers on the execution time for the synthetic data

2.1.2.5. Impact of Number of Dimensions

We designed Orion for multi-dimensional data streams. As the number of data dimensions grows, it becomes extremely difficult to measure the similarity between data points and hence detecting outliers becomes a difficult task. Therefore, in this experiment, we study the impact of number of dimensions on the accuracy and execution time of all three algorithms. In this experiment we only use the synthetic data since we can only change the number of dimensions for the synthetic data. The number of dimensions is fixed for the irrigation data and hence we omit the irrigation data for this experiment.

2.1.2.5.1. Precision

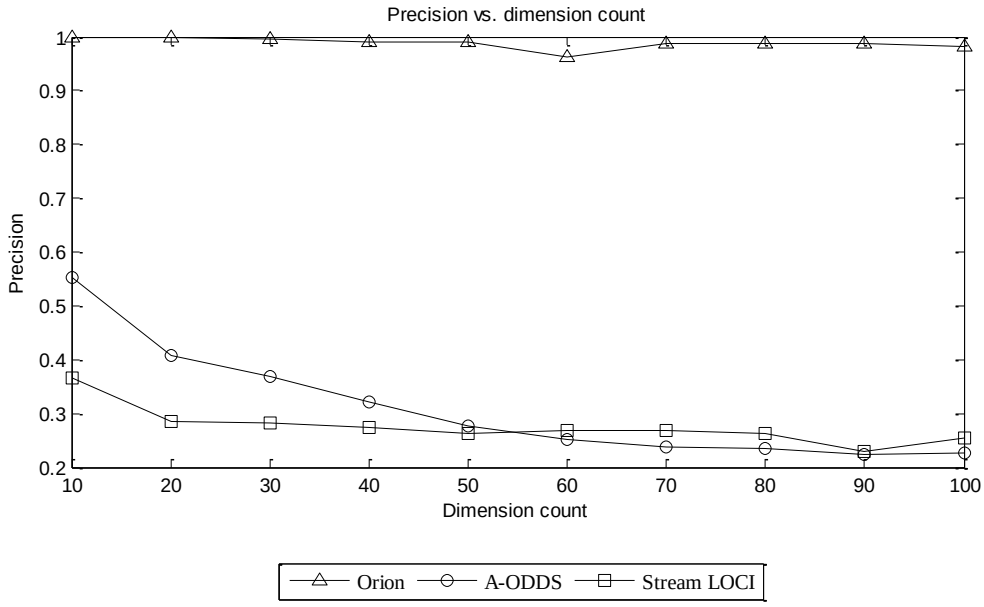

Figure 47. Impact of the number of dimensions on the precision for the synthetic data

Orion performs better than the other two algorithms in terms of precision regardless of the number of dimensions as shown in Figure 47. Orion's precision shows no variation at all when the number of dimensions changes. This is because whenever a data point is detected as an outlier in Orion, it is indeed an outlier with respect to at least one p-dimension. Hence the number of dimensions does not impact Orion's precision. However, the precisions of A-ODDS and Stream LOCI decrease exponentially with the increase of the number of dimensions. Stream LOCI measures the similarity among the data points using Euclidean distance; as the number of dimensions increases, Stream LOCI fail to measure the similarity among the data points using Euclidian distance; and therefore the precision of Stream LOCI decreases exponentially with the linear increase of number of dimensions. A-ODDS does not consider multiple dimensions at all.

Hence, none of them is effective for outlier detection for data with a large number of dimensions.

2.1.2.5.2. Recall

The recall of Orion increases with the increase of the number of dimensions (Figure 48). As the number of dimensions increases the multi-dimensional space grows exponentially and it becomes easier to find an appropriate p-dimension for outliers. Therefore Orion can successfully find the appropriate p-dimensions for outliers and its recall increases with the number of dimensions. The recall of A-ODDS also increases as well. This is because our outliers are generated randomly and A-ODDS checks each dimensions independently. If the number of dimensions increases, it becomes more likely that A-ODDS can find abnormality in one dimension of the data points and hence its recall increases a little bit.

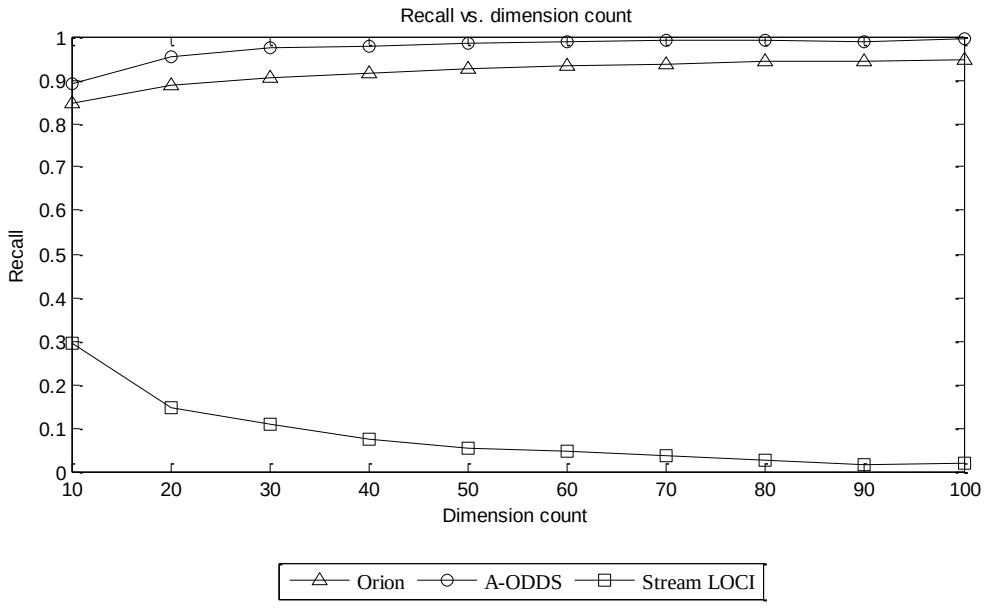

Figure 48. Impact of the number of dimensions on the recall for the synthetic data

However, the recall of Stream LOCI decreases exponentially again. This is because as the number of dimensions grows, Euclidian distance completely fails to measure the similarity between data points; and thus, outliers and inliers all look similar and all are equidistance from one another. Thus the recall of Stream LOCI decreases with the increase of the number of dimensions.

2.1.2.5.3. Jaccard Coefficient

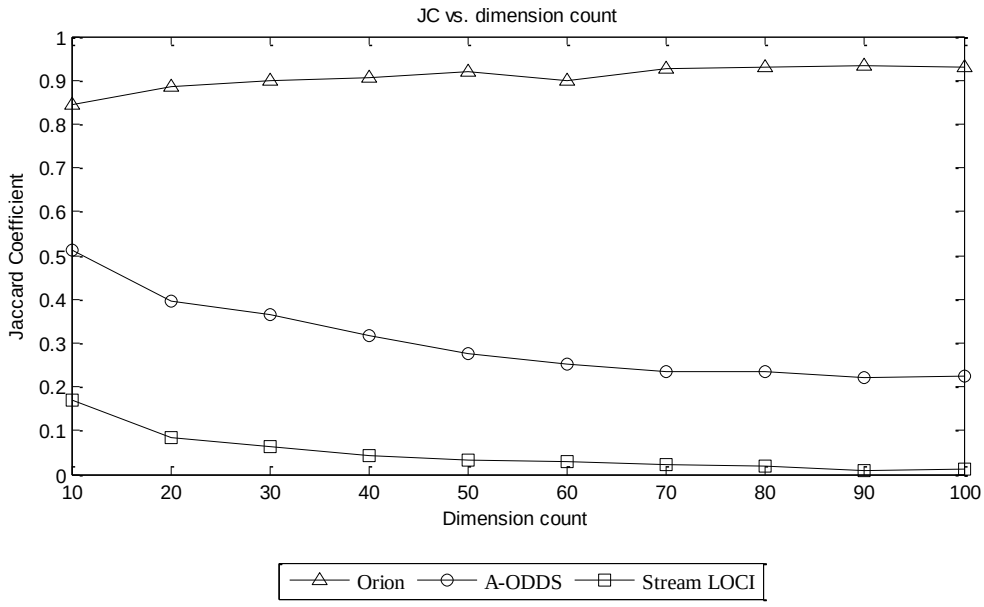

Figure 49. Impact of the number of dimensions on the Jaccard Coefficient for the synthetic data

Figure 49 shows the impact of the number of dimensions on the JC of all three algorithms. Overall the JC of Orion shows no impact, while the JCs of A-ODDS and Stream LOCI decrease exponentially with the linear increase of the number of dimensions, and thus are not suitable for outlier detection for multi-dimensional data streams.

2.1.2.5.4. Execution Time

Although Orion can successfully detect outliers regardless of the number of dimensions, its execution time grows in a cubic way with the linear increase of the number of dimensions. According to our theoretical analysis in Section 1.1.1.1, the time complexity of Orion was cubic with respect to number of dimensions. We can see the similar increase of execution time with respect to number of dimensions. The cubic time complexity is induced by the matrix inversion which is required to compute the maximum absolute normalized deviation. However, its execution time for up to 100 dimensions is easily tractable. Stream LOCI and A-ODDS also show a similar trend: the execution time increases with the increase of number of dimension. The execution time of Stream LOCI increases because as the number of dimensions increases, the Euclidian distance computation time increases linearly with the linear increase of number of dimensions. So, the execution time of Stream LOCI increases linearly with respect to number of dimensions.

A-ODDS considers each dimension individually and executes outlier detection independently. As the number of dimensions increases, A-ODDS needs to consider more dimensions, therefore, the execution time of A-ODDS increases with the increase of number of dimensions.

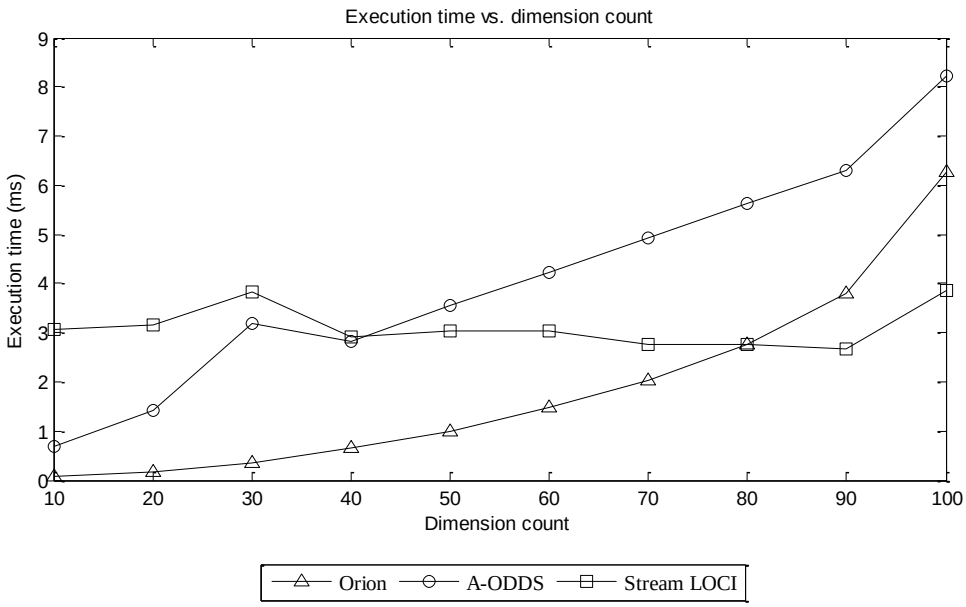

Figure 50. Impact of the number of dimensions on execution time for the synthetic data

2.1.2.6. Impact of Population Count

Orion is an evolutionary algorithm, in which it starts with an initial set of solutions for optimal minimum density dimensions and gradually improves the solutions using an objective function [79]. At any point in time a fixed number of p-dimensions are present in the system and each p-dimension is a population. The total number of p-dimensions present in the system is called population count. In this section we study the impact of population count. Since A-ODDS and Stream LOCI do not require this parameter, their performance remains unchanged when the population count changes.

2.1.2.6.1. Precision

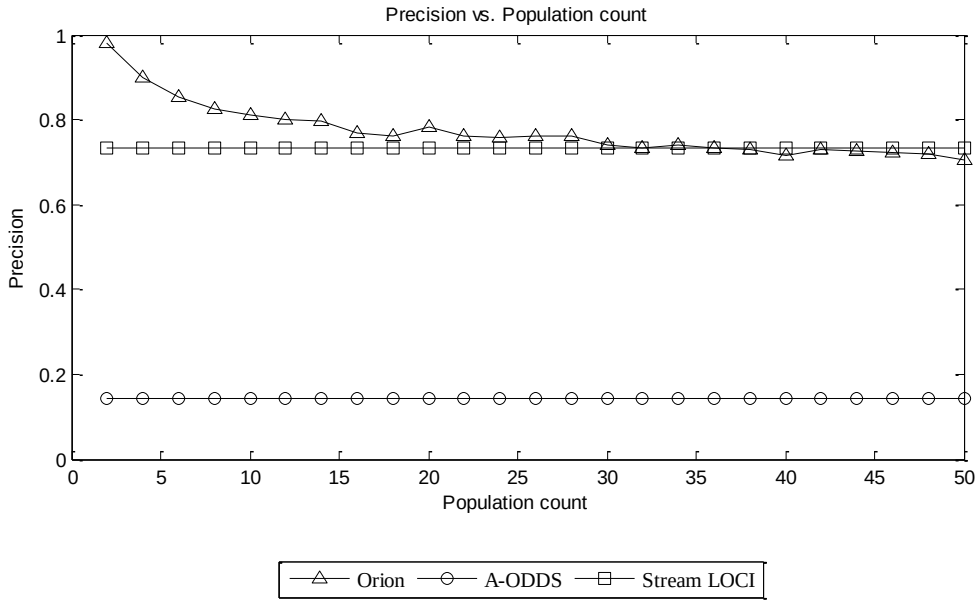

Figure 51. Impact of population count on precision for the irrigation data

Figures 51 and 52 show the impact of the population count on the precision. Interestingly, the precision of Orion decreases with the increase of the population count (Figure 51). This is a very counter-intuitive result. Since for many evolutionary algorithms, accuracy increases with the increase of population count, as the number of population increases Orion would more likely find an appropriate p-dimension that reveals the outlier-ness of the data point under investigation. However, as the population count increases Orion not only reveals the outlier-ness of the outlier, but also reveals the outlier-ness of the inliers. So, if an inlier is a little far from the other data points, Orion can easily find the p-dimension for that data point and identify it as an outlier. Thus the precision starts to decrease a little bit at the beginning and eventually reaches a stable value.

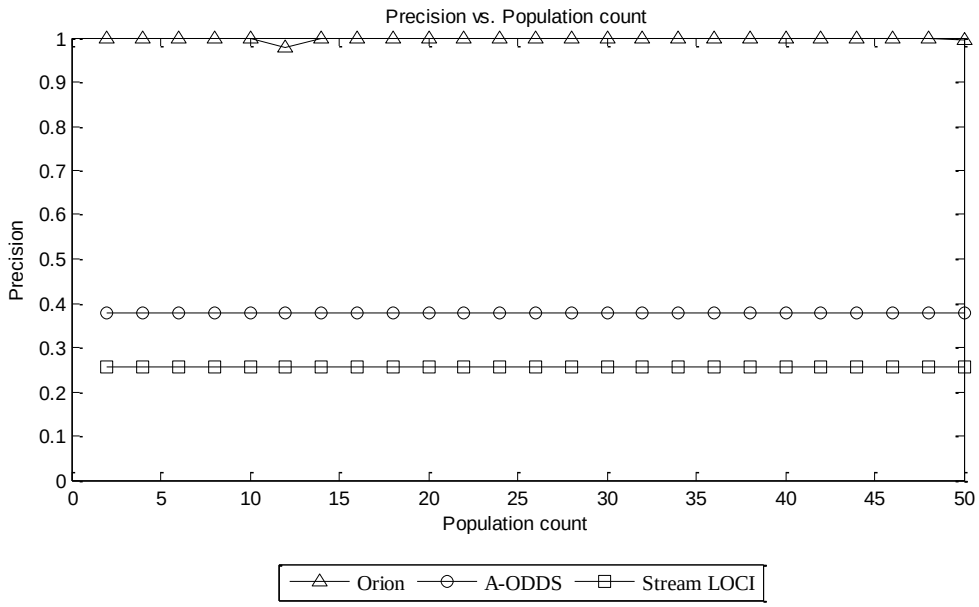

Figure 52. Impact of population count on precision for the synthetic data

However, Figure 52 shows that none of the algorithms is sensitive to the population count in terms of precision for the synthetic data. This is because the synthetic data has less noise compared to the irrigation data, there exists no p-dimension that reveals any inlier as an outlier and thus the precision remains constant for the synthetic dataset.

2.1.2.6.2. Recall

The impact on recall with respect to the change of population count is very intuitive (Figures 53 and 54). As the number of populations increases, Orion is more likely to find an appropriate p-dimension for outliers. Therefore, the recall increases with the increase of population count. This relationship is consistent across the datasets, hence we see the same result for both the irrigation data (Figure 53) and synthetic data (Figure 54).

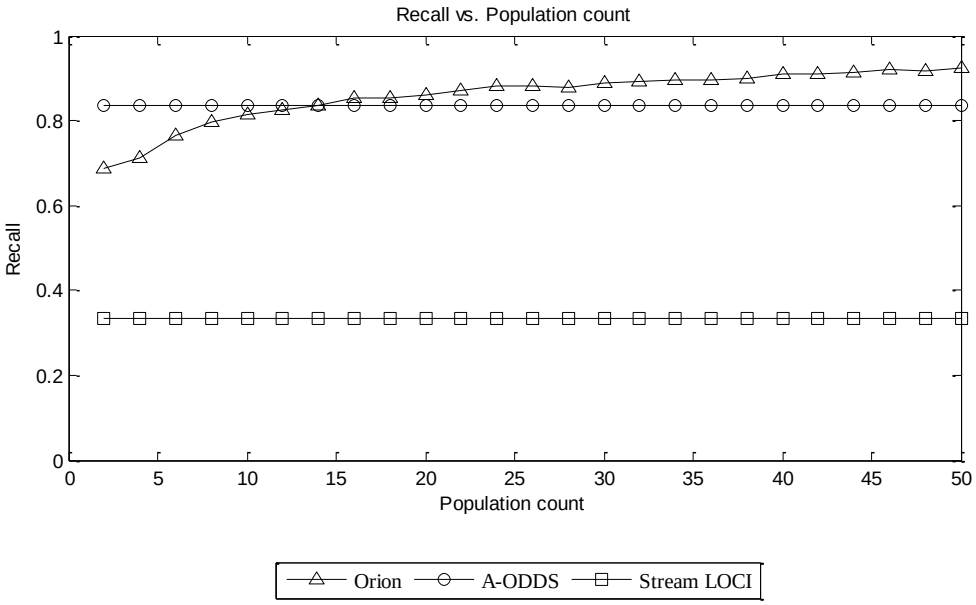

Figure 53. Impact of population count on recall for the irrigation data

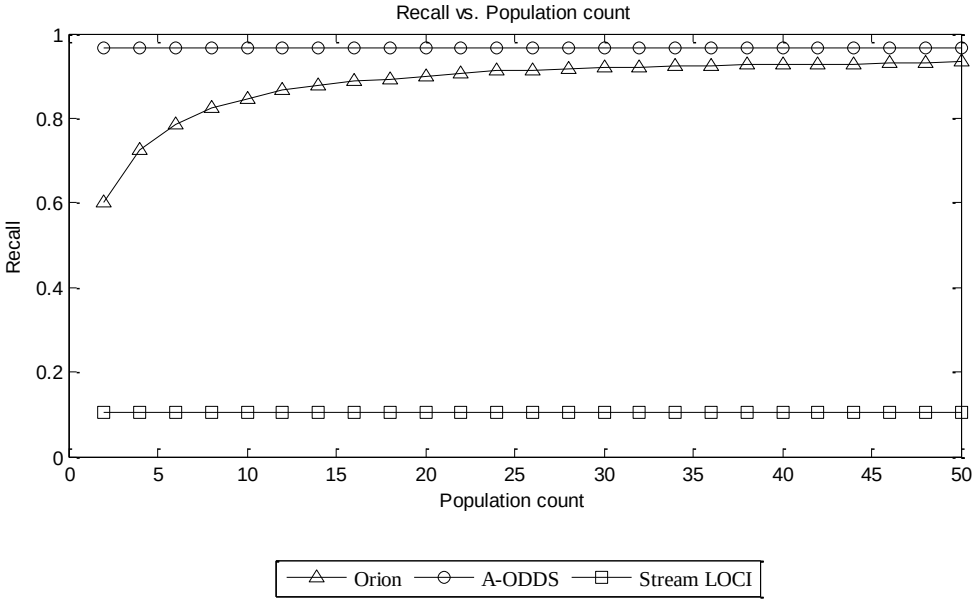

Figure 54. Impact of population count on recall for the synthetic data

2.1.2.6.3. Jaccard Coefficient

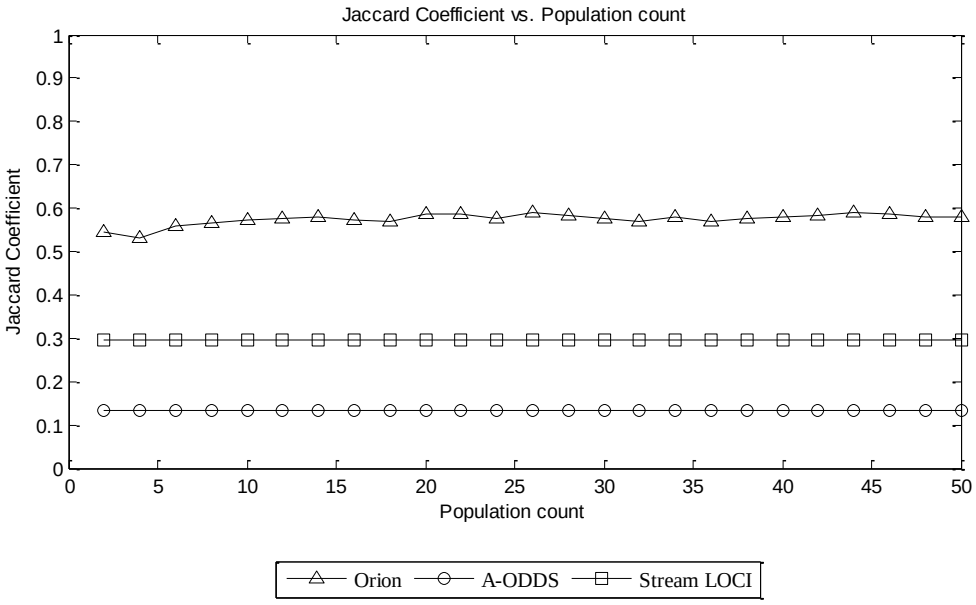

Figure 55. Impact of population count on Jaccard Coefficient for the irrigation data

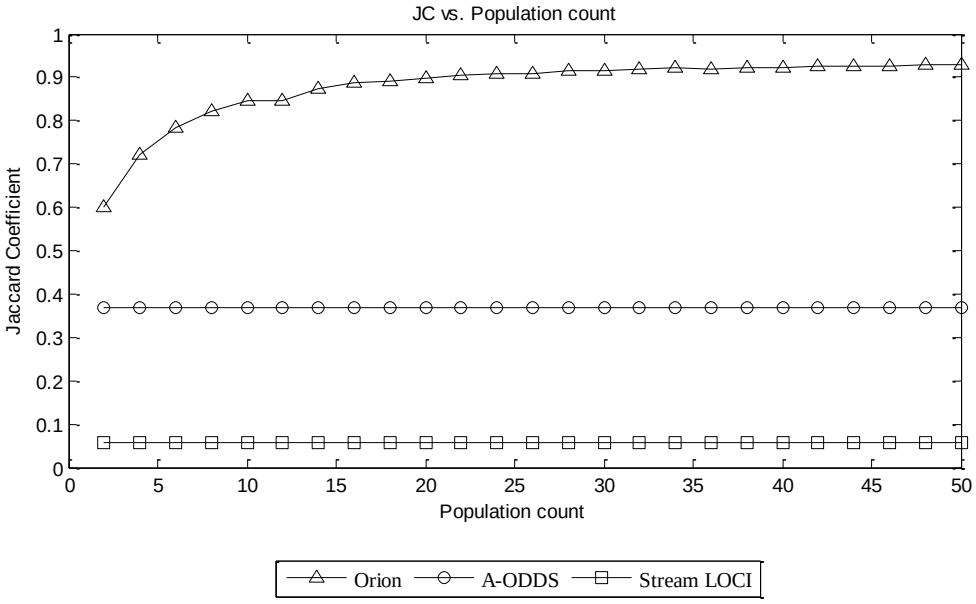

Figure 56. Impact of population count on Jaccard Coefficient for the synthetic data

Since the precision of Orion decreases a little bit and its recall increases a little bit with the increase of the population count, the JC of Orion remains constant with respect to the change of population count (Figure 55). However, the JC of Orion for the synthetic data increases logarithmically with the increase of the population count (Figure 56). This is because the precision of Orion is constant and the recall increases with respect to the increase of the population count, thus the JC increases as well.

2.1.2.6.4. Execution Time

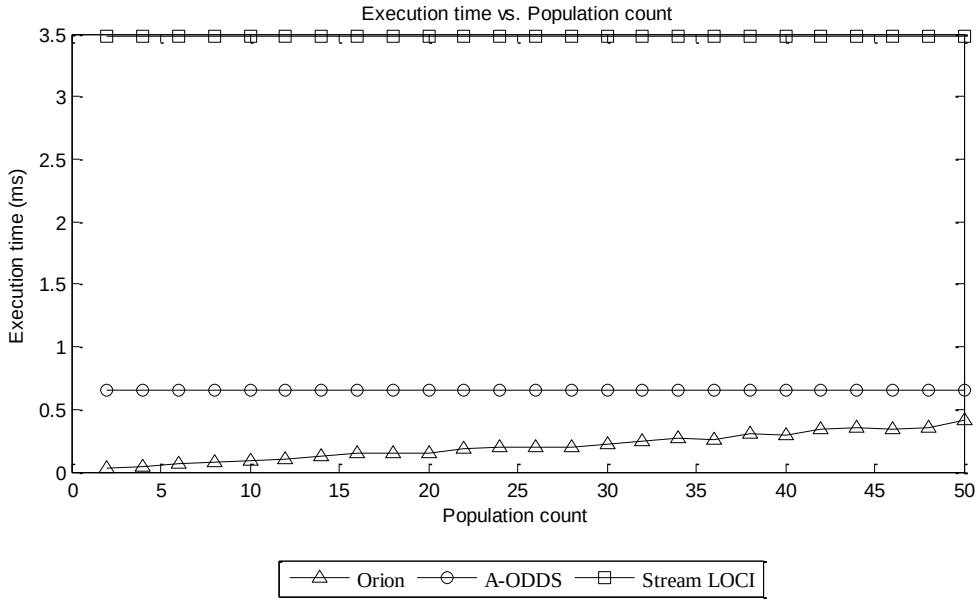

Figure 57. Impact of population count on execution time for the irrigation data

The execution time of Orion increases linearly with the linear increase of the population count (Figures 57 and 58). This is because as the number of populations increases, Orion has to compute the neighbor density for each p-dimension which takes a constant time and update each p-dimension as well, which also takes a similar amount of time. Thus overall the complexity increases linearly with the increase of the population count.

Hence the execution time also increases linearly with the increase of the population count. This is a trend that is consistent across the datasets (Figures 57 and 58).

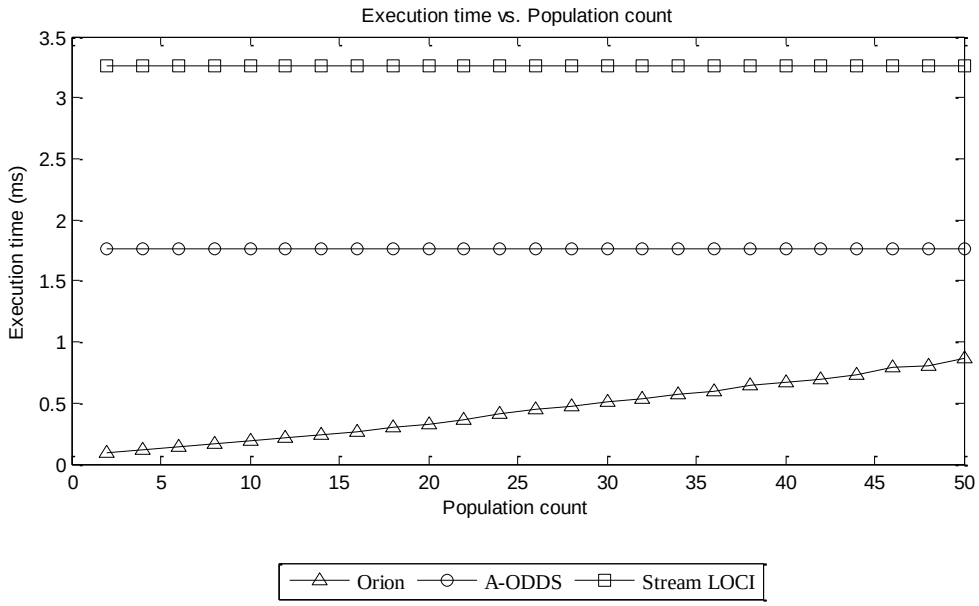

Figure 58. Impact of population count on execution time for the synthetic data

2.1.2.7. Impact of Bin Count

We use a binned implementation for our data density function. Moreover, we argue that more than 90% percent of the data points are within 6 standard deviations and hence it is sufficient to create 400 hundred bins for 6 standard deviations for any p-dimension. However, as the heuristic of 400 hundred bins for 6 standard deviation dispersion produces great result [85]; we vary the value of the number of bins and study the impact of the bin count. This section presents the study of bin count for the irrigation and synthetic data. Since A-ODDS and Stream LOCI do not require this parameter, they remain unchanged throughout the experiments.

2.1.2.7.1. Precision

Figures 59 and 60 show the impact of bin count on precision. If the bin count is small, the accuracy of the data density function is poor. The accuracy of the data density function increases with the increase of number of bins up to 400 hundred bins. In our case we see that the precision decreases a little bit (Figure 59) with the increase of bin count for the irrigation data and remains the same for the synthetic data (Figure 60). However, the change of precision with respect to bin count for the irrigation data is very insignificant. For a small bin count, the proposed data density function becomes less accurate and some inliers that are a little bit apart from other data points share the same bins; thus they are not identified as outliers. As the bin count increases and the data density function becomes more accurate, those inliers appear like outliers and hence Orion misclassified them as outliers.

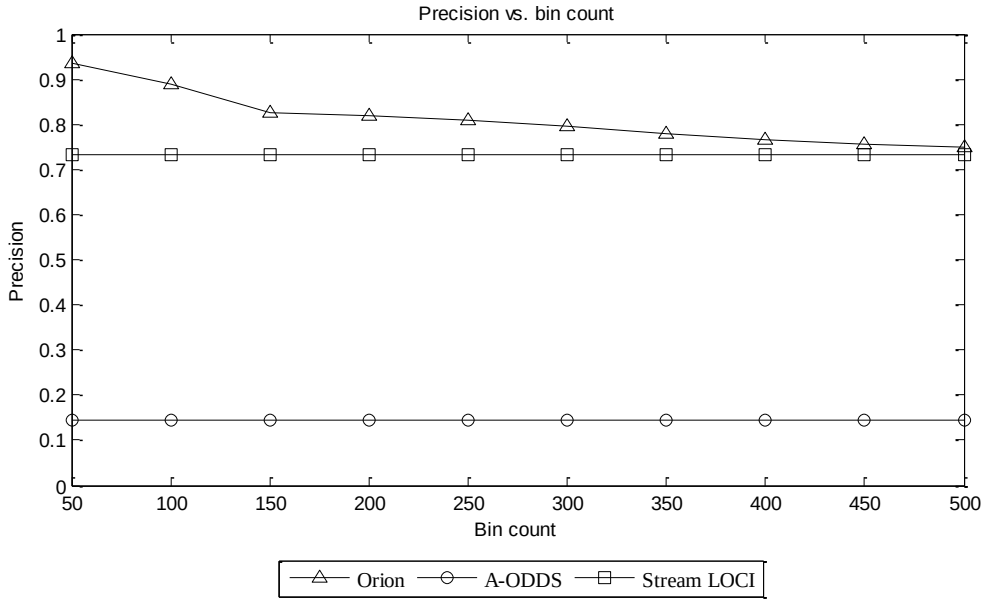

Figure 59. Impact of bin count on precision for the irrigation data

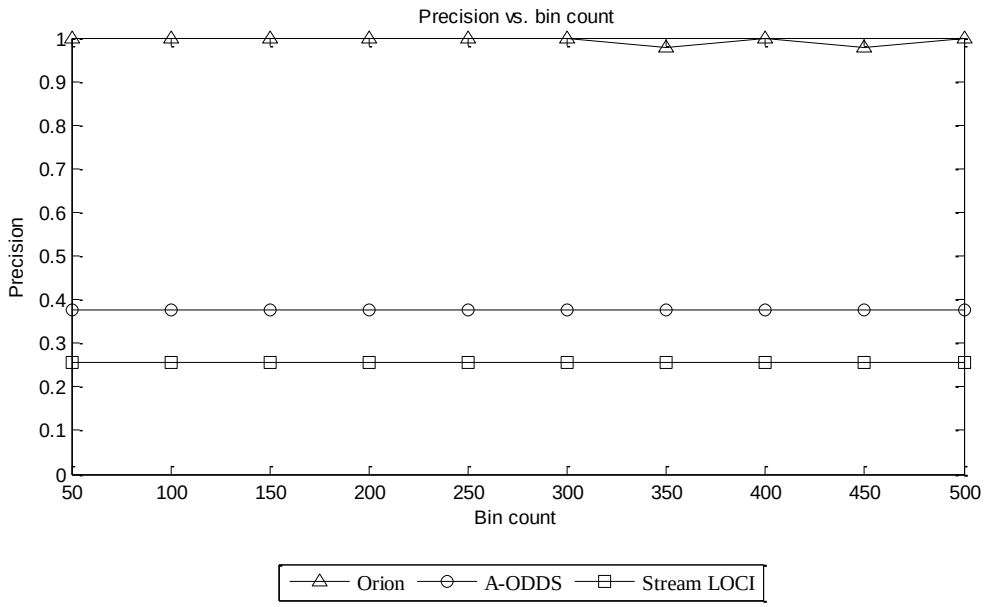

Figure 60. Impact of bin count on precision for the synthetic data

2.1.2.7.2. Recall

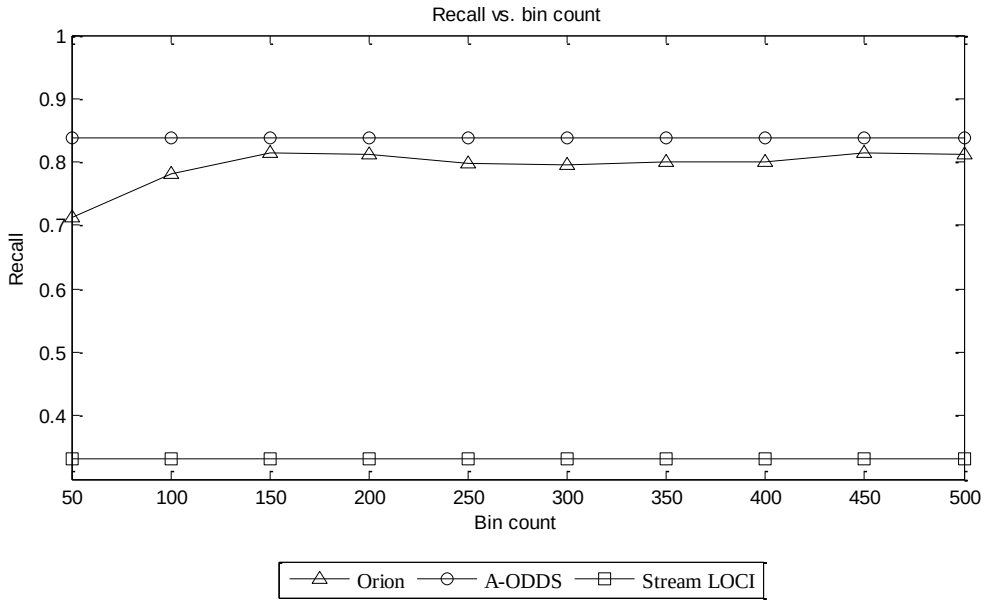

Figure 61. Impact of bin count on recall for the irrigation data

The recall of Orion increases with the increase of bin count. If the number of bins increases, the data density function becomes more accurate and outliers fail to hide

themselves in the mass of inliers. In case of a small bin count, the outliers easily share the same bin with the inliers and hide themselves. So the recall increases with the increase of bin count. This trend is consistent across the datasets (Figures 61 and 62).

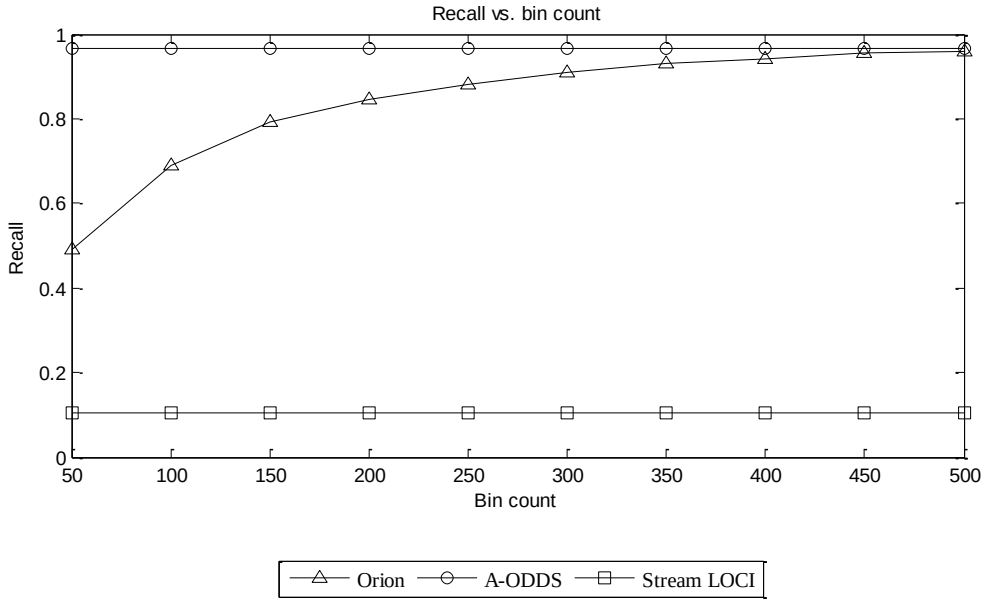

Figure 62. Impact of bin count on recall for the synthetic data

2.1.2.7.3. Jaccard Coefficient

The impact of bin count on the JC for the irrigation data is portrayed in Figure 63. According to Figure 63, bin count has no impact on JC. This is because the precision of Orion decreases a little bit and its recall increases a little bit with the increase of bin count. These two balance each other out and the JC remains constant with respect to the change of bin count. However the JC of Orion for the synthetic data increases with the increase of bin count (Figure 64). This is because the precision of Orion for the synthetic data does not show any variation, while the recall increases with the increase of bin count. Thus JC increases with the increase of bin count as well.

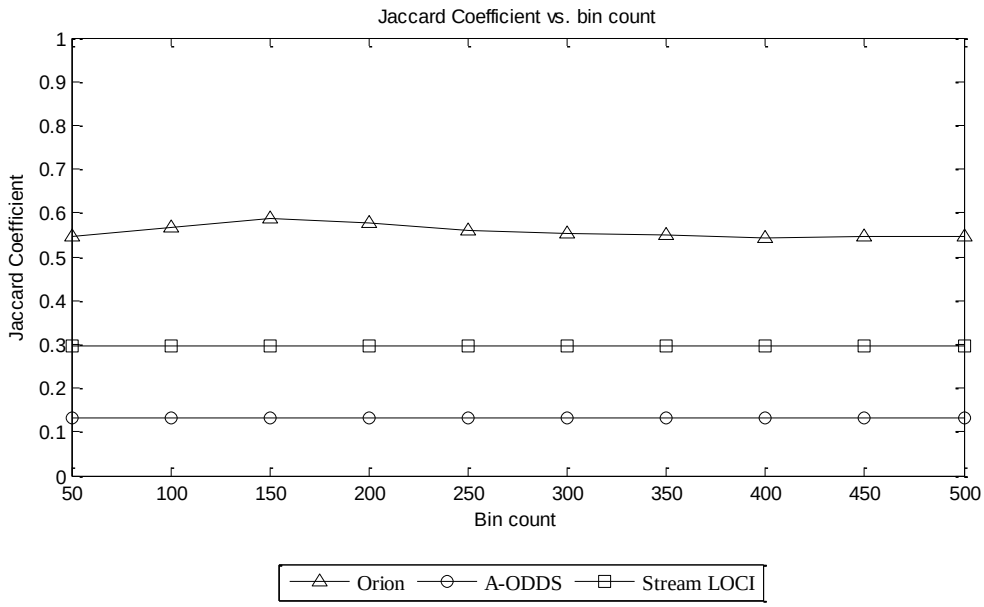

Figure 63. Impact of bin count on Jaccard Coefficient for the irrigation data

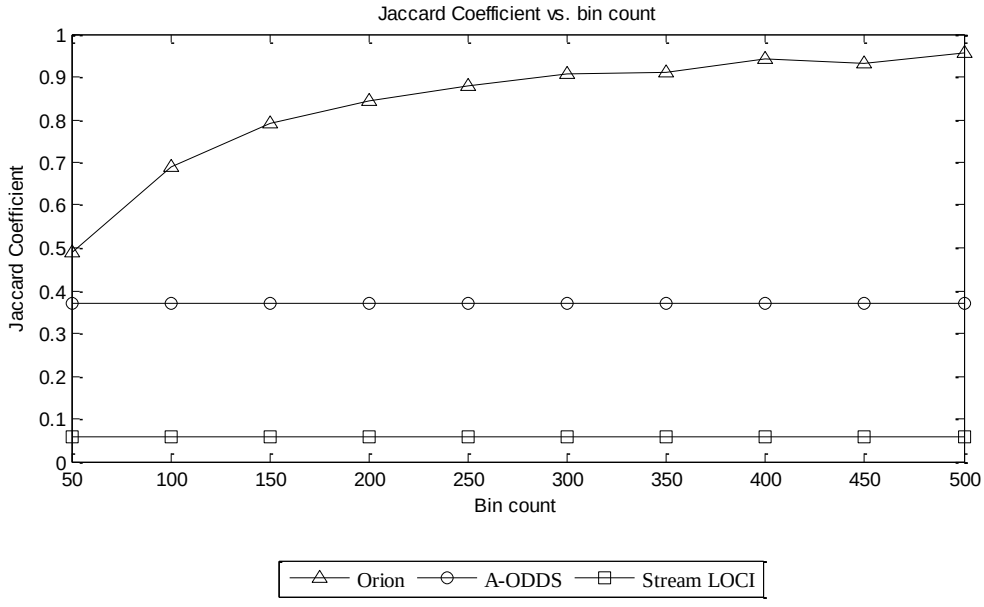

Figure 64. Impact of bin count on Jaccard Coefficient for the synthetic data

2.1.2.7.4. Execution Time

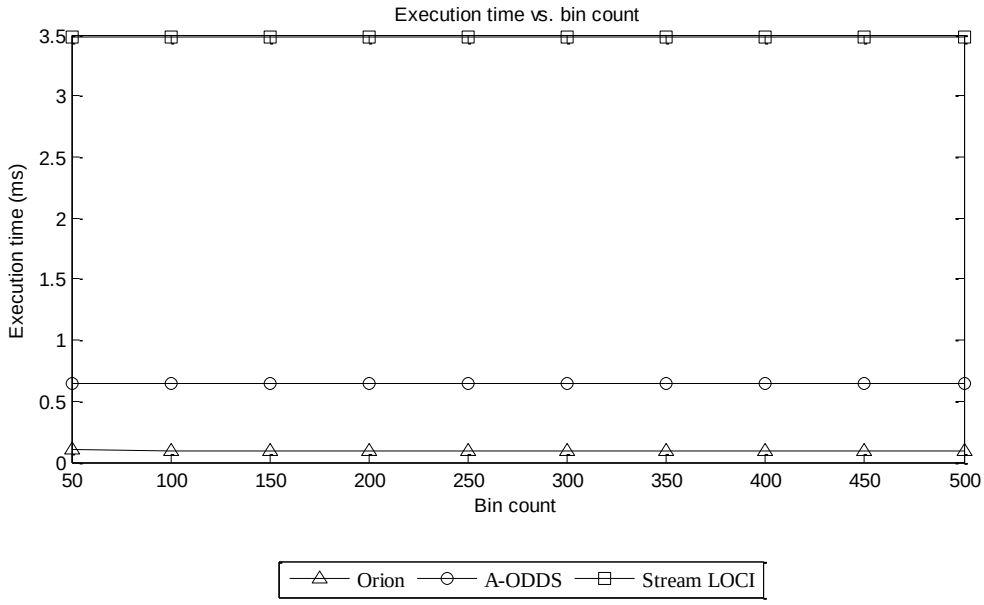

Figure 65. Impact of bin count on execution time for the irrigation data

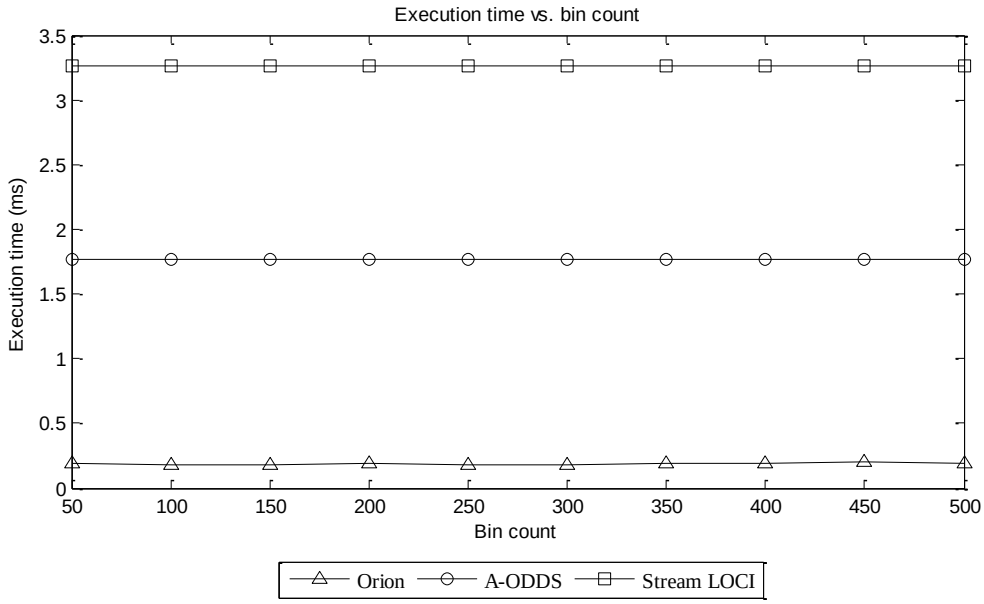

Figure 66. Impact of bin count on execution time for the synthetic data

Figures 65 and 66 show the impact of bin count on execution time. Since our neighbor density computation and k-distance computation times are almost similar regardless of

the number of bins, the execution time of Orion remains unchanged when the bin count changes.

2.1.2.8. Impact of Bootstrapping Size

Orion uses a set of data points to learn the forgetting factor and initializes the cluster centers. Hence we perform a study to show how the number of bootstrapping rounds impacts the performance of Orion.

2.1.2.8.1. Precision

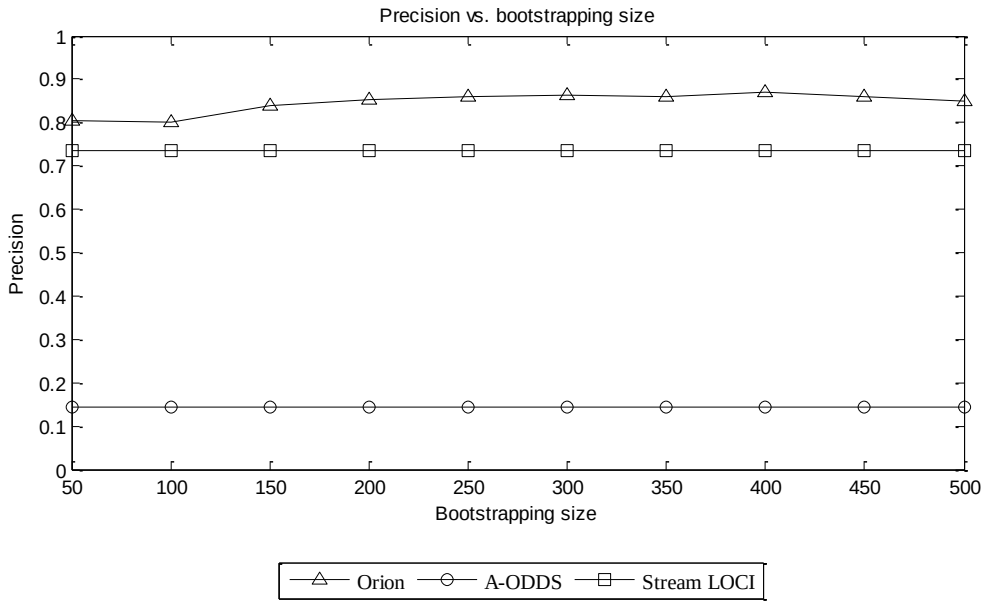

Figure 67. Impact of bootstrapping size on precision for the irrigation data

The precision of Orion almost does not change regardless of the number of bootstrapping rounds (Figures 67 and 68). This is because Orion can learn the forgetting factor from a small set of data points, adding more data points to the bootstrapping rounds does not help Orion to perform better. Hence we observe an almost constant precision with respect to the change of the bootstrapping size. This is consistent for both

the datasets (Figures 67 and 68). Stream LOCI and A-ODDS remain unchanged since they do not use bootstrapping data points at all.

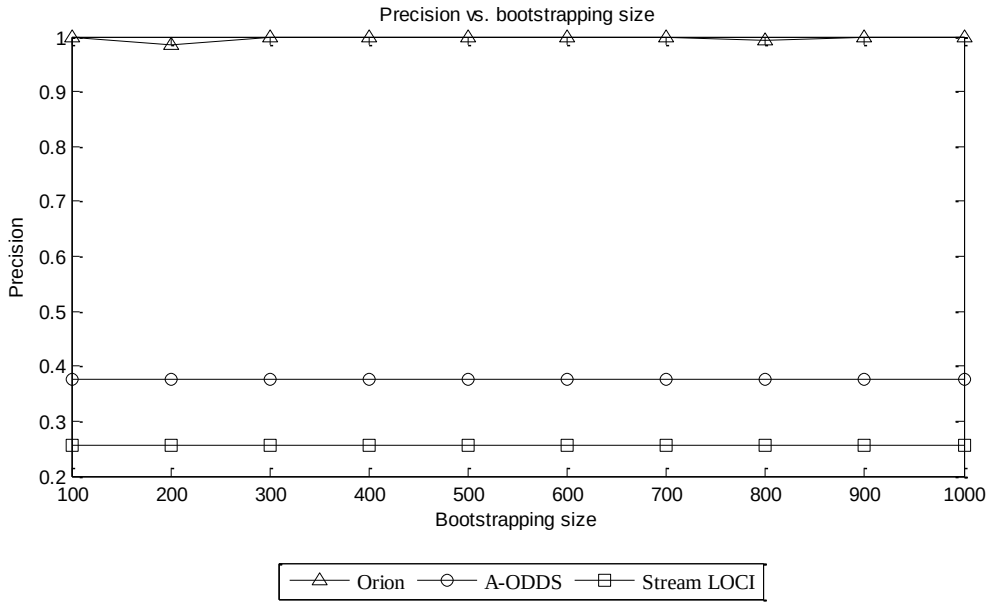

Figure 68. Impact of bootstrapping size on precision for the synthetic data

2.1.2.8.2. Recall

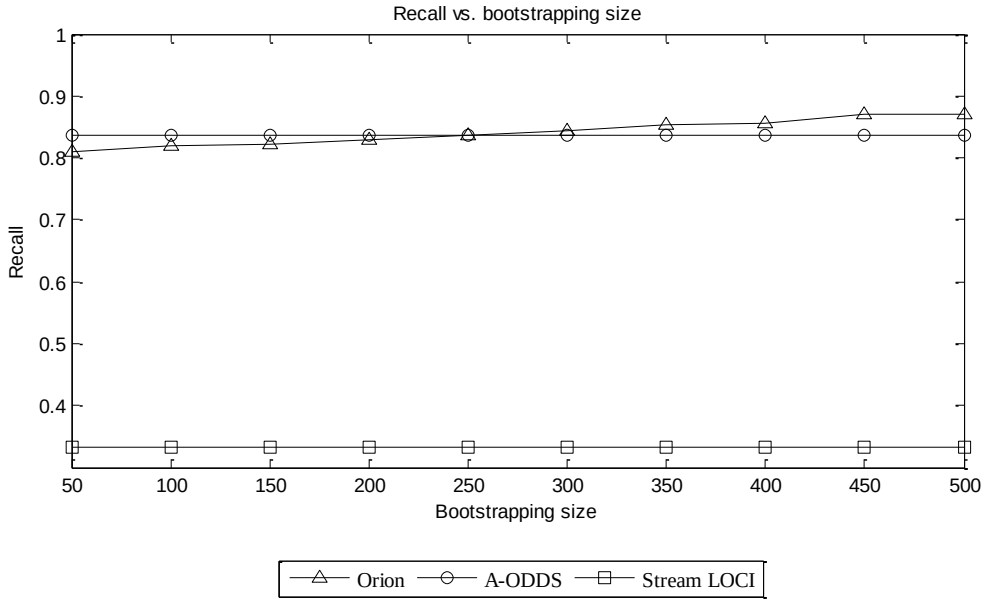

Figure 69. Impact of bootstrapping size on recall for the irrigation data

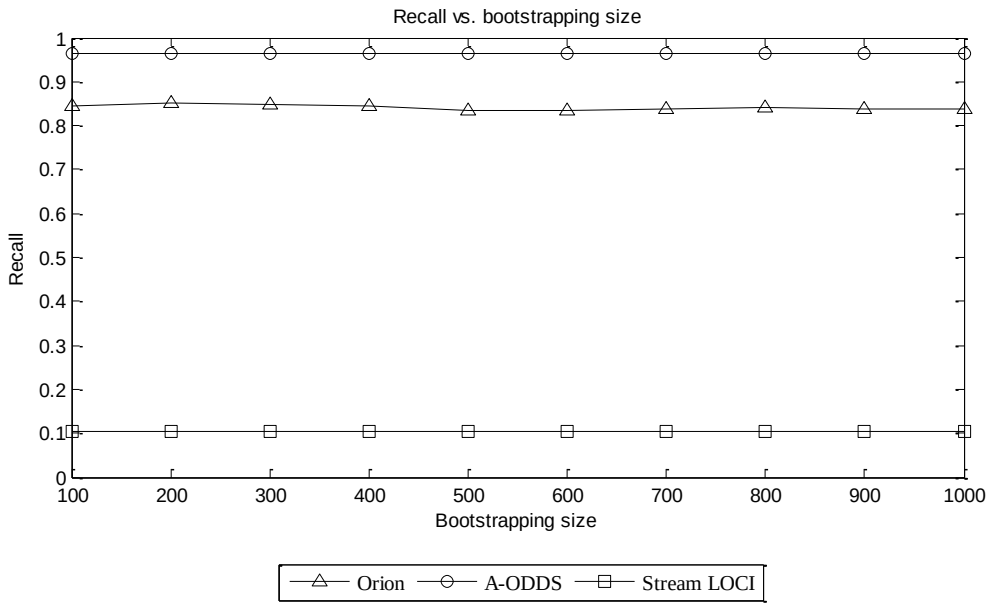

Figure 70. Impact of bootstrapping size on recall for the synthetic data

We observe the similar pattern in terms of recall as well (Figures 69 and 70). The recall of Orion is insensitive to the change of the bootstrapping size.

2.1.2.8.3. Jaccard Coefficient

Since the precision and recall remain unchanged, the Jaccard coefficient remains unchanged as well (Figures 71 and 72) when the bootstrapping size changes. This is useful since the user can choose a small bootstrapping size and detect the outlier-ness of all data points after that.

2.1.2.8.4. Execution Time

Finally like with other performance metrics, the execution time is not impacted by the bootstrapping size (Figures 73 and 74). Since we measure the execution time as the time to detect the outlier-ness of a data point which is independent of the bootstrapping size, we see no impact of the bootstrapping size on the execution time.

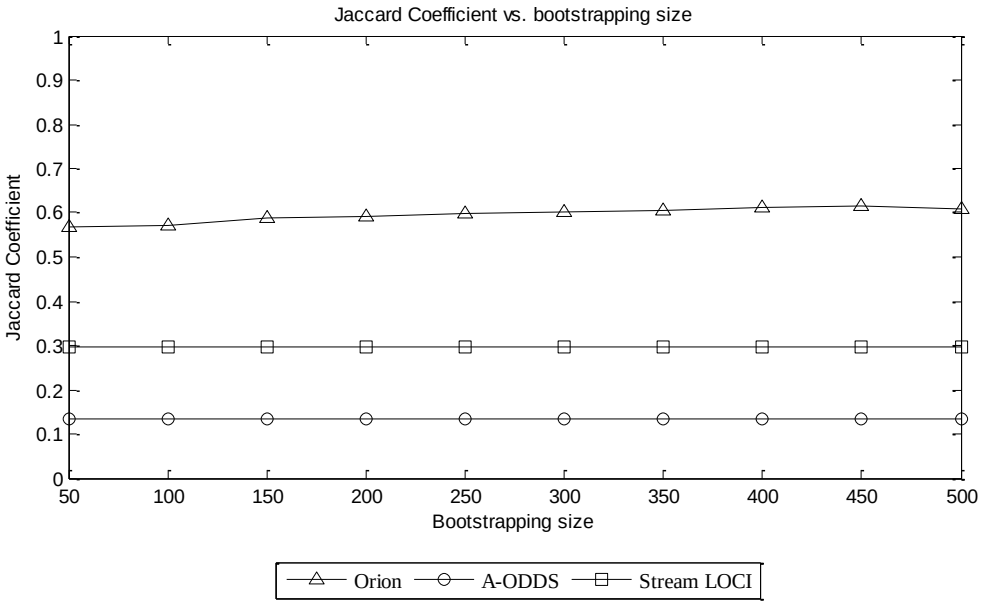

Figure 71. Impact of bootstrapping size on Jaccard Coefficient for the irrigation data

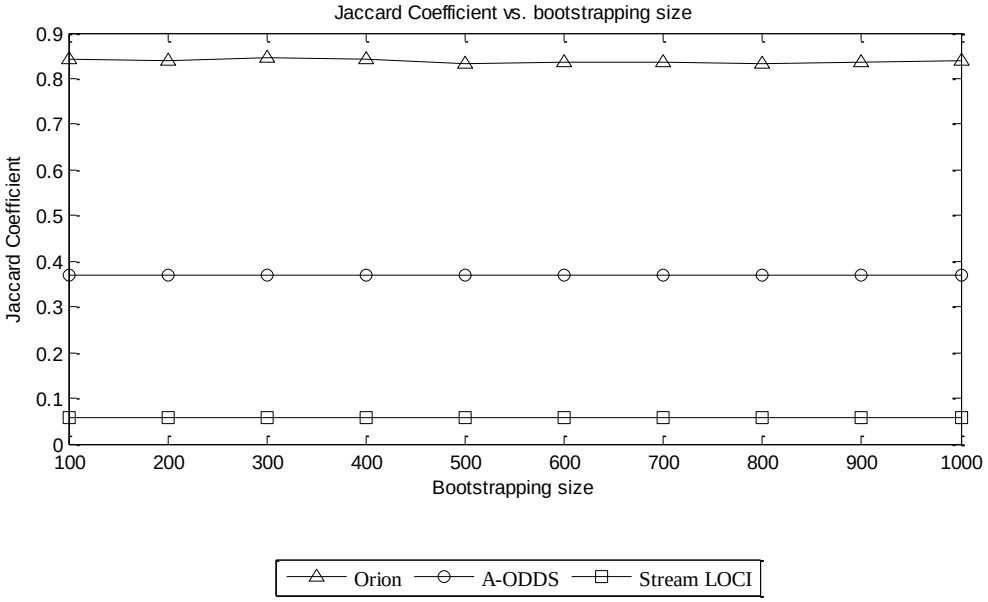

Figure 72. Impact of bootstrapping size on Jaccard Coefficient for the synthetic data

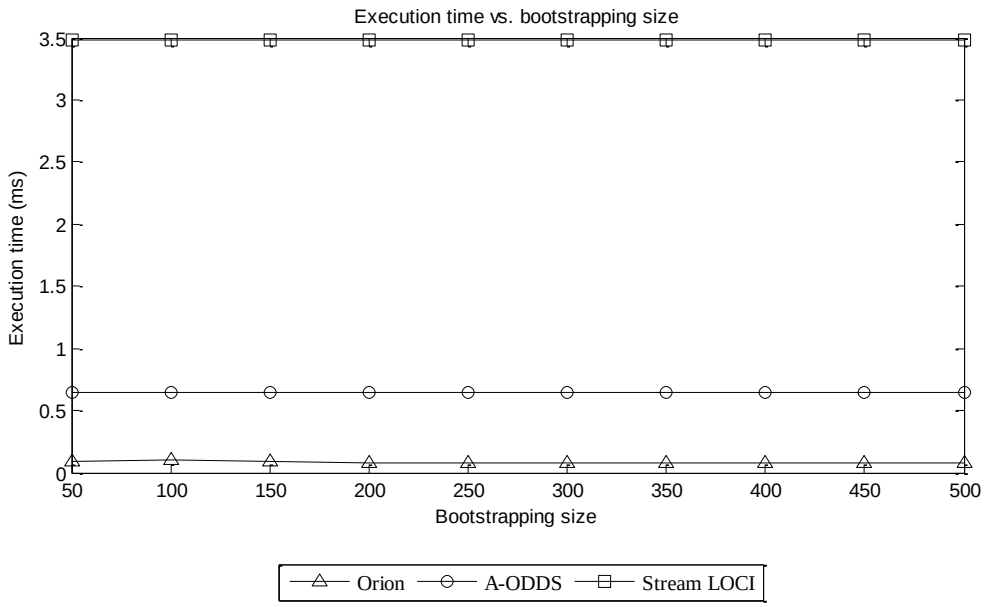

Figure 73. Impact of bootstrapping size on execution time for the irrigation data

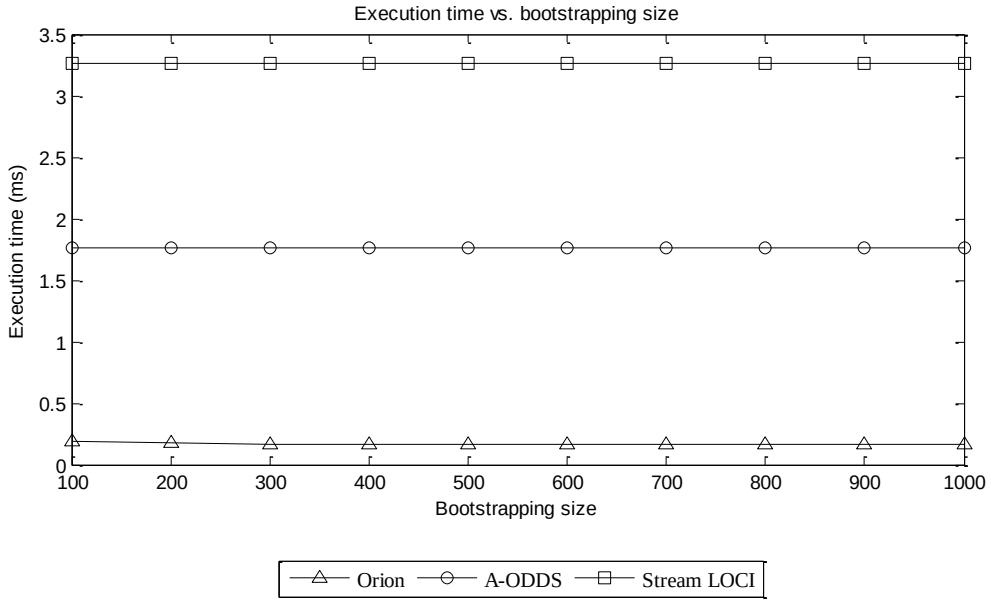

Figure 74. Impact of bootstrapping size on execution time for the synthetic data

2.1.2.9. Impact of Number of Data Rounds

A data stream is an infinite sequence of data points; therefore it is important for any outlier detection technique to perform well for a large number of data points; otherwise

it is hard to predict the behavior of an algorithm in case of data streams. This study reveals the impacts of number of data rounds on the performance of all three algorithms.

2.1.2.9.1. Precision

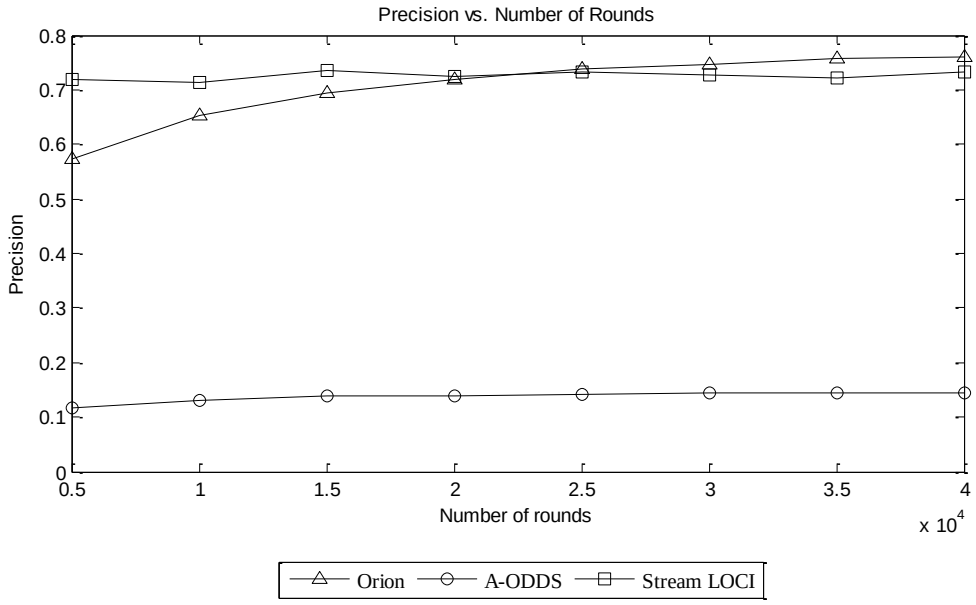

Figure 75. Impact of number of data rounds on precision for the irrigation data

Figures 75 and 76 show the impacts of number of data rounds on the precision of all three algorithms for the irrigation and synthetic datasets. The precision of Orion increases as the number of data rounds increases. This is because at the beginning, the co-clustering algorithm does not know where to draw the line between high, average and low neighbor density and k-distance. Hence at the beginning Orion identifies some inliers as outliers. But as time passes more data points come in, Orion learns the centers of the clusters in our co-clustering algorithm and stops misclassifying inliers as outliers. Thus the precision increases as the number of data points increases.

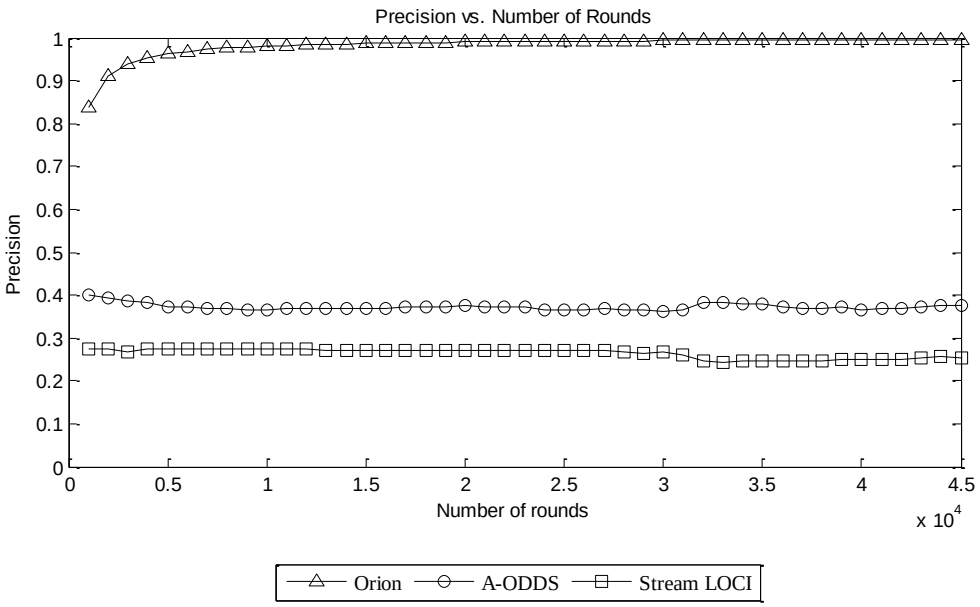

Figure 76. Impact of number of data rounds on precision for the synthetic data

The precision of A-ODDS and Stream LOCI remains unchanged irrespective of the number of data points. This is because none of them relies on the old data points very much. Stream LOCI uses a sliding window and A-ODDS uses the forgetting factor to decay the weights of older data points compared to newer data points. Thus they are insensitive to the number of data points.

2.1.2.9.2. Recall

The recall of Orion for the irrigation data changes a little bit with the change of the number of data rounds (Figure 77). At the beginning when the number of data points is very small, the recall of Orion changes a little bit, but it quickly stabilizes. This is only for the irrigation data since it has a lot of noise in it, and therefore Orion takes some time to stabilize the co-clusters. Once the co-clusters are fixed, the impact of number of data points is not visible for any of the performance.

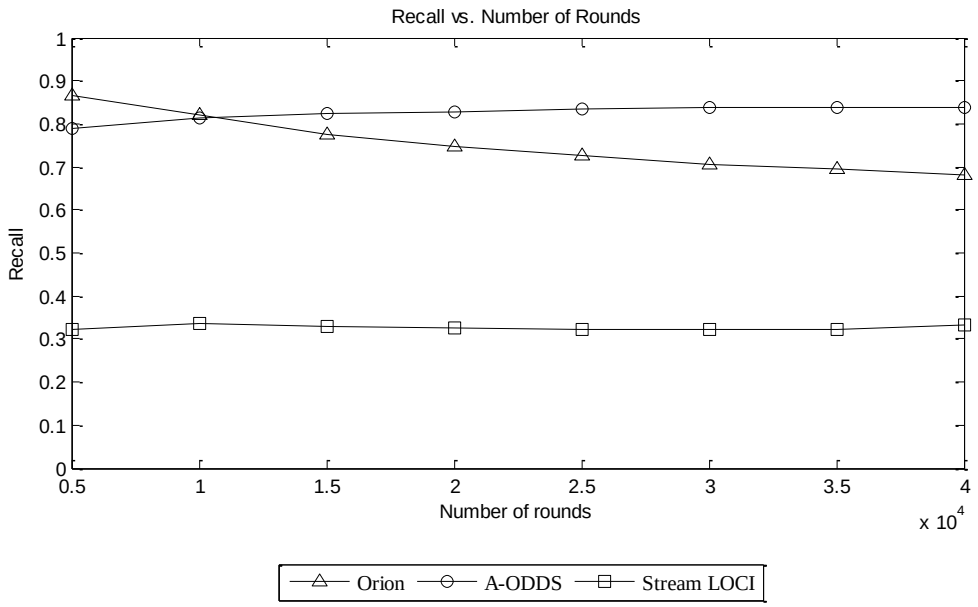

Figure 77. Impact of number of data rounds on recall for the irrigation data

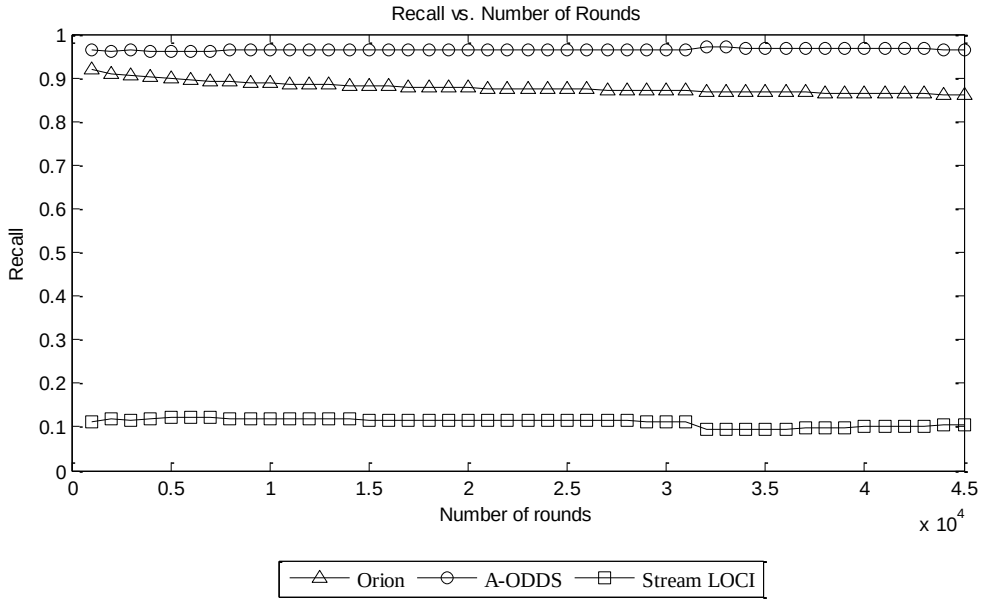

Figure 78. Impact of number of data rounds on recall for the synthetic data

Moreover, the recall of Orion for the synthetic dataset remains the same with respect to the change of the number of data points (Figure 78). This is because Orion can quickly learn the cluster centers of the co-clustering step. As Orion learns the cluster centers , it

does not misclassify many inliers as outliers. Thus, the recall of Orion shows no sensitivity with respect to change of number of rounds for the synthetic dataset.

2.1.2.9.3. Jaccard Coefficient

Overall the number of data points has no impact on the JCs of all three algorithms (Figures 79 and 80). Thus the performance of all three algorithms does not change when the number of data points changes.

2.1.2.9.4. Execution Time

The execution times of all three algorithms remain constant with respect to the change of the number of data rounds (Figures 81 and 82); hence, it can be concluded that all three algorithms are well suited for infinite data streams.

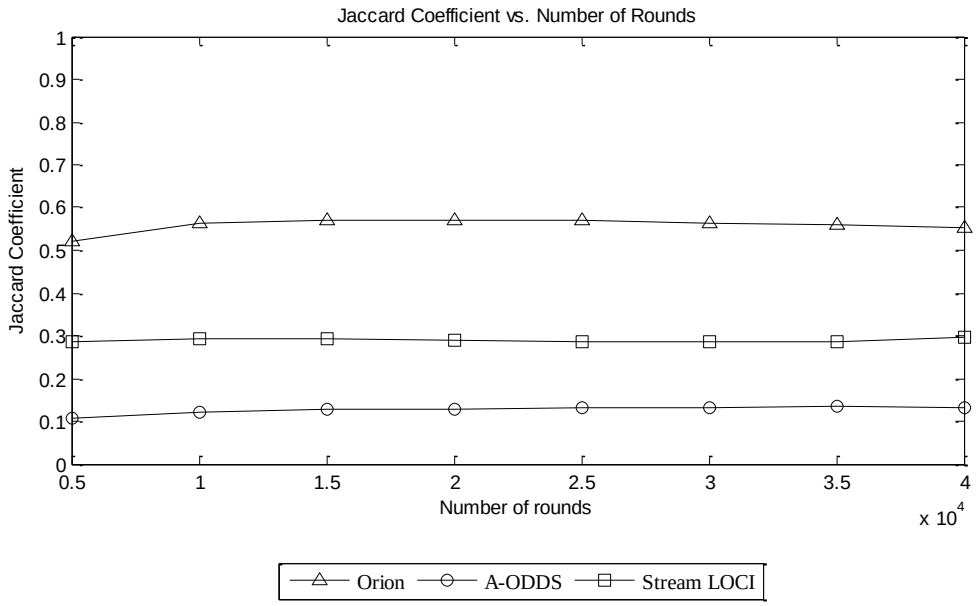

Figure 79. Impact of number of data rounds on Jaccard Coefficient for the irrigation data

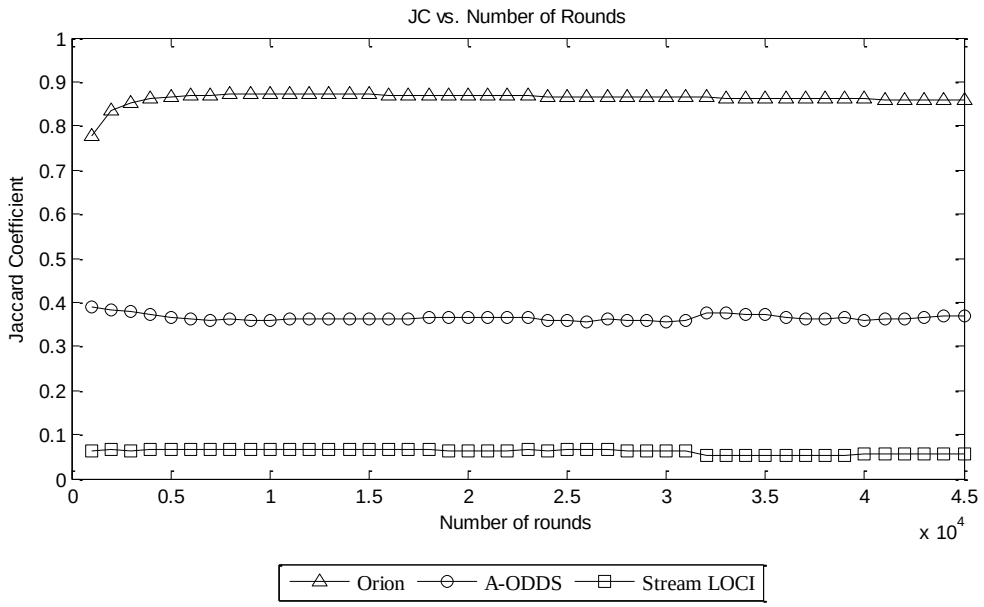

Figure 80. Impact of number of data rounds on Jaccard Coefficient for the synthetic data

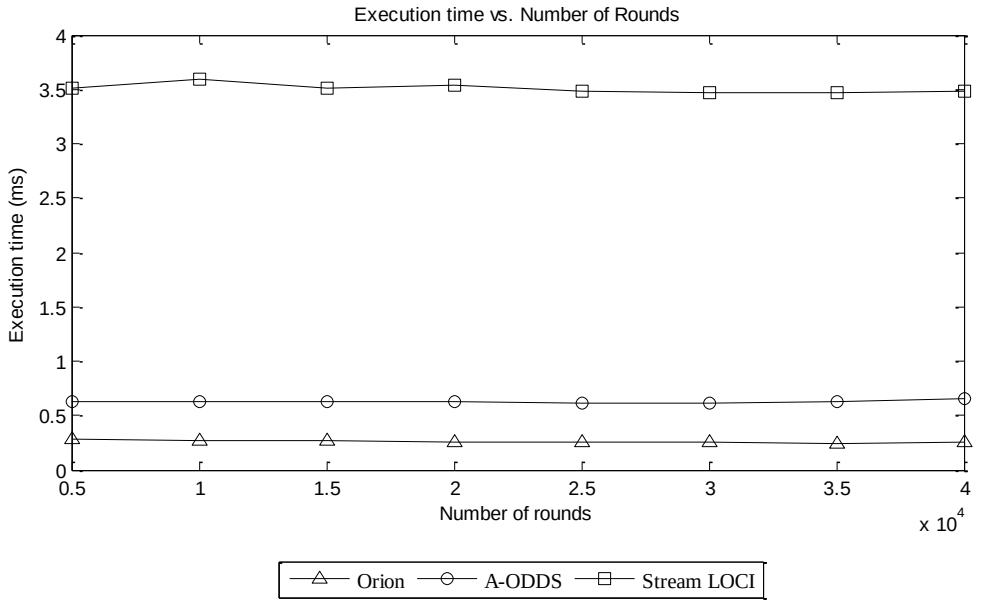

Figure 81. Impact of number of data rounds on execution time for the irrigation data

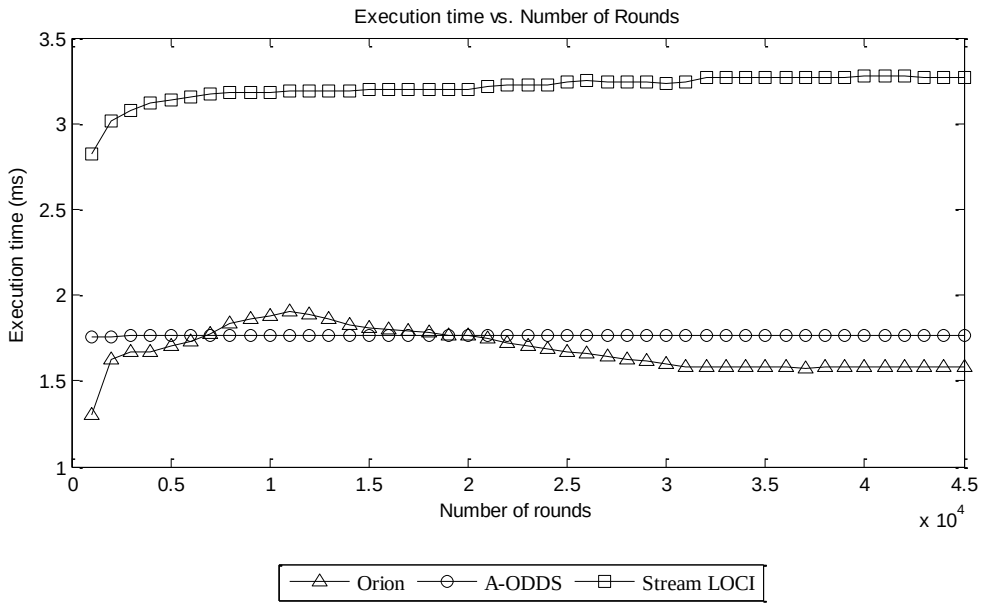

Figure 82. Impact of number of data rounds on execution time for the synthetic data

2.1.2.10. Impact of Concept Drift

In this section, we present the study of impacts of concept drift on the accuracy and execution time of Orion, A-ODDS and Stream LOCI. Tsymbal argues that there are two types of concept drifts that occur in real world [99]: gradual concept drift and abrupt concept drift. Gradual concept drift refers to the problem where the value of an item gradually changes over time such as machine wear and tear, environment temperature, etc. Abrupt concept drift refers to the problem where the value of an item changes suddenly. Not only that, the distribution family of the value may changes suddenly as well such as credit card transactions and salary of an employee [99]. Our synthetic dataset contains a harmonic component which gradually changes the attribute values. Hence, our synthetic dataset is used as a representative dataset for gradual concept drift. We create a separate dataset, where data points are distributed from a randomly chosen distribution type among the Gaussian distribution, Triangular distribution, Beta

distribution, Exponential distribution and Uniform distribution. The distribution of the data points in this dataset does not changes over time and it has no harmonic component to simulate gradual concept drift. We call this dataset a no concept drift dataset.

We also synthesize a third dataset that simulates abrupt concept drift. In this dataset, the data points starts with an initial distribution. After a certain time (randomly chosen) the distribution of data points changes from its previous distribution to a new distribution randomly chosen from the Gaussian distribution, Triangular distribution, Beta distribution, Exponential distribution and Uniform distribution. This random change of distributions happens at a randomly chosen time interval. This dataset is a representative dataset for abrupt concept drift. Thus, in this experiment we execute all three algorithms for these three datasets and record accuracy (precision, recall, Jaccard Coefficient) and execution time.

2.1.2.10.1. Precision

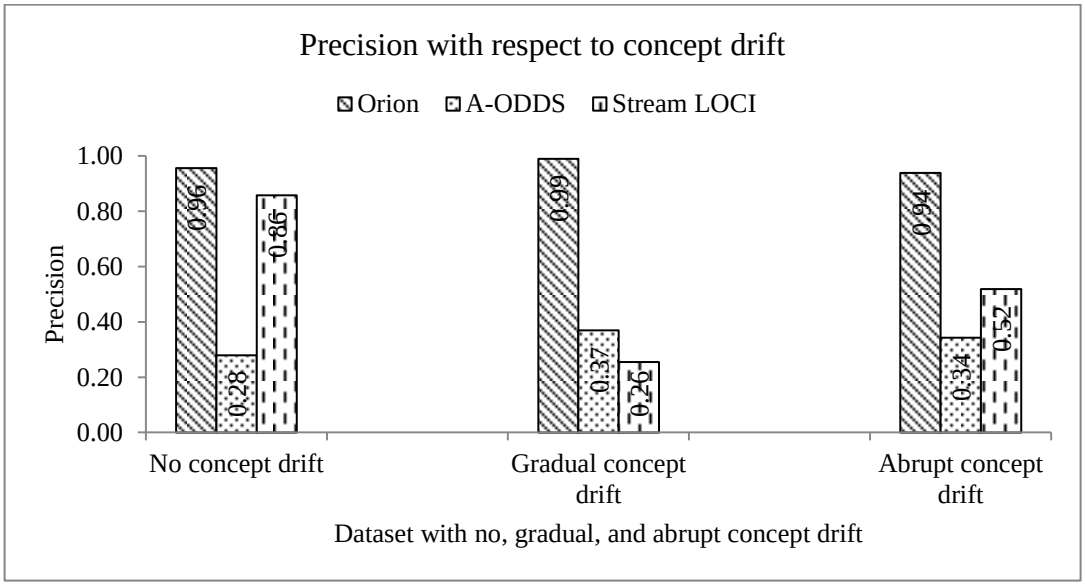

Figure 83. Impact of Concept drift on precision for the synthetic data

Figure 83 presents the impacts of concept drift on the precision of Orion, A-ODDS and Stream LOCI. One of the objective of Orion is to be able to detect outliers despite the presence of concept drifts. The precision of Orion for the no concept drift data is the same as its precision for the abrupt concept drift data. In the no concept drift data, it is easier for Orion to find an appropriate p-dimension. Since there is no concept drift, the distribution of data never changes and hence, Orion can work with the same set of p-dimensions. The precision of Orion for the abrupt concept drift data is the same as its precision for the no concept drift data. Thus, we can conclude that abrupt concept drift has no impact on the precision of Orion. This is because once the distribution of the data points changes, Orion adaptively changes its p-dimensions so that the existing set of p-dimensions can reveal the outlier-ness of the data points. Hence, Orion can be adaptive well with the sudden change of concepts. Finally, the precision of Orion for the gradual concept drift data is a little bit better than its precision for the no concept drift data and the abrupt concept drift data; but the increase of precision for the gradual concept drift data is not statistically significant. Thus, we can conclude the precision of Orion is not affected by any type of concept drift.

The precision of Stream LOCI is significantly affected by the concept drift. Stream LOCI uses a sliding window to capture the recent subset of the data points. Hence, if there is no concept drift, the sliding window always contains the data points from the same distribution. Therefore, Stream LOCI can separate outliers from inliers easily. In case of abrupt concept drift, the sliding window of Stream LOCI contains data points from more than one group. In that case, if one group is not significantly outnumbered by another, Stream LOCI can separate them and detect outliers from individual groups

(this is because it is a density based approach; it works with local neighborhood only). Once an abrupt concept drift occurs, the sliding window of Stream LOCI gradually fills with the data points from the new concept. At the beginning, the data points from the new concept appear as outliers with respect to the data points from the old distribution. Hence, Stream LOCI creates some false alarms. Thus the precision of Stream LOCI for the abrupt concept drift data is smaller than that for the no concept drift data. Stream LOCI performs worse in terms of precision for gradual concept drift. In case of gradual concept drift, the data points are continuously changing, but with a tiny amount. Thus, the sliding window of Stream LOCI is composed of the data points where each of them (data points) has little different data distribution compared to the other data points. Unfortunately, Stream LOCI cannot handle this kind of data and therefore, its precision is the lowest for the gradual concept drift data (Figure 83).

The precision of A-ODDS is a little high for gradual concept drift compared to the cases of no concept drift and abrupt concept drift. In a dataset with gradual concept drift, A-ODDS can successfully detect the local data points and identify the outliers [60]. For no concept drift data, the GDF and LDF of A-ODDS become the same and hence it only detects outliers on the global context. The precision of A-ODDS is higher for the abrupt concept drift data compared to the no concept drift data because in that case, GDF and LDF become different again and A-ODDS can then detect outliers from the global and local contexts. However, irrespective of the dataset, Orion shows its supremacy over Stream LOCI and A-ODDS in terms of precision.

2.1.2.10.2. Recall

Figure 84 shows the impact of concept drift on recall. The recall of Orion is the same for three cases: no, gradual and abrupt concept drift. Orion can detect an outlier if it contains an appropriate p-dimension that reveals the outlier-ness of the data point. In all three cases, Orion can readily find the appropriate p-dimensions for outlying instances that reveal their outlier nature. Since our evolutionary algorithm adaptively modifies the set of p-dimensions, Orion always have an appropriate p-dimension that can reveal an outlier. Therefore, the recall of Orion is not affected by concept drift; thus, our design goal, which is to design an appropriate outlier detection technique the performance of which is not affected by concept drift, is achieved. Moreover, Orion outperforms both Stream LOCI and A-ODDS for all types of concept drifts in terms of recall.

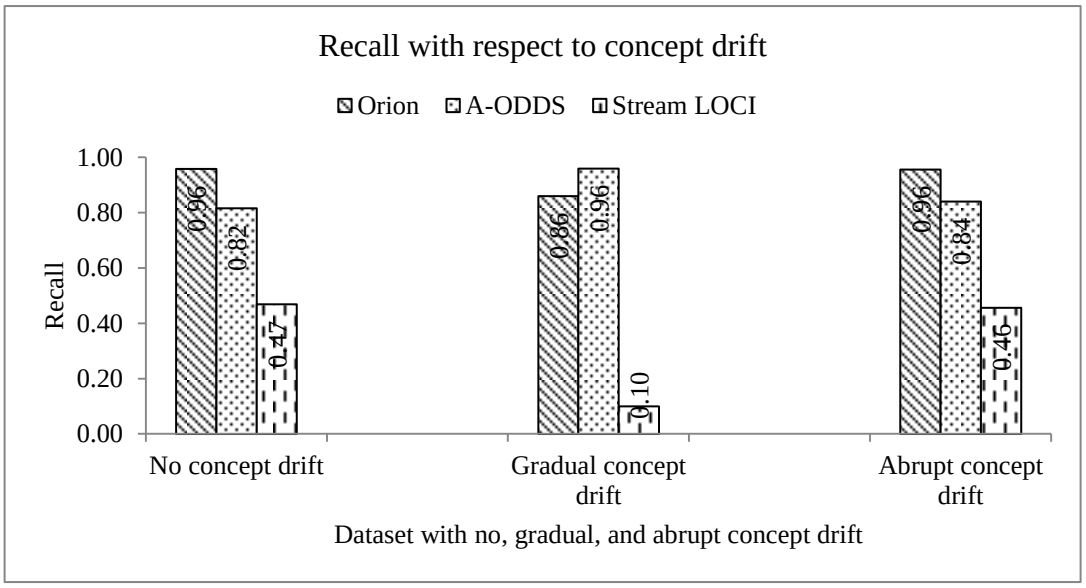

Figure 84. Impact of Concept drift on recall for the synthetic data

According to Figure 84, the recall of Stream LOCI is affected by concept drift. The recall of Stream LOCI is the lowest for the gradual concept drift data. As we have discussed before, the sliding window of Stream LOCI contains a set of data points, each

of which is slightly different from the rest. Therefore, Stream LOCI confuses inliers with outliers and cannot find outliers. The recall of A-ODDS for the gradual concept drift data is higher than its recall for the no concept drift data and the abrupt concept drift data. This is because once A-ODDS can identify the local context for LDF, it can easily identify the outliers that are non-conformist to the local data points.

2.1.2.10.3. Jaccard Coefficient

Figure 85 shows the accuracy in terms of Jaccard Coefficient of Orion, Stream LOCI and A-ODDS with respect to different types of concept drift data. Since the precision and recall of Orion are not affected by concept drift, JC is not affected by concept drift either. We have discussed our reasoning for such insensitivity of Orion in terms of accuracy for all types of concept drifts. This experiment confirms that Orion can successfully handle concept drift. Compared with the gradual and abrupt concept drift data, Stream LOCI has a higher precision and recall, and thus also has a higher JC, for the no concept drift data. The JC of A-ODDS is for gradual concept drift is higher than the JC of A-ODDS for no concept drift and abrupt concept drift (we have discussed our reasoning in previous two sections). Therefore, A-ODDS is particularly suitable for gradual concept drift data.

Finally, Orion is superior to both Stream LOCI and A-ODDS in terms of accuracy for all three metrics. On average, Orion has 225% and 173% better JC than Stream and A-ODDS.

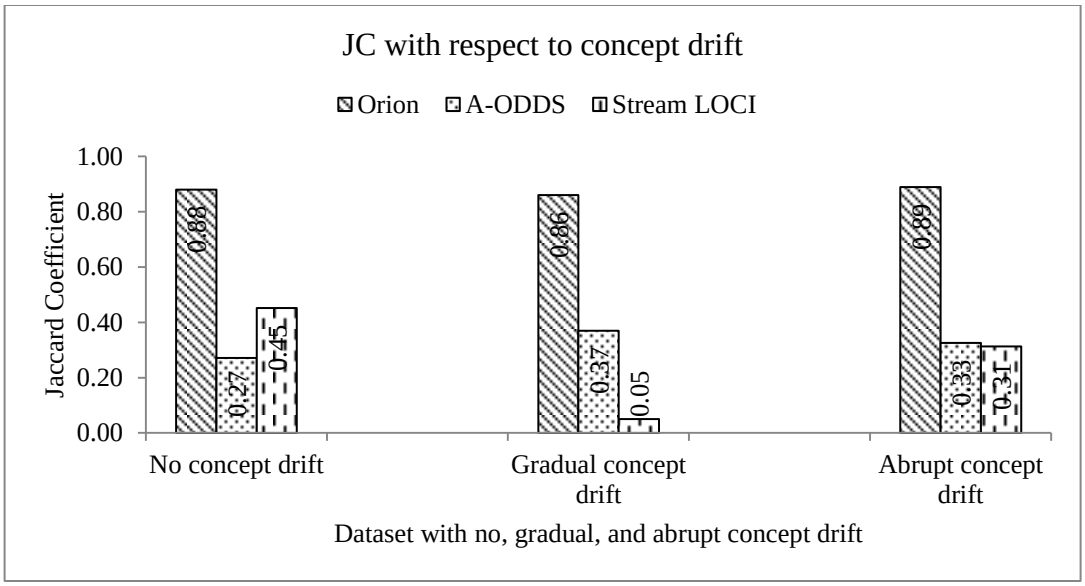

Figure 85. Impact of Concept drift on JC for the synthetic data

2.1.2.10.4. Execution Time

Figure 86 shows the effectiveness of Orion in terms of execution time. The execution times of Orion for all three types of concept drift are similar. This is because Orion adaptively changes the set of p-dimensions regardless of the existence of the concept drift. Thus, the time complexity of Orion is not affected by the existence of concept drift. Hence, the execution times of the Orion are also very similar for three types of concept drift.

The execution time of Stream LOCI is the highest for gradual concept drift compared to other two types of concept drifts. This result can be easily explained. Stream LOCI computes the MDEF of each data point for a choice of radius; however, Stream LOCI does not take radius as a user input, rather it computes the MDEF value for all possible radiuses starting from the minimum distance to the maximum distance between two data points. If the MDEF value of a data point for any radius is beyond three standard deviations, the data point is identified as an outlier. Hence, in case of the gradual

concept drift data, the maximum distance between two data points is always very high, thus Stream LOCI has to compute the MDEF value for lots of radiuses. Thus the execution time of Stream LOCI is large for gradual concept drift (Figure 86). The execution time of A-ODDS is also unaffected by concept drift.

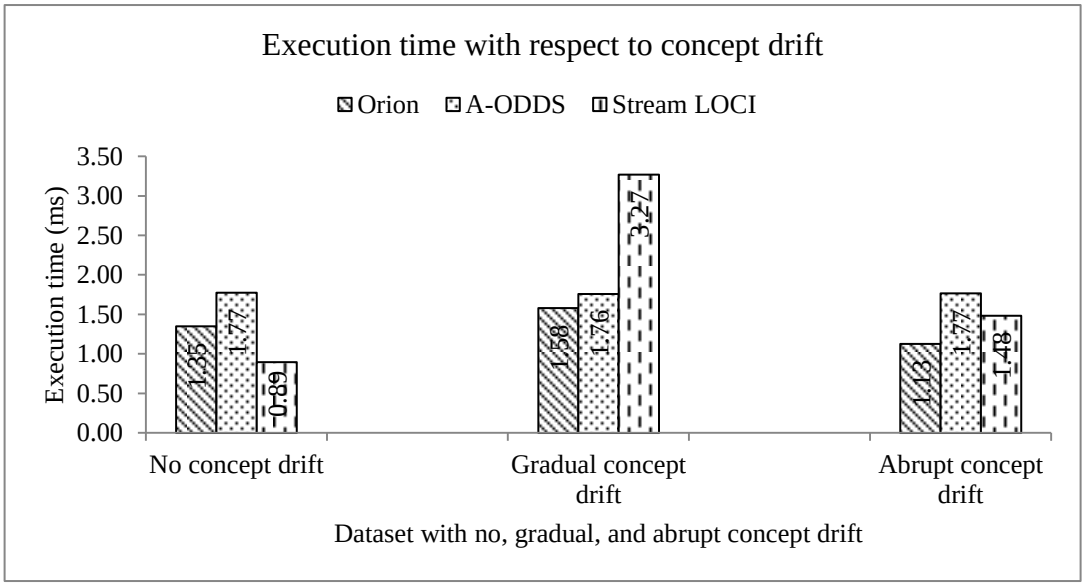

Figure 86. Impact of Concept drift on execution time for the synthetic data

2.1.2.11. Conclusions on Experimental Results for Orion

Our conclusions from the experimental evaluations that we have presented in the previous sections are as follows:

- Orion performs better than the existing state-of-the-art outlier detection algorithms for real applications like network intrusion detection, Physical Action classification, and erroneous sensor reading detection. The diversity of the datasets shows the applicability of Orion for a wide range of applications.

- Orion not only possesses better accuracy compared to the exiting outlier detection techniques, but also has better or competitive execution time.

- The impact of neighbor distance (a parameter of Orion) on accuracy is very small. Thus the user has much liberty in choosing an appropriate value for this parameter. Even if the user chooses a value that is not optimal, Orion is still capable of detecting outliers.

- The second most important parameter for Orion is k that is used to compute k-distance. The impact of k is also very insiginificant. Many outlier detection techniques are very sensitive to the choice of their parameters, especially distance based outlier detection techniques [9, 58]. The idea of density based outlier detection evolved in order to eliminate this drawback of distance based outlier detection. In our case, the sensitivity of accuracy of Orion to k is negligible. Therefore, it becomes easier for the user to choose an appropriate k value for Orion.

- The execution time of Orion is also insensitive to neighbor distance and k. Thus, Orion is applicable to many applications regardless of their values of neighbor distance and k.

- We have examined the accuracy and execution time of Orion for the cases where the percentage of outliers is in the range of 1-10%. 10% is considered a very high percentage of outliers, where the typical percentage of outliers lies between 0.001-5% [98]. Our experimental results show that the accuracy of Orion is superior to the state of the art outlier detection techniques regardless of the percentage of outliers.

- Orion is a multi-dimensional outlier detection technique. The accuracy of Orion is affected by the number of dimensions at all. The outlier detection technique Stream LOCI does not work well for high number of dimensions. This is because Stream LOCI uses Euclidian distance to measure the similarity among the data points, but Euclidian distance cannot measure the similarity if the number of dimensions is high. Thus our choice of avoiding Euclidean distance is well-justified.

- The execution time of Orion increases in a cubic way with the linear increase of number of dimensions as opposed to exponential increases suggested by multi-dimensional data density functions.

- The accuracy of Orion is small for a very small number of p-dimensions (aka population count). But once the number of p-dimensions increases a little bit, the accuracy becomes insensitive to the number of p-dimensions. We propose a heuristic in which we keep the same number of p-dimensions as the number of data dimensions. This heuristics produces optimum results for Orion. Hence, we recommend users to follow this heuristics for all applications.

- The optimal number of bin width is established from literature [5]. We use 400 bins for six standard deviation dispersion. According to our experiments, 400 bins for six standard deviations always produce optimal results. Thus, users are recommended not to modify the bin count.

- Orion does not require a huge set of bootstrapping rounds to initialize itself. The number of bootstrapping rounds typically varies between 100-500. In a typical

application like irrigation, 100 data points constitute 0.2% of the datasets. Moreover, a data stream is considered as an infinite set of data points; hence, 100 data points is negligible for an infinite set of data points.

- The accuracy and execution of Orion remain unchanged with respect to the number of data points. Since data streams are envisioned for an infinite set of data points, it is important to have constant accuracy and execution time with respect to the number of data points. Thus the effectiveness of Orion for data streams is once again validated by its constant accuracy and execution time with respect to the change of the number of data points.

- Orion can successfully handle both gradual concept drifts and abrupt concept drifts. Handling concept drifts was one of the design goals of Orion and our experiments studying the impacts of concept drifts show that we have achieved this design goal.

2.2. Experimental Analysis for Wadjet

2.2.1. Simulation Model

We use the same simulation model as we have described for Orion in Section 2.1.1. Thus we omit the description of the simulation model in this section.

2.2.1.1. Datasets

2.2.1.1.1. Synthetic Dataset

We use the same synthetic dataset that we have discussed in Section 2.1.1.3.6 for Orion.

2.2.1.1.2. Sensor Scope Dataset

Sensor Scope is an environmental monitoring system based on a wireless sensor network [6]. Sensor Scope is aimed for outdoor deployment. Barrenetxea et al. [6] deployed six different sensor networks starting from the EPFL's campus to a high mountain site. We have collected the data from their largest deployment at the EPFL's campus. Each environment monitoring station is composed of multiple sensors monitoring ambient temperature, surface temperature, solar radiation, relative humidity, soil moisture, watermark, rain meter wind speed and wind direction. However, not all sensors from all stations report all the data. Thus some stations may report less data than others. This give rise of heterogeneous schemas for data streams. Moreover, the monitoring stations are very close to each other (within the campus of the EPFL); hence the data from one station is strongly correlated with the data from another station. In our experiments we collected data from 97 stations, each of which has roughly 40K to 400K data points and 3 to 7 dimensions.

2.2.1.2. Competitive Algorithms

DB-Outlier is a continuous distance based outlier proposed by Ishida and Kitagawa [32]. DB-Outlier is based on a popular distance based outlier detection technique proposed by Knorr et al. [50]. We choose DB-Outlier as a representative technique of multiple related data streams. In the literature, there exists only a handful set of outlier detection techniques that work for multiple related streams [32, 48, 86, 100]. However, none of them works for multi-dimensional data points except DB-Outlier. Moreover, distance based outlier detection is superior compared to other outlier detection schemes in terms of accuracy [101]. Thus, we choose to compare Wadjet with DB-Outlier. DB-

Outlier detects distance-based outliers given a user-defined distance d and minimum neighbor count k; a data point is an outlier if it has fewer than k neighbors within a distance d. DB-Outlier adopts the idea of the cell based outlier detection proposed by Knorr, Ng and Tucakov [50] and takes the cell based outlier detection to the next level for data streams. This algorithm assumes multiple data streams for outlier detection and compares the data points across the data streams and detects distance based outliers. If the number of data streams is N, it assumes there exist N data streams $D_1, D_2, \dots, D_N$. At any particular point in time t, each stream produces one data point. So at time t, $D_1, D_2, \dots, D_N$ produces $s_1^t, s_2^t, \dots, s_N^t$ where s_i^t is a p-dimensional data point. To detect the outliers s_i^t is compared to all s_j^t, where $1 \leq j \leq N$ and $i \neq j$, so outliers are detected among $s_1^t, s_2^t, \dots, s_N^t$. DB-Outlier compares the data point of a stream to the data points of other streams. If a data point has fewer than k neighbors within a distance k, the data point is identified as an outlier.

2.2.1.3. Simulation Parameters

We study the impacts of changing various parameters on the performance of Wadjet. The range of values of the default value of each parameter are presented in Table 9. Wadjet executes Orion in its first phase. Therefore, Wadjet has the same set of parameters as Orion. The list of parameters we have studied for Orion is also valid for Wadjet. The impacts of the parameters in the first phase of Wadjet are the same as their impacts on Orion. Therefore, conducting experiments again based on those parameters would be redundant to what we have studied for Orion in Section 2.1.2. In order to avoid the repetition of the study of the same set of parameters we omit those parameter studies from this section and include only the studies of the additional set of parameters

for Wadjet that are not present in Orion. Although the number of dimensions and the percentage of outliers appear in the studies conducted for Orion, we also include them in the studies for Wadjet; this is because they have different impacts on the second phase of Wadjet. The default values of the percentage of outlier and confidence interval parameters are chosen based on literature [91, 98]; and the default values of the number of streams and number of dimensions parameters are chosen based on the average number of data streams and number of dimensions found in various datasets in the UCI machine learning repository [96]. For every experiment, when the impact of a parameter is under study, we vary its value within its range and fix the other parameters at their default values.

Table 9. List of parameters studied for Wadjet

Name	Synthetic data		Sensor Scope data	
	Range of Values	Default Value	Range of Values	Default Value
Number of streams	20 - 50	50	97	97
Number of dimensions	50 -100	100	4 - 10	4 – 10
Percentage of outliers	1 -10	5	5%	5%
Confidence interval	97 – 100	99.5	97 – 100	99.5

2.2.2. Experimental Results

In this section we present our experimental results for Wadjet. We start with an overall performance of both algorithms and then present the detailed studies of the parameters.

2.2.2.1. Overall Performance

2.2.2.1.1. Precision

Table 10 shows the precision of Wadjet and DB-Outlier for the synthetic and sensor scope datasets. The sensor scope data has heterogeneous schemas, but DB-Outlier does not deal with heterogeneous schemas; thus we cannot report any result for DB-Outlier

for this dataset. To the best of our knowledge, no existing outlier detection technique works for heterogeneous schemas. DB-Outlier performs really poorly compared to Wadjet for the synthetic data because DB-Outlier is not really designed for data streams where data points have different values but are correlated. DB-outlier assumes that data points from multiple streams are the same or equal. Any data point which is significantly different from the other data points is identified as an outlier. However, in our synthetic data set, data points are not equal, rather they are correlated. Thus when DB-Outlier tries to find the outliers based on the assumption that they are equal, it makes lots of mistakes and identifies lots of inliers as outliers. Thus the precision of DB-Outlier is very poor. Unfortunately, we have not come across any existing algorithm that deals with data points that are not the same but correlated. Hence our algorithm is the first one that addresses this problem. However, in order to give the reader an impression of what happens if the equality assumption does not hold, we compare our algorithm with DB-Outlier.

Table 10. Precision of Wadjet and DB-Outlier

	Synthetic data	Sensor Scope data
Wadjet	0.94	0.99
DB-Outlier	0.05	N/A

On the contrary the precision of our algorithm is very good. Wadjet can detect outliers with 94% precision for the synthetic data and 99% precision for the sensor scope data. These are very good precision since Wadjet identifies a data point as an outlier if it is identified as an outlier in either Phase 1 or Phase 2; thus two levels of error can accumulate while we are considering the precision of Wadjet.

2.2.2.1.2. Recall

DB-Outlier assumes that the data points from multiple streams are equal and it identifies a data point as an outlier if it does not have enough neighbors. However, in our dataset, the data points from multiple data streams are not equal and thus almost all data points have a very few neighbors. According to the definition of distance based outliers, these data points (data points with small sets of neighbors) are outliers. Thus, DB-Outlier identifies them as outliers and produces lots of false alarms; however the true outliers are also far from the other data points. Hence, DB-Outlier successfully detects all the outliers (Table 11), and thus it has a better recall than Wadjet. However, the 100% recall of DB-Outlier comes with the cost of a very poor precision.

Table 11. Recall of Wadjet and DB-Outlier

	Synthetic data	Sensor Scope data
Wadjet	0.99	0.67
DB-Outlier	1.00	N/A

Wadjet shows a very high recall for the synthetic data. This is because Wadjet detects outliers in two phases. In the first phase it identifies the outliers that are nonconformist to the other data points from the same streams. In the second phase, it identifies the outliers that are nonconformist to the other data points from the other correlated streams. Hence after these two phases all the outliers are essentially filtered by Wadjet. Thus Wadjet almost identifies 100% of the outliers for the synthetic data. Unlike DB-Outlier, Wadjet makes very few mistakes while identifying all the outliers.

The recall value of Wadjet for the sensor scope data is also very promising. It can approximately detect 67% of the outliers. Interestingly the recall for the synthetic data is much higher than the recall for the sensor scope data. This is because the sensor scope

data are heterogeneous in nature. The temperature data from all sensors are more correlated than the temperature from one sensor and the wind speed from another sensor. Since the sensor scope data produces a heterogeneous set of attributes, Wadjet often fails to find the cross-correlation and thus fails to identify outliers.

2.2.2.1.3. Jaccard Coefficient

Since Jaccard Coefficient is the combination of precision and recall, DB-Outlier shows a very poor performance in terms of Jaccard Coefficient as well. Although DB-Outlier identifies all the outliers in the dataset, it makes lots of false positives as well (Table 12). Thus the reliability of the algorithm is significantly poor. Hence the Jaccard Coefficient of the algorithm is poor as well.

Table 12. Jaccard Coefficient of Wadjet and DB-Outlier

	Synthetic data	Sensor Scope data
Wadjet	0.93	0.67
DB-Outlier	0.05	N/A

Wadjet can identify most of the outliers without making a lot of mistakes like DB-Outlier for the synthetic dataset. Hence the reliability and completeness of the algorithm are very good for outlier detection for heterogeneous data streams. The JC of Wadjet for the sensor scope data is also very promising. The identified set of outliers has 67% in common with the true set of outliers.

2.2.2.1.4. Execution Time

Wadjet offers not only better accuracy, but also competitive execution time (Table 13). The execution time of Wadjet is very appealing since Wadjet is processing 50 different data points from 50 different data sources where each data point has 100 dimensions. On the average, Wadjet takes 0.000936 milliseconds to process one attribute. Thus the

processing time is very suitable for data stream applications. The execution time of Wadjet for the sensor scope data is significantly small, 0.01 milliseconds. Our further experiment suggests that Wadjet is able to cluster only 10.58% of the attributes. This is because due to the heterogeneous nature of the attributes, many attributes fail to form a cluster. Once Wadjet fails to form a cluster for an attribute it does not compute the linear regression function for that attribute and skips the outlier detection test. Hence the execution time of Wadjet improves significantly.

Table 13. Execution time (in ms) of Wadjet and DB-Outlier

	Synthetic data	Sensor Scope data
Wadjet	4.68	0.03
DB-Outlier	3.61	N/A

Since the sensor scope data is a real dataset, we do not have much control over this dataset in terms of change of parameter values. So, for our parameter studies, we use the synthetic dataset alone.

2.2.2.2. Impact of Number of Streams

2.2.2.2.1. Precision

Figure 87 shows the impacts of the number of data streams on the precision of both Wadjet and DB-Outlier. As we are increasing the number of data streams, we are actually adding more and more data streams to detect outliers in Wadjet. If the newly added data streams are cross-correlated, the number of correlated attribute would increase. If the number of correlated attributes increases, the number of correlated attributes of any particular attribute would also increase. Hence, if the number of streams is large each cluster would have more attributes compared to that if the number of streams is small. However, in order to detect an outlier, an attribute value is

compared to its nearest neighbor. Therefore, the total number of attributes in a cluster does not influence the decision of outlier-ness of a data point. Therefore, the precision of Wadjet remains the same. If the newly added streams are not correlated the cluster size would not be impacted by the additional data streams. In that case the precision would not be impacted either. Hence, in both cases the precision of Wadjet remain unchanged.

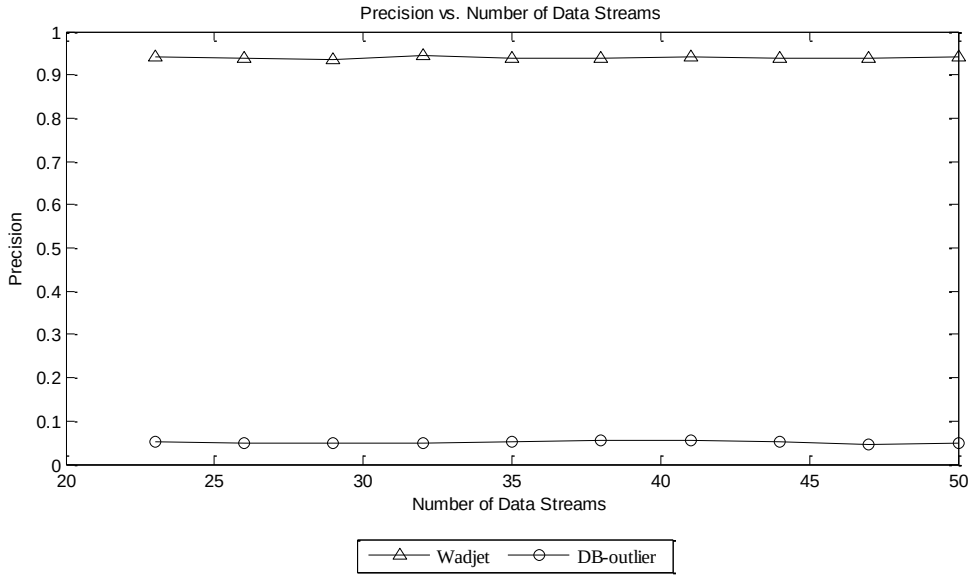

Figure 87. Impact of number of data streams on precision for the synthetic data

Interestingly, the precision of DB-Outlier is also unchanged with respect to the number of streams. In case of DB-Outlier, the number of data points is the same as the number of streams; but these data points are not equal, rather they are correlated. Hence, they are far from each other in terms of value. Thus, if the number of data points increases, DB-Outlier also misclassifies inliers as outliers. Therefore, the precision of DB-Outliers also remains unchanged. However, the precision of DB-Outlier is significantly smaller

than that of Wadjet. The precision of Wadjet is twenty times higher than the precision of DB-Outlier.

2.2.2.2.2. Recall

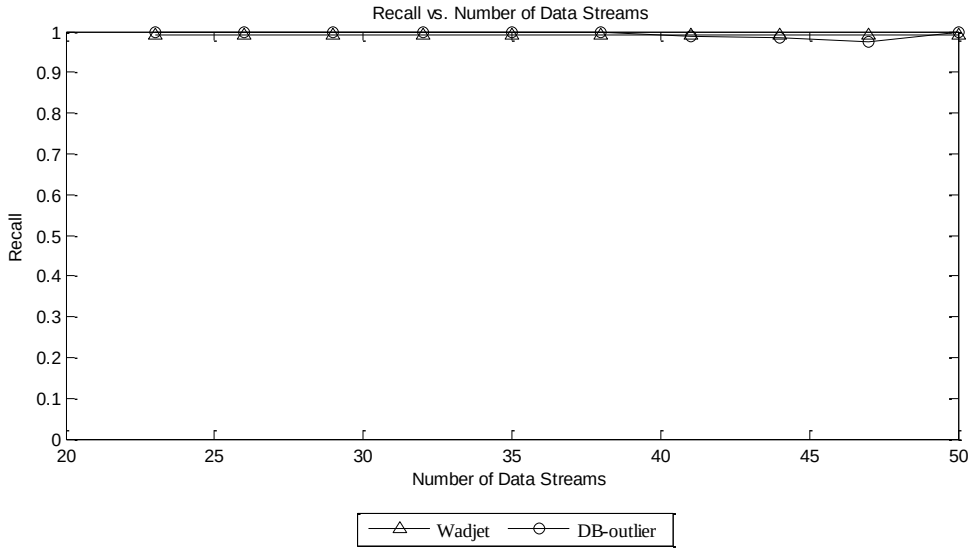

Figure 88. Impact of number of data streams on recall for the synthetic data

Figure 88 presents the impacts of the number of data streams on the recall of both Wadjet and DB-Outlier. In Figure 88, the graph for recall of Wadjet and the graph for recall of DB-Outlier superimpose one another. Thus, it is difficult to separate the recall of DB-Outlier from that of Wadjet from Figure 88. Interestingly, both Wadjet and DB-Outlier show excellent recall values for this dataset.

As the number of streams increases we add more streams to the experiments. Adding more streams means adding more attributes. Therefore, Wadjet would be able to find more correlated attributes if the newly added streams are cross-correlated. Hence, adding more attributes might increase the size of clusters. However, in order to detect significantly different attributes, an attribute value is compared to its nearest neighbor.

Thus, having more attributes in the same cluster does not increase the accuracy of outlier detection. Therefore, we see that the recall of Wadjet remains unchanged with respect to varying the number of streams.

The recall of DB-Outlier is approximately 1, meaning DB-Outlier can successfully detect all the outliers. Having more data points means having more similar or dissimilar data points. If the data points from the newly added streams are similar (in terms of Euclidian distance), they are not outliers and thus do not impact the recall. If the data points from the newly added streams are dissimilar (in terms of Euclidian distance), they might be inliers or outliers. In either case, DB-Outlier identifies the data points as outliers. Hence, the recall of DB-Outlier may increase with the increase of number of streams. However, the recall is already 100% even for a small number of data streams, thus adding more streams does not increase the accuracy of DB-Outlier in terms of recall. Practically, the recall of Wadjet and DB-Outlier is the same for different numbers of data streams.

2.2.2.2.3. Jaccard Coefficient

Since the precision and recall remain unchanged, the JC remains unchanged with respect to the number of data streams as well (Figure 89). According to Figure 89, the accuracy of Wadjet and DB-Outlier is insensitive to the change of the number of streams. The JC of Wadjet is unchanged with respect to the change of the number of streams. This result implies that we can add as many data streams as we want to detect outliers using Wadjet without affecting its JC at all. This result also implies that Wadjet is well scalable with respect to the number of streams in terms of accuracy (precision, recall and JC).

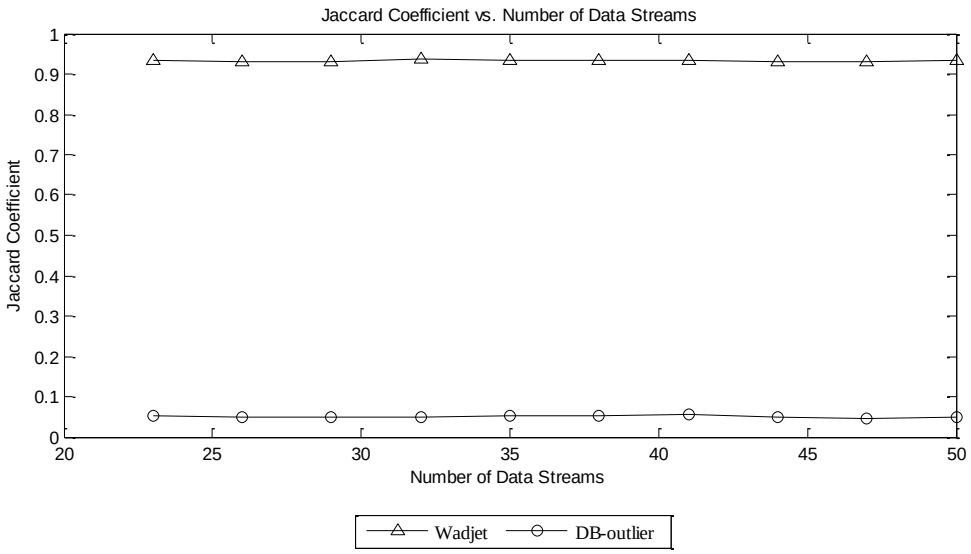

Figure 89. Impact of number of data streams on Jaccard Coefficient for the synthetic data

The JC of DB-Outlier also remains unchanged with respect to the change of number of streams; but the JC of Wadjet is twenty times higher than the JC of DB-Outlier.

2.2.2.2.4. Execution Time

Although the precision, recall and JC remain unchanged, the execution time shows some variation with respect to the change of the number of streams. As the number of streams increases, the size of the covariance matrix that finds the cross-correlation increases as well. Hence, it takes more time to process that cross-correlation matrix and thus eventually the algorithm takes longer time to finish. However, the execution time increases linearly with the increase of the number of streams. Our theoretical analysis shows that the time complexity of Wadjet is quadratic with respect to the number of streams. However, our experimental results show that the execution time increases linearly with the increase of the number of streams. This inconsistency between the theoretical and empirical analyses can only be explained if the execution time truly

increases quadratically with the increase of the number of streams; but we do not see the quadratic increase of execution time because the effect of the number of stream is not prominent up to 50 streams. Hence, if we perform experiments with a very large number of streams, we might confirm our theoretical analysis. In future work we will experiment with even a large number of data streams. On the other hand, the execution time of DB-Outlier is somewhat random. This is because we simulate our data streams as asynchronous streams and therefore not all data points from all streams are present at the same period of time. So the execution time of DB-Outliers changes sporadically (Figure 90).

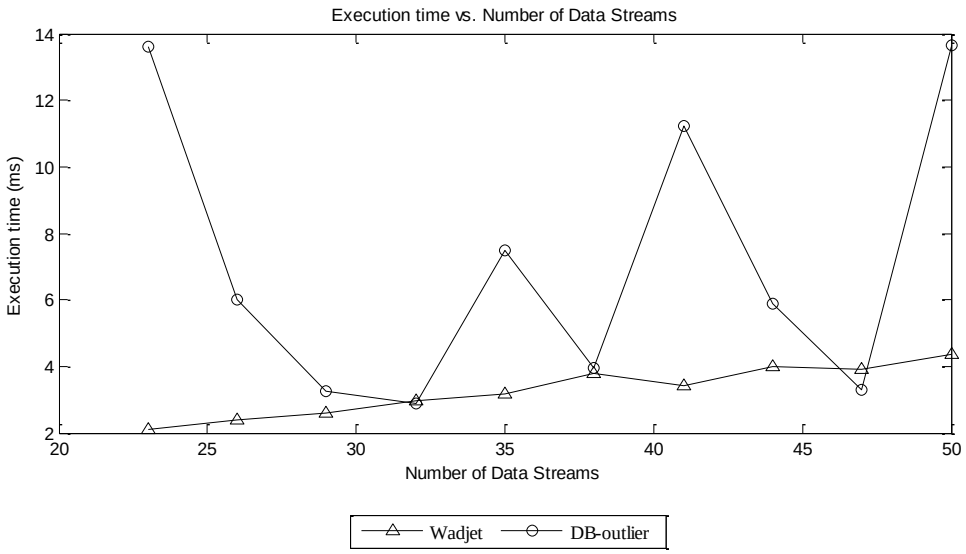

Figure 90. Impact of number of data streams on execution time for the synthetic data

2.2.2.3. Impact of Number of Dimensions

We aim to design our algorithm such that it is scalable to a large number of dimensions without negatively affecting its accuracy. Hence, for validation of whether our goal has

been reached, here we perform studies that reveal the impacts of the number of dimensions on Wadjet and DB-Outlier.

2.2.2.3.1. Precision

The precision of Wadjet and DB-Outlier is not sensitive to the number of dimensions. However, this result can be misleading since the precision of DB-Outlier is so small that it is hardly recognizable if the number of dimensions has any impact on it. However, the precision of Wadjet remains immune to the change of the number of dimensions (Figure 91). This result is very interesting because if the number of dimensions increases for each stream, the total number of attributes increases as well. Now, there can only be two cases: (1) the new attributes are correlated and (2) the new attributes are not correlated.

If the new attributes are correlated, they would be placed in a cluster and equate to the cluster head. If the attribute value is an outlier, it would be significantly different from the rest of the attribute values in the cluster. It is possible that two such outlier attribute values may appear close to each other. In that case, Wadjet would fail to identify them as outliers; otherwise Wadjet would capture them as outliers. The chance of being close to each other for two outliers is very small and increases with the increase of the number of dimensions. Thus the precision of Wadjet may decrease a little bit, but we hardly notice any change in precision with the increase of the number of dimensions.

Now, if the new attributes are not correlated, they would not be a part of a cluster and therefore, would not be detected as outliers. Any data point identified as an inlier cannot affect the precision of Wadjet since it is the ratio of the identified set of outliers and the

true set of outliers. Therefore, in the second case, the precision of Wadjet remains unchanged.

The precision of DB-Outlier is also unchanged with respect to the change of the number of dimensions. This result is a little misleading because the precision of DB-Outlier is already very small and further deterioration of precision cannot be noticed in the result. Since DB-Outlier assumes that all data points are equal, if they are not, DB-Outlier identifies them as outliers. Hence, we see that the precision of DB-Outlier is so small. Moreover, if the number of dimensions increases, the quality of similarity measurement (in terms of Euclidian distance) of DB-Outlier deteriorates. However, the effect of that is not noticeable for such low precision of DB-Outlier.

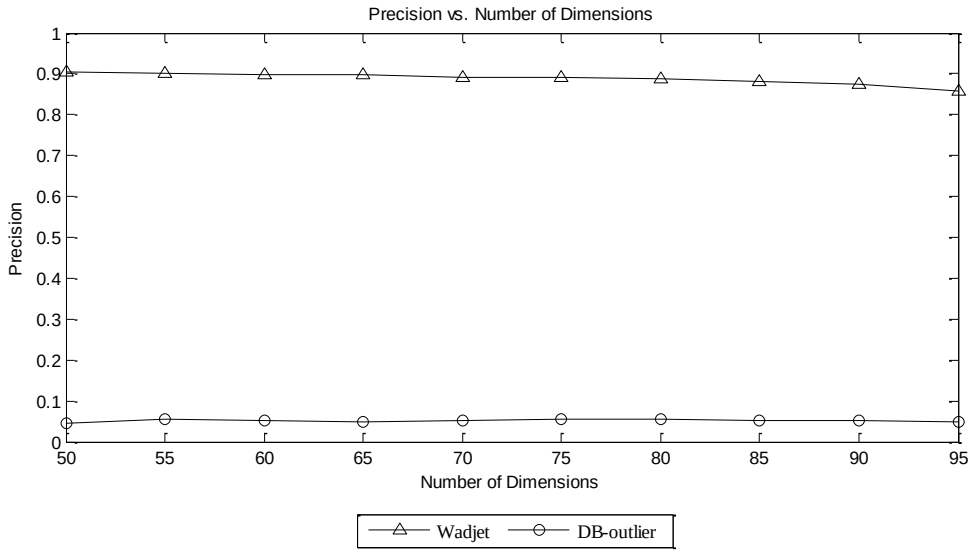

Figure 91. Impact of number of dimensions on precision for the synthetic data

2.2.2.3.2. Recall

Figure 92 shows the impact of the number of dimensions on recall. The graph of recall for Wadjet superimposes the graph of recall for DB-Outlier in Figure 92. Hence, it is difficult to identify them separately.

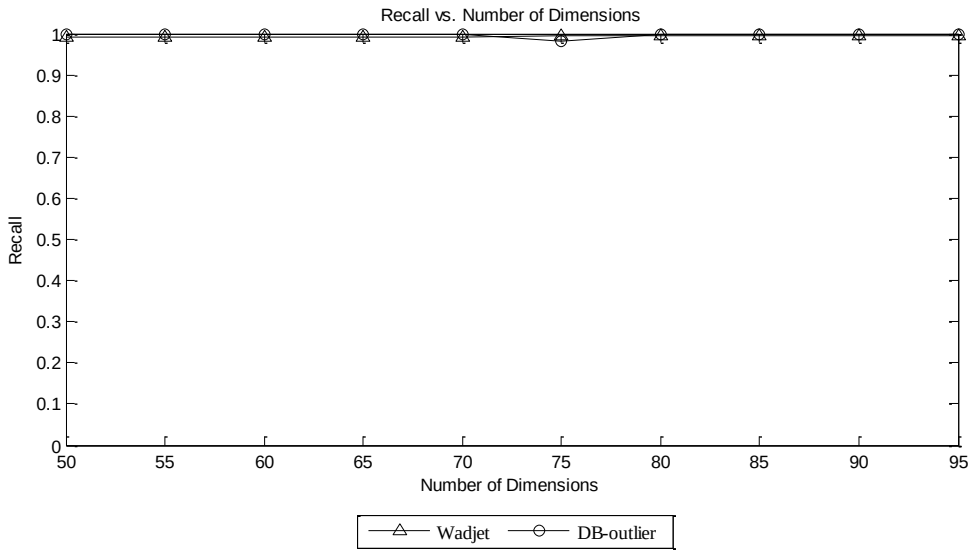

Figure 92. Impact of number of dimensions on recall for the synthetic data

We argued that as the number of dimensions increases, DB-Outlier would fail to measure the similarity among the data points, and more outliers should disguise among the inliers and thus we expected the decrease of recall of DB-Outlier with the increase of the number of dimensions. Interestingly, we do not see that result in Figure 92. The recall of DB-Outlier is almost constant with respect to the change of the number of dimensions. This is because all data points satisfy the outlier criteria of DB-Outlier. DB-Outlier assumes the data values for all data points to be equal, but in our dataset, they are not equal. Hence, the distance between any two data points is very large; so all data points are outliers for DB-Outlier. Therefore, the recall of DB-Outlier is very high.

The recall of Wadjet is also very high. Since Wadjet uses a two phase outlier detection, most of the outliers are correctly identified in the two phases of Wadjet. According to the experiments presented in Section 2.1.2.5.2 (the impact of the number of dimensions on Orion), the first phase of Wadjet can detect approximately 85-90% of the outliers. Since the recall of Wadjet is 100%, the rest of the outliers 10-15% are detected in the

second phase. Interestingly, the recall of Orion and Wadjet is not sensitive to the change of the number of dimensions. Thus, none of the two phases of Wadjet is actually sensitive to the change of the number of dimensions.

In our experimental results in Figure 92, we do not see any sensitivity with respect to the change of the number of dimensions. However, arguably if the number of dimensions increases, it is very likely that the cluster size would be increased and some attributes may find new correlated attributes. Therefore, the outlier-ness of some attributes would have been revealed which might be missed if there are not enough attributes. However, this theoretical conjecture is not supported by our experimental results in Figure 92. This is because almost 85-90% of the outliers are already detected in the first phase and the second phase contributes the rest. Therefore, any change of the recall with respect to the number of dimensions becomes insignificant for Wadjet.

2.2.2.3.3. Jaccard Coefficient

Since the precision and recall remain unchanged for both Wadjet and DB-Outlier, the JC of both algorithms remains unaffected with respect to the number of dimensions as well (Figure 93). The detailed discussion of the results for unchanged precision and recall with respect to the number of dimensions is presented in the previous two Sections 2.2.2.3.1 and 2.2.2.3.2. Therefore, we do not provide the same reasoning in this section again.

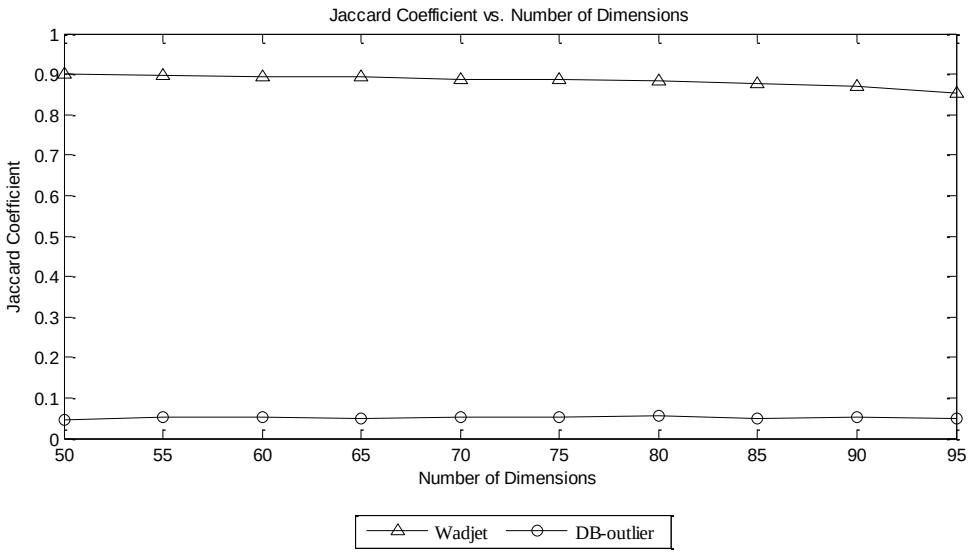

Figure 93. Impact of number of dimensions on Jaccard Coefficient for the synthetic data

2.2.2.3.4. Execution Time

The execution time of Wadjet is affected by the number of dimensions (Figure 94). As the number of dimensions increases, the size of the cross-correlation matrix increases as well. Wadjet has to process a bigger cross-correlation matrix that eventually increases its execution time. According to our theoretical analysis, the time complexity of Wadjet is cubic with respect to the number of dimensions. According to Figure 94, the increase of execution time is linear to the increase of the number of dimensions. Once reason for this behavior might be the effect of the number of dimensions is not prominent for such a small number of dimensions. Hence, the increase of execution time is linear with respect to the number of dimensions.

The execution time of DB-Outlier increases linearly with the increase of the number of dimensions in Figure 94. This is because DB-Outlier distributes the data points into grid cells and measures the similarity using Euclidian distance. Finding the appropriate grid

cell for a m dimensional data point requires m number of comparisons; and Euclidian distance computation requires m difference computations along all dimensions. Both of these algorithms require linear time with respect to the number of dimensions. Hence, the execution time of DB-Outlier increases linearly with the increase of the number of dimensions. The average execution times of Wadjet and DB-Outlier are 2.4 ms and 2.8 ms, respectively. Hence, on average the execution time of Wadjet is 14.28% smaller than that of DB-Outlier.

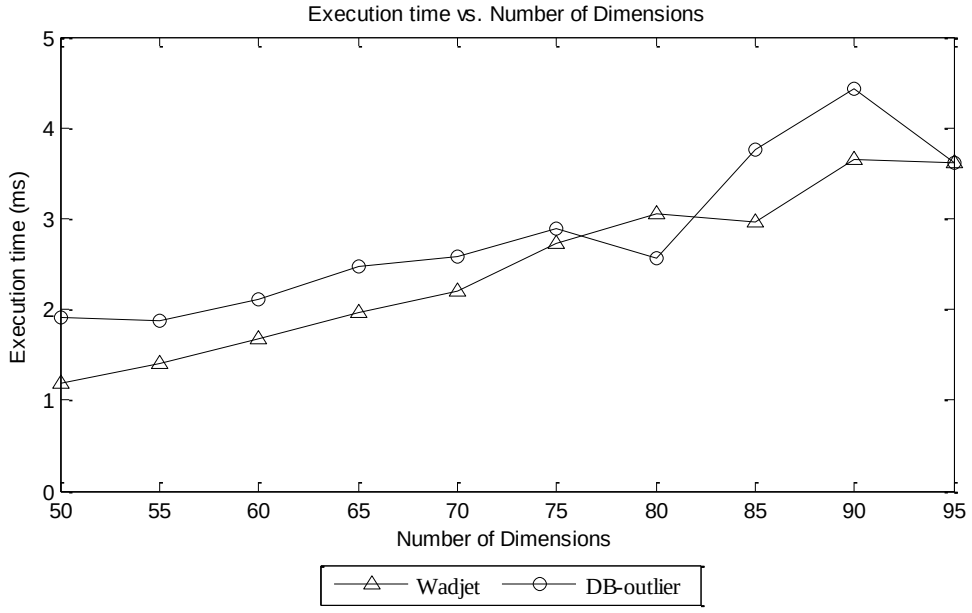

Figure 94. Impact of number of dimensions on execution time for the synthetic data

2.2.2.4. Impact of Percentage of Outliers

In this section we study the impacts of the percentage of outliers on the performance of the two algorithms.

2.2.2.4.1. Precision

Figure 96 shows the impact of percentage of outliers on precision. The precision of both DB-Outlier and Wadjet increases with the increase of percentage of outliers. DB-Outlier

assumes all data points from different data streams to be equal; however, in our dataset, the data points from different streams are not equal, rather correlated. Therefore, DB-Outlier detects all the data points as outliers. Hence, as the number of outliers increases in the dataset, DB-Outlier is able to find more outliers. Thus the precision of DB-Outlier increases with the increase of the percentage of outliers. A further investigation shows that the precision of DB-Outlier is almost equal to the percentage of outliers. This is because the precision is defined as true positives divided by the total detected positives. Since the recall of DB-Outlier is 100% (Figure 97), all the outliers are also identified as outliers; on top of that all inliers are identified as outliers as well. Thus the precision becomes the number of outliers divided by the total number of data points, which is the same as the percentage of outliers. Thus, the precision of DB-Outlier is the same as the percentage of outliers.

The precision of Wadjet also increases when the percentage of outliers increases as shown in Figure 96. According to Figure 40, the first phase of Wadjet is sensitive to the percentage of outliers; as the percentage of outliers increases the precision of the first phase increases as well. We observe a similar trend for the second phase as well. Hence, the precision of Wadjet increases with the increase of the percentage of outliers. In the first phase, the increase of precision with the increase of the percentage of outliers is due to the shift of the cluster centers. In the second phase, the number of false positives increases with the increase of the percentage of outliers (the total number of false positives for 1% of the outliers is 317 and it increases to 664 if the percentage of outliers becomes 10%). Therefore, the precision should decrease; but the increase of true positives is much higher than the increase of false positives (Figure 95). Therefore,

as the percentage of outlier increases, the number of true positives outnumbers the number of false positives and we see the increase of precision for the second phase of Wadjet. Thus, based on the results of the two phases, the precision of Wadjet increases with the increase of the percentage of outliers.

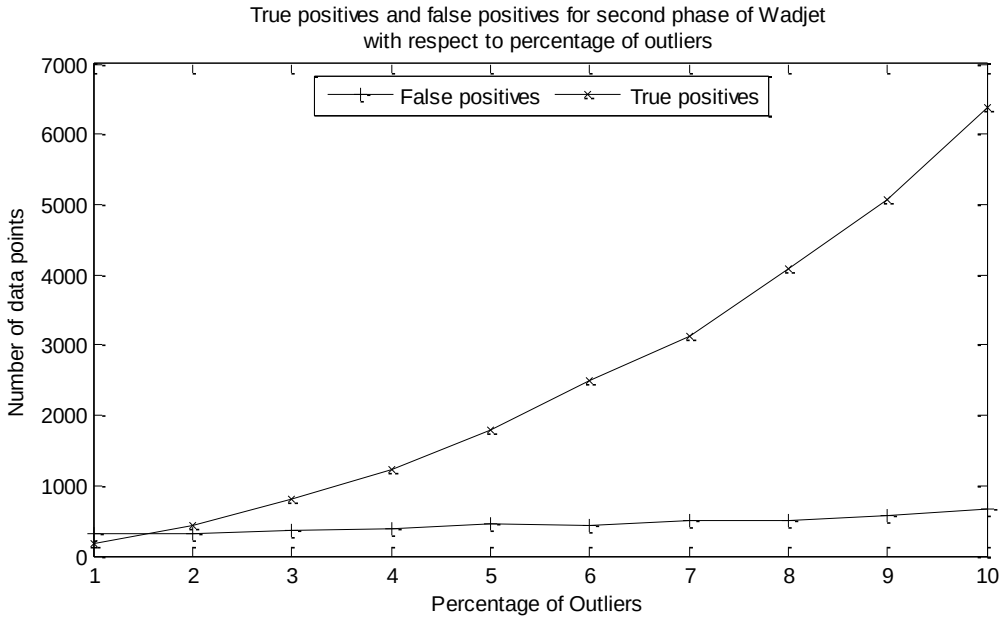

Figure 95. Impact of percentage of outliers on true positives and false positives for the second phase of Wadjet in synthetic dataset

Moreover, regardless the percentage of outliers, Wadjet significantly outperforms DB-Outlier in terms of precision.

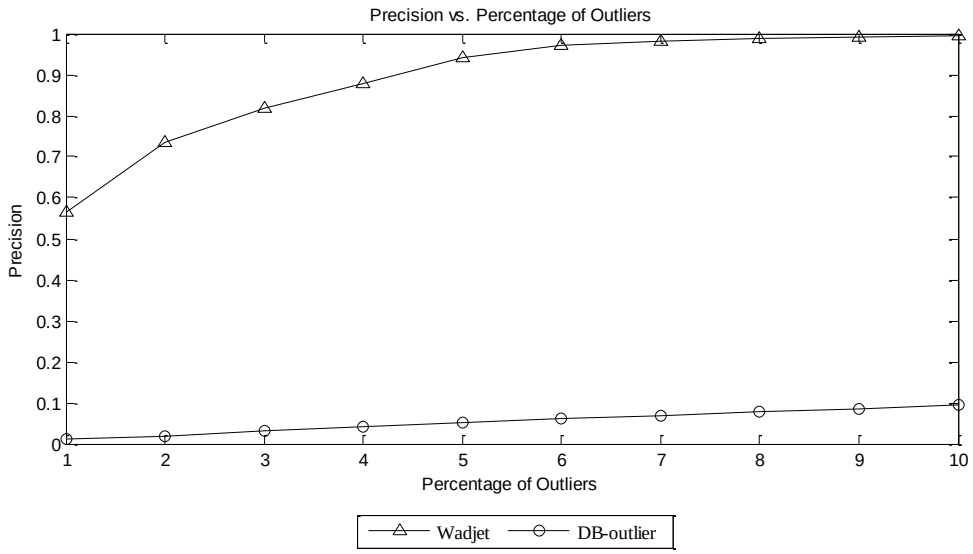

Figure 96. Impact of percentage of outliers on precision for the synthetic data

2.2.2.4.2. Recall

Although the precision of Wadjet and DB-Outlier increases with the increase of the percentage of outliers, the recall of both algorithm remains unchanged with respect to the percentage of outliers (Figure 97). DB-Outlier identifies all the data points as outliers, hence for any percentage of outliers, the recall of DB-Outlier is 100%.

Our algorithm detects outliers in two phases. Based on the results we presented in Figure 42, the recall of the first phase decreases with the increase of the percentage of outliers. The recall of Orion (the first phase of Wadjet) becomes as low as 70%. Thus, although the first phase missed some outliers, the second phase successfully identifies those outliers and the recall of Wadjet is close to 100%.

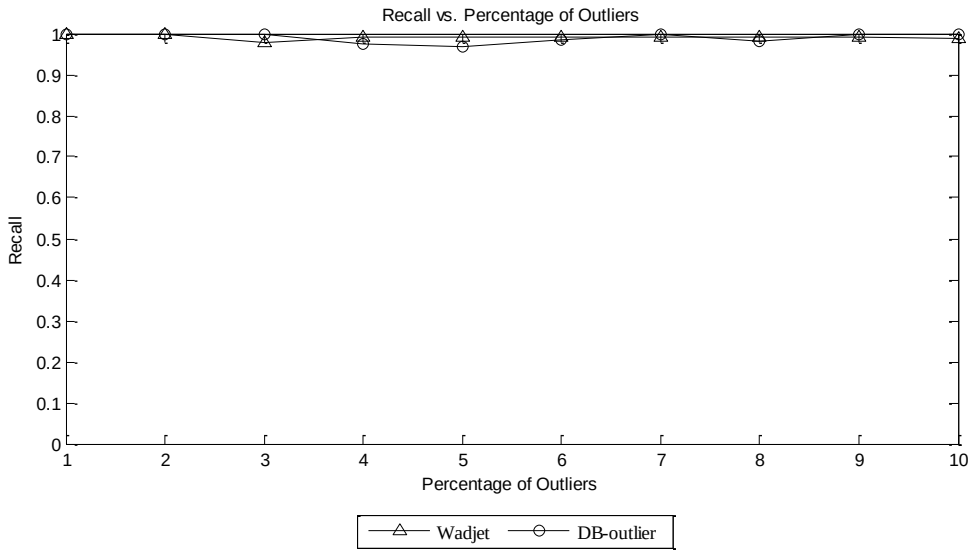

Figure 97. Impact of percentage of outliers on recall for the synthetic data

2.2.2.4.3. Jaccard Coefficient

Figure 98 shows the impacts of the percentage of outliers on JC for both Wadjet and DB-Outlier. The change of the JC of Wadjet with respect to the change of the percentage of outliers is as expected. The precision of Wadjet increases and its recall remains unchanged with respect to the change of the percentage of outliers; thus, the JC increases a little bit due to the increase of precision of Wadjet. The same conclusion also holds for DB-Outlier: the JC of DB-Outlier increases with the increase of the percentage of outliers. Since we have already discussed the results in the last two Sections 2.2.2.4.1 and 2.2.2.4.2, we omit the discussion here.

2.2.2.4.4. Execution Time

The execution time of Wadjet drastically reduces as the percentage of outliers increases (Figure 99). The execution time of the first phase is unaffected by the percentage of outliers (Figure 46). Hence, the reduction of execution time must be induced by the second phase of Wadjet. This is because as the percentage of outliers increases, many

outliers are detected in the first phase and they never really pass through the second phase. Figure 100 shows the number of data points processed in the second phase of Wadjet. As the percentage of outlier increases, the first phase detects many of them and the second phase does not process them at all. Thus if the percentage of outliers increases, the second phase has to process less data and therefore, the execution time of the second phase reduces. However, the execution time of the first phase remains unchanged. Thus the execution time decreases with the increase of the percentage of outliers (Figure 99).

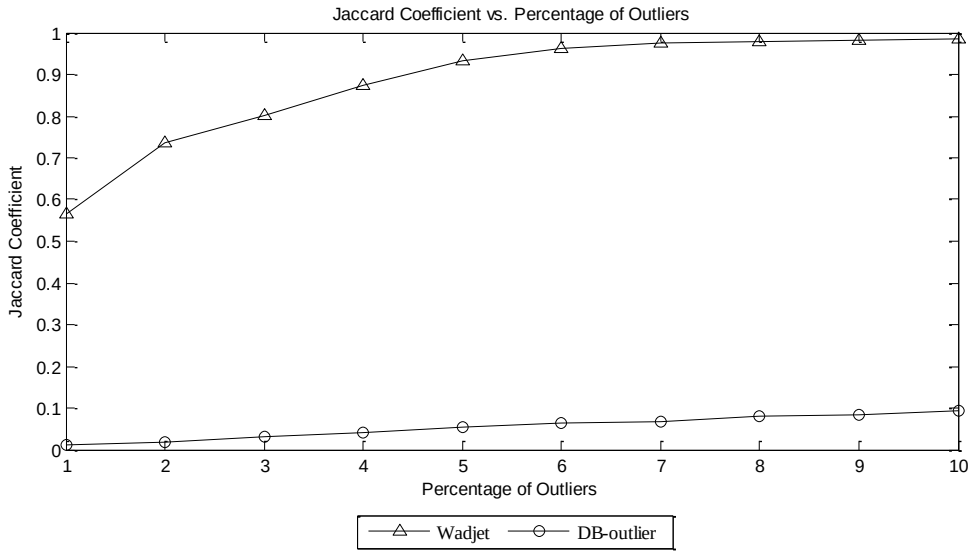

Figure 98. Impact of percentage of outliers on Jaccard Coefficient for the synthetic data

The execution time of DB-Outlier also decreases with the increase of the percentage of outliers. If a lot of data points are outliers, they are placed themselves in the yellow cells. Based on the algorithm, the data points in the yellow cells are outliers, we do not need to process them further, which reduces the computation significantly. Therefore the execution time decreases with the increase of the percentage of outliers.

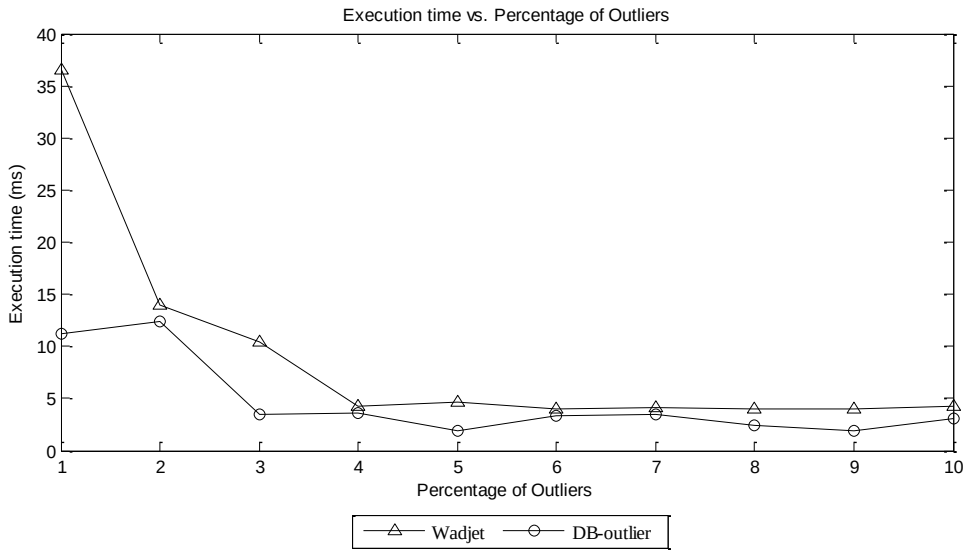

Figure 99. Impact of percentage of outliers on execution time for the synthetic data

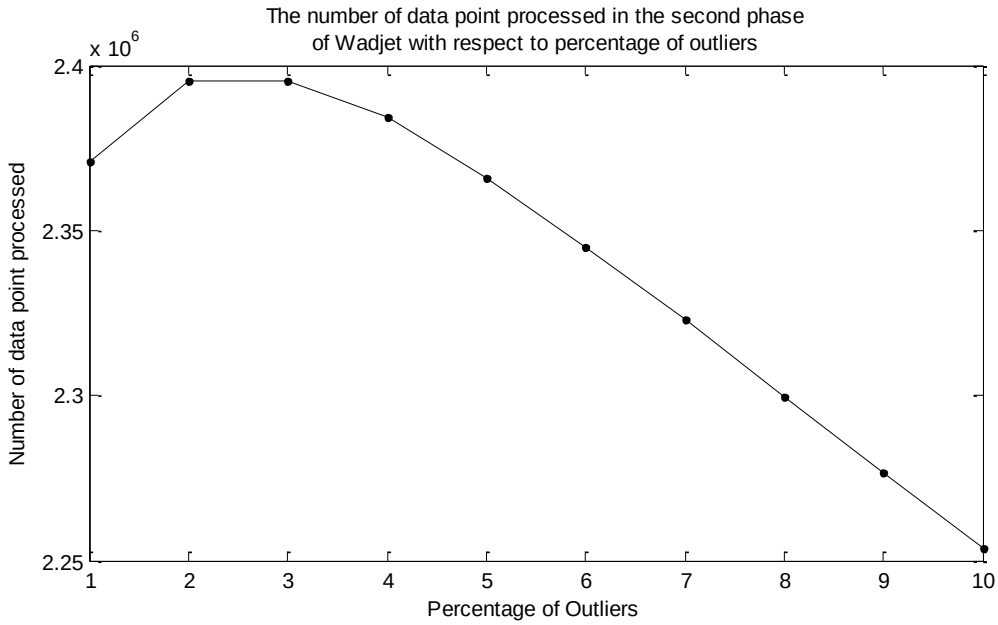

Figure 100. Impact of percentage of outliers on total number of data processed in the second phase of Wadjet

2.2.2.5. Impact of Confidence Interval

The only additional parameter we need for Wadjet besides the parameters of Orion is confidence interval. Confidence interval reveals how confident we are on our results. If

the confidence interval increases we would be more confident on the precision. The recall may decrease a little bit but the precision will increase with the increase of confidence interval. In this section we present the impact of confidence interval on Wadjet. Since DB-Outlier does not require the confidence interval as a parameter, the performance of DB-Outlier remains unchanged with respect to the change of the confidence interval.

2.2.2.5.1. Precision

In our experimental study, we find that our precision is insensitive to the change of the confidence interval (Figure 101). The first phase of Wadjet does not require the confidence interval as a parameter, and therefore, the confidence interval has no impact on the first phase. The second phase of Wadjet uses the confidence interval as a parameter. Thus the second phase of Wadjet might have some impact with respect to the change of the confidence interval. This result could be misleading because the precision value is already very high for even 97.5% confidence level. Thus there is not much room to increase the precision anymore. Hence the precision of Wadjet appears insensitive to the confidence interval.

Since DB-Outlier does not use the confidence interval as a parameter, its precision remains unchanged with respect to the change of the confidence interval. Moreover, the precision of Wadjet is almost twenty times of the precision of DB-Outlier for any confidence interval.

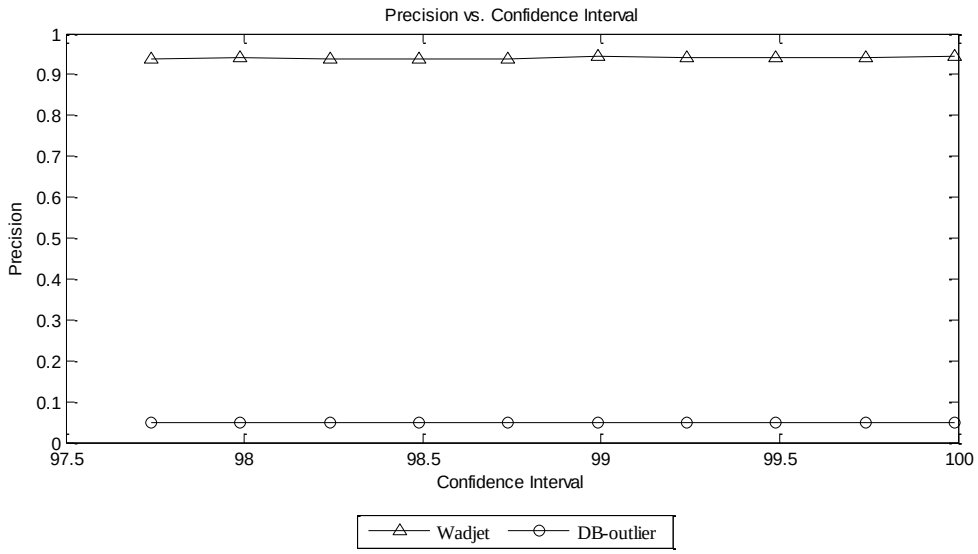

Figure 101. Impact of confidence interval on precision for the synthetic data

2.2.2.5.2. Recall

Like precision, the recall of Wadjet also remains unchanged with respect to the change of the confidence level (Figure 102). According to our discussion in Section 2.6 in Chapter 3, if the confidence interval increases, the results become more reliable and Wadjet would be able to detect few outliers. As we have discussed before, the confidence interval does not affect the accuracy of the first phase of Wadjet, it only affects the second phase. In our results we observe an unchanged recall value for Wadjet in Figure 102. This is because a lot of outliers are actually captured in the first phase. Therefore, even though the second phase fails to detect some outliers, the impact of that failure is insignificant in the total recall in Figure 102. In order to illustrate the idea, we add the impact of the confidence interval on the recall of the second phase only in Figure 103. The recall of the second phase decreases with the increase of the confidence interval as our hypothesis (the recall would decrease with the increase of

confidence interval) has predicted but the effect of change of the combined recall is very insignificant.

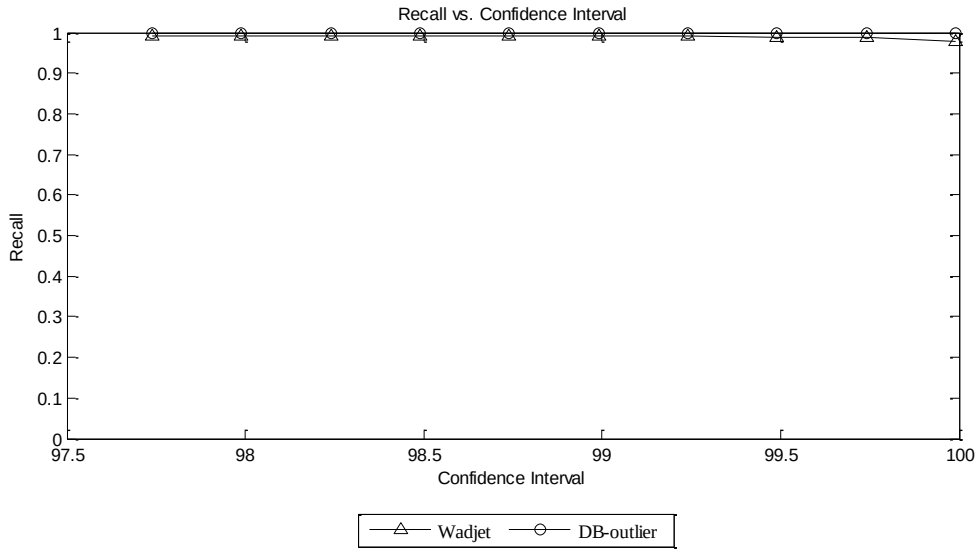

Figure 102. Impact of confidence interval on recall for the synthetic data

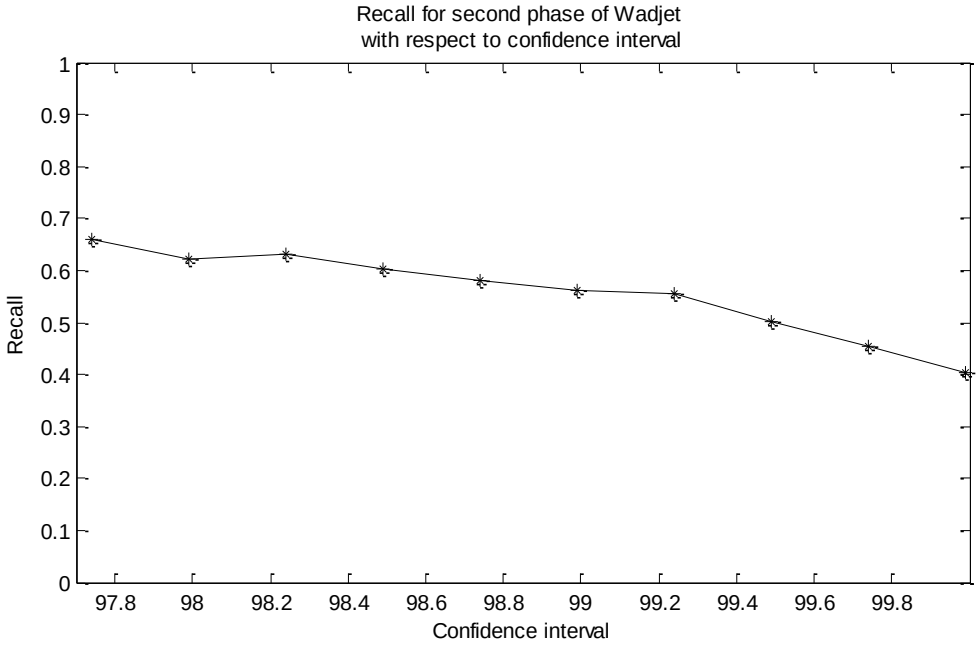

Figure 103. Impact of confidence interval on the second phase of Wadjet for the synthetic dataset

DB-Outlier does not use the confidence interval as a parameter and thus its recall r is unaffected with respect to the change of the confidence interval.

2.2.2.5.3. Jaccard Coefficient

Since precision and recall remain unchanged with respect to the change of the confidence level, the JC also remains unchanged with respect to the change of the confidence level (Figure 104).

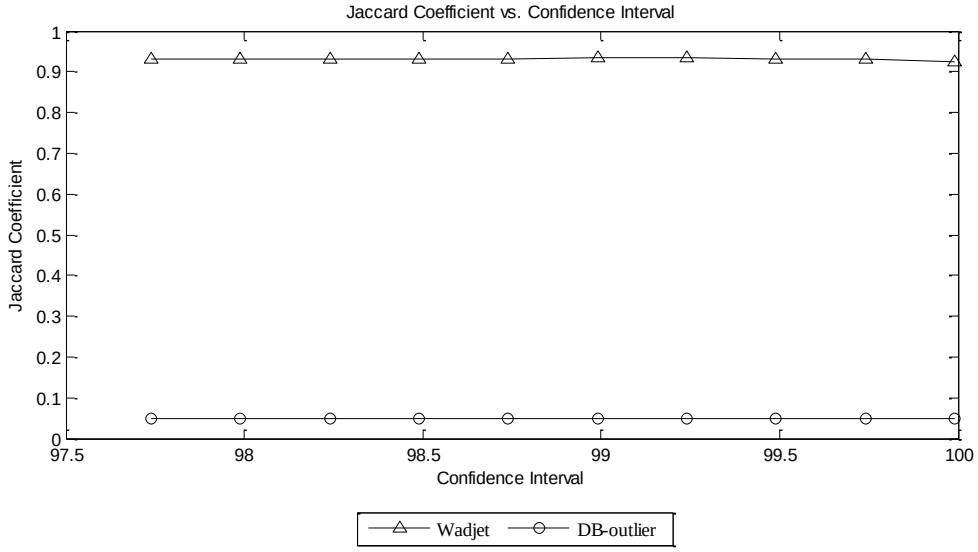

Figure 104. Impact of confidence interval on Jaccard Coefficient for the synthetic data

2.2.2.5.4. Execution Time

Although with respect to the change of the confidence interval, the precision, recall and JC remain unchanged, the execution time shows some sensitivity (Figure 105). This is because as the confidence interval becomes large, two attributes can form a cluster only if they are highly correlated (meaning their Pearson correlation coefficient is close to 1) and hence, the number of attributes in a cluster becomes smaller. Thus, if the number of attributes in a s small, Wadjet finds lots of clusters with single attributes. Those

attributes are not processed further for outlier detection. Therefore, the execution time is reduced a little bit. Hence, the execution time of Wadjet decreases a little bit with the increase of the confidence interval. Since DB-Outlier does not take the confidence interval as a parameter, its execution time remains unaffected with respect to the change of the confidence interval.

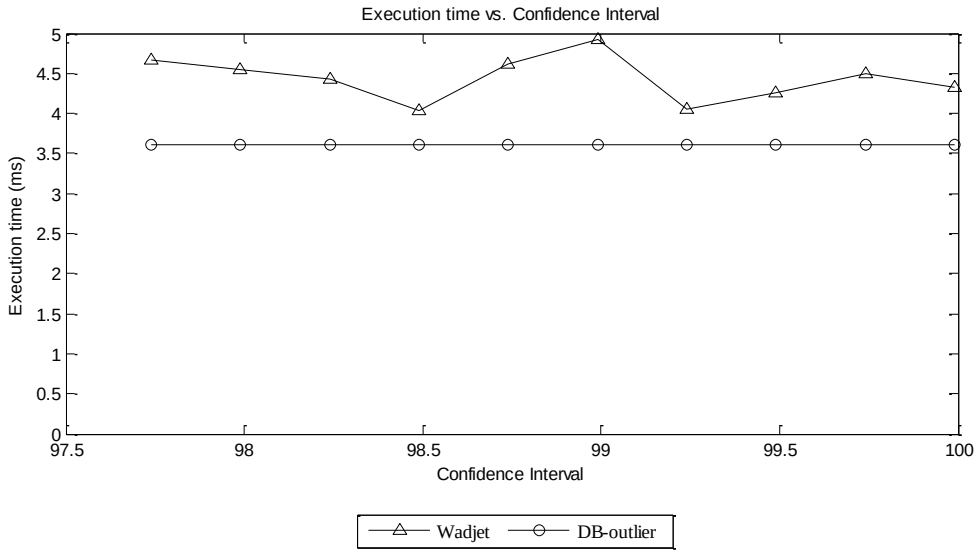

Figure 105. Impact of confidence interval on execution time for the synthetic data

2.2.2.6. Conclusions on Experimental Results for Wadjet

Our conclusions from the experimental evaluations presented in the previous sections for Wadjet are as follows:

- The state of the art outlier detection technique DB-Outlier is not applicable to a dataset where data points from multiple streams are not equal, but rather correlated.

- DB-Outlier is also inapplicable for a dataset where data points are multi-dimensional and the number of dimensions is large.

- Our algorithm, Wadjet, is applicable and effective in both cases. Moreover, Wadjet does not assume data points from multiple streams to be equal. Hence, the applicability of Wadjet is much broader compared to the existing outlier detection technique.

- Wadjet scales well with respect to the number of streams in terms of accuracy. As the accuracy (precision, recall, and JC) of Wadjet is not affected by the number streams, Wadjet is able to handle a large number of streams without affecting its accuracy.

- Wadjet is perfectly applicable for the heterogeneous dataset that we have obtained from the SensorScope project [6] where the existing outlier detection technique fails to work.

- Wadjet can handle multi-dimensional data points. Wadjet scales well with respect to the number of dimensions in terms of accuracy. We have performed experiments with 100 dimensional data points and this high number of dimensions does not affect the accuracy of Wadjet at all.

- Both the number of streams and number of dimensions affect the execution time. The execution time increases with the increase of either number of streams or number of dimensions. Thus, adding more streams or dimensions may require more computational resources for Wadjet.

- The JC of Wadjet increases with the increase of percentage of outliers. Even if a dataset has a large number of outliers, Wadjet can detect them. In an application like intrusion detection, corrupting lots of data points by an intruder would not weaken the accuracy of Wadjet.

- Interestingly, the higher percentage of outlier reduces the execution time due to our two phase outlier detection scheme. The first phase is insensitive to the percentage of outliers and detects many of them; thus the first phase reduces the burden on the second phase and the overall execution time improves with the increase of percentage of outliers.

- The accuracy (precision, recall and JC) of Wadjet is not very sensitive to the selection of the confidence interval. Thus, the user has freedom to choose an appropriate confidence interval from a wide range of values.

- The execution time of Wadjet increases with the decrease of the confidence interval. Therefore, the user must consider the execution time before choosing an appropriate confidence interval.

CHAPTER V
CONCLUSIONS AND FUTURE WORK

In this dissertation, we have proposed two outlier detection techniques for data streams, called Orion and Wadjet. The first algorithm, Orion, has been designed to detect outliers in single data streams that are independent of each other. There are a lot of data stream applications where data sources are so different and so far from one another that there is hardly any relationship among them [15]. Examples of such applications are environmental monitoring for a large area [15], physical action separation [96], and carbon sequestration [14]. Orion treats each data stream individually and detects its outliers based on the temporal correlations among the data points from same stream. Orion addresses the following characteristics of data streams: transiency, notion of time, notion of infinity, uncertainty, concept drift, and multidimensionality.

Our second algorithm, Wadjet, has been designed to detect outliers in multiple data streams which may or may not be related to each other. In some applications, such as environmental monitoring in a small area [6] and chlorine measurement [48], the data points from multiple streams are not independent of one another. To increase accuracy, Wadjet exploits the cross-correlations, if any, among the data points from multiple streams and identifies a data point in a data stream as an outlier if it is nonconformist to either the temporal correlation with the data points from the same stream or the cross-correlations with the data points from other streams. Outlier detection for asynchronous heterogeneous data streams is a relatively new area. To the best of our knowledge,

Wadjet is the first algorithm that works with a set of heterogeneous data streams that can be asynchronous in nature.

We have conducted a complexity analysis to evaluate the time and space complexity of Orion and Wadjet. By means of simulation and using both real and synthetic datasets, we have performed comprehensive experiments to compare Orion with the two existing algorithm, A-OODS and Stream LOCI, for single data streams and compare Wadjet with the existing algorithm, DB-Outlier, for multiple data streams. The comparison studies are based on execution time, precision, recall and Jaccard Coefficient. In the following sections, we first summarize the performance evaluation results and then discuss our future research.

1. Summary of the Performance Evaluation Results

1.1. Summary of the Results of Orion

Outlier detection for multi-dimensional data streams is a relatively new area of research. Outlier detection for multi-dimensional data streams possesses critical challenges. Outlier detection requires similarity measurements among the data points. Popular similarity measurement techniques such as distance metrics are incapable of dealing with multi-dimensional data due to the curse of dimensionality. This is a significant challenge since more and more data are becoming multi-dimensional every day. Hence, tracking outliers for multi-dimensional data is very challenging.

In this dissertation we have proposed an effective and efficient outlier detection technique for multi-dimensional independent data streams, Orion. We use a data density

function along a projected dimension that reveals the outlier nature of a data point. The summary of Orion is as follows:

- Orion is an effective and efficient Outlier detection technique for multi-dimensional data streams.

- In order to detect outliers, we do not need to analyze the data points from a multi-dimensional perspective; rather, we can analyze a data point from a single dimensional perspective that reveals the outlier nature of the data points.

- We have shown the effectiveness of an evolutionary algorithm in the area of data streams. To the best of our knowledge, Orion is the first algorithm that uses an evolutionary algorithm for data streams. Orion validates the proposition that an evolutionary algorithm has good prospects in the area of data streams analysis.

- Orion uses two outlier metrics to detect outliers. The rationale behind the usage of two outlier metrics instead of one is that if one outlier metric fails to reveal the outlier-ness of a data point, the other can be helpful in revealing the outlier-ness of that data point.

- Time complexity of Orion is cubic with respect to the number of dimensions which is much better compared to exponential time complexity of multi-dimensional data density function.

- The space complexity of Orion is linear with respect to number of dimensions which is way better than the exponential space complexity of multi-dimensional hyperspace.

- Neither time nor space is sensitive to number of data points. The memory usage of Orion is limited although the number of data points is infinite.

- Orion performs better than existing state-of-the-art outlier detection algorithms, Stream LOCI and A-ODDS, for real applications like network intrusion detection, physical action classification, and erroneous sensor reading detection. The diversity of the datasets shows the applicability of Orion for a wide range of applications.

- Orion not only possesses better accuracy but also has better or competitive execution time compared to exiting outlier detection techniques.

- The impact of neighbor distance (a parameter of Orion) on Orion's accuracy is insignificant. Thus the user has much liberty while choosing an appropriate value for the parameter. Even if the user chooses a neighbor distance value that is not optimal, Orion is still capable of detecting outliers.

- The second most important parameter of Orion is k which is used to compute k-distance. The impact of k is also very insignificant. Many outlier detection techniques are very sensitive to the choice of values of their parameters, especially distance based outlier detection techniques [9, 58]. The idea of density based outlier detection evolved in order to eliminate this drawback of distance based outlier detection. In our case, the sensitivity of accuracy of Orion

to this k parameter is negligible. Therefore, it is easier for the user to choose an appropriate value for k for Orion.

- The execution time of Orion is also insensitive to neighbor distance and k. Thus, Orion is applicable to many applications regardless of their values of neighbor distance and k.

- We have examined the accuracy of execution time of Orion for 1-10% of outliers. 10% is considered a very high percentage of outliers, where the typical percentage of outliers lies between 0.001-5% [98]. Our experimental results show that the accuracy of Orion is superior to that of the state-of-art outlier detection techniques regardless of the percentage of outliers.

- Orion is multi-dimensional outlier detection technique. The accuracy of Orion is not affected by the number of dimensions. The existing outlier detection technique Stream LOCI does not work for a high number of dimensions. This is because Stream LOCI uses Euclidian distance to measure the similarity among the data points, but Euclidian distance cannot measure the similarity if the number of dimensions is high. Thus our choice of avoiding Euclidean distance is well justified.

- The accuracy of Orion is low for a very small number of p-dimensions (aka population count); but once the number of p-dimensions increases a little bit, the accuracy becomes insensitive to the number of p-dimensions. We propose a heuristic in which we keep the same number of p-dimensions as the number of

data dimensions. This heuristics produces optimum results for Orion. Hence, we recommend users to follow this heuristics for all applications.

- The optimal number of bin width is established from literature [5]. We use 400 bins for six standard deviation dispersion. According to our experiments, 400 bins for six standard deviations always produce optimal results. Thus, users are recommended not modify the bin count.

- Orion does not require a huge set of bootstrapping rounds to initialize it. The number of bootstrapping rounds typically varies between 100-500. In a typical application like irrigation data, 100 data points constitute 0.2% of the datasets. Moreover, a data stream is considered as an infinite set of data points; hence, 100 data points is negligible for an infinite set of data points.

- The accuracy and execution of Orion remain unchanged with respect to the number of data points. Since data streams are envisioned for an infinite set of data points, it is important to have constant accuracy and execution time with respect to the number of data points. Thus the effectiveness of Orion for data streams is once again validated by its constant accuracy and execution time with respect to the change of the number of data points.

- Experiments studying the impacts of concept drift show that Orion is unaffected by concept drift and can handle both gradual and abrupt concept drifts effectively without affecting its accuracy or execution time.

1.2. Summary of the Results of Wadjet

Wadjet is designed for multiple data streams that may or may not be correlated. Unlike other approaches, Wadjet does not assume equality correlation among the data points from multiple streams. Moreover, in the first step, Wadjet identifies the cross-correlations among the data points. Once Wadjet finds significant cross-correlations, then it tries to detect outliers based on those correlations. Below is a summary of the key contributions of Wadjet.

- To the best of our knowledge, Wadjet is the only outlier detection technique for multiple data streams that does not assume equality correlation among the data points from different streams.

- Wadjet works for asynchronous data points which is very important because synchronization is hard to achieve in many practical applications.

- Wadjet does not assume any fixed cross-correlation among the data points, rather it identifies the cross-correlation, if any.

- One big advantage of Wadjet is that Wadjet does not assume any relationship among the data points blindly, rather it explores the cross-correlation among the data points and detects the outlier-ness of a data point by comparing it to its cross-correlated data points only.

- Wadjet is the only outlier detection technique that detects an outlier based on both temporal correlation of a data point to the other data points from the same stream and cross-correlation of a data point to the other data points from the other data streams.

- Outlier detection for heterogeneous schemas is a novel problem. No existing algorithm works with a set of data points from multiple heterogeneous data streams.

- The time complexity of Wadjet is quadratic with respect to the number of streams and cubic with respect to the average number of dimensions in a stream. The space complexity of Wadjet is quadratic with respect to both the number of streams and number of dimensions.

- Both time and space complexity of Wadjet are independent of the number of data points. Hence, the memory and time usage of Wadjet do not depend upon the number of data points in a stream.

- The state of the art outlier detection technique DB-Outlier is not applicable to a dataset where data points from multiple streams are not equal, but correlated. DB-Outlier is also inapplicable for a dataset where data points are multi-dimensional and the number of dimensions is large.

- Wadjet is applicable and effective in both above cases. Moreover, Wadjet does not assume data points from multiple streams to be equal. Hence, the applicability of Wadjet is much broader compared to the existing outlier detection technique.

- Wadjet is perfectly applicable for the heterogeneous dataset that we have obtained from the SensorScope project [6] where the existing outlier detection fails to work.

- Wadjet scales well with respect to the number of streams in terms of accuracy. As the accuracy (precision, recall, and JC) of Wadjet is not affected by the number streams, Wadjet is able to handle a large number of streams without affecting its accuracy.

- Wadjet can handle multi-dimensional data points. Wadjet scales well with respect to the number of dimensions in terms of accuracy. We have performed experiment with 100 dimensional data points and this high number of dimensions does not negatively affect the accuracy of Wadjet.

- Both the number of streams and number of dimensions affect the execution time. The execution time increases with the increase of either number of streams or number of dimensions. Thus, adding more streams or dimensions may require more computational resources for Wadjet.

- The JC of Wadjet increases with the increase of percentage of outliers. Even if a dataset has a large number of outliers, Wadjet can detect them. In an application like intrusion detection, corrupting lots of data points by an intruder would not weaken the accuracy of Wadjet.

- Interestingly, a high percentage of outliers reduces the execution time due to our two phase outlier detection scheme. The first phase is insensitive to the percentage of outliers and detects many of them; thus the first phase reduces the burden on the second phase and the overall execution time improves with the increase of percentage of outliers.

- The accuracy (precision, recall and JC) of Wadjet is not very sensitive to the selection of the confidence interval. Thus, the user has freedom to choose an appropriate confidence interval from a wide range of values.

- The execution time of Wadjet increases with the decrease of the confidence interval. Therefore, the user must consider the execution time before choosing an appropriate confidence interval.

2. Future Research

Orion is a very effective outlier detection technique for multi-dimensional independent data streams. Although it is well scalable in terms of accuracy, its execution time increases with the increase of dimensions. In our future work we plan to design a more scalable version of Orion in terms of execution time.

Orion assumes a fixed arrival rate for all data points from the same stream. However, this assumption could be restrictive in real applications. Thus in our future work we would like to make our algorithm adaptive to the dynamic arrival rate of data streams.

Wadjet is just the first effort for outlier detection for heterogeneous data streams that may or may not be correlated. The type of cross-correlation we explored in Wadjet is very limited and hence in our future work we would like to explore more complex cross-correlations among the data points from multiple streams.

Wadjet deals with a cross-correlation matrix that captures the cross-correlations among all attributes in all streams. The computation of the cross-correlation matrix is the

bottleneck of Wadjet. In our future work we plan to design a more efficient approach to capture cross-correlations.

Our experiments are limited within the scope of large datasets. Each of our streams has approximately 50K data points and we use 100 streams. In future, we would like to conduct our experiment with Big data where we can use a larger number of heterogeneous streams.

REFERENCES

[1] B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom, "Models and Issues in Data Stream Systems," in *21st ACM Symposium on Principles of Database Systems*, pp. 1-16, 2002.

[2] N. Jiang and L. Gruenwald, "Research Issues in Data Stream Association Rule Mining," *ACM SIGMOD Record*, pp. 14-19, 2006.

[3] L. Golab and T. M. Özsu, "Issues in Data Stream Management," *ACM SIGMOD Record*, pp. 5-14, 2003.

[4] B. Liu, Classifying Data Streams Using a Concept Drifting Indicator, Norman: University of Oklahoma Thesis Library, 2006.

[5] S. Sadik, Outlier Detection for Data Streams, Norman, Oklahoma: The University of Oklahoma, 2010.

[6] G. Barrenetxea, F. Ingelrest, G. Schaefer and M. Vetterli, "The Hitchhiker's Guide to Successful Wireless Sensor Network Deployments," in *the 6th ACM Conference on Embedded Networked Sensor Systems*, Raleigh, pp. 43-56, 2008.

[7] S. Babu, L. Subramanian and J. Widom, "A Data Stream Management System for Network Traffic Management," in *Workshop on Network-Related Data Management*, Santa Barbara, California, 2001.

[8] V. Hodge and J. Austin, "A Survey of Outlier Detection Methodologies," *Artificial Intelligence Review*, vol. 22, pp. 85-126, October 2004.

[9] V. Chandola, A. Banerjee and V. Kumar, "Anomaly Detection: A Survey," *ACM Computing Surveys*, vol. 41, pp. 1-58, July 2009.

[10] V. Barnett and T. Lewis, Outliers in Statistical Data, New York: John Wiley & Sons, Inc.,, 1994.

[11] B. Z. J. L. Naoki Abe, "Outlier Detection by Active Learning," in *12th ACM SIGKDD international conference on Knowledge discovery and data mining*, Philadelphia, pp. 504-509, 2006.

[12] V. Chandola, A. Banarjee and V. Kumar, "Outlier Detection : A Survey," University of Minnesota, Minneapolis, 2007.

[13] S. Basu and M. Meckesheimer, "Automatic Outlier Detection for Time Series: An Application to Sensor Data," *Knowledge and Information System,* pp. 137-154, 2006.

[14] J. Xiao, Q. Zhuang, D. D. Baldocchi, B. E. Law, A. D. Richardson, J. Chen, R. Oren, G. Starr, A. Noormets, S. Ma, S. B. Verma, S. Wharton, S. C. Wofsy, P. V. Bolstadm and S. P. Burns, "Estimation of net ecosystem carbon exchange for the conterminous United States by combining MODIS and AmeriFlux data," *Agricultural and Forest Meteorology,* vol. 148, no. 11, pp. 1827-1847, 2008.

[15] "CIMIS Data," 2009. [Online]. Available: http://wwwcimis.water.ca.gov/cimis/data.jsp. [Accessed 14 April 2010].

[16] S. Stolfo, W. Fan, W. Lee, A. Prodromidis and P. Chan, "Cost-based modeling for fraud and intrusion detection: results from the JAM project," in *DARPA Information Survivability Conference and Exposition*, pp. 130-144, 2000.

[17] J. Lin, E. Keogh, A. Fu and H. Herle, "Approximations to Magic: Finding Unusual Medical Time Series," in *Computer-Based Medical Systems*, p. 329, 2005.

[18] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul and S. Zdonik, "Monitoring Streams: A New Class of Data Management Applications," in *the 28th international conference on Very Large Data Bases*, Hong Kong, pp. 215-226, 2002.

[19] A. Arasu, S. Babu and J. Widom, "An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations," dbpubs.stanford.edu:8090/pub/2002-57, Palo Alto, 2002.

[20] N. A. Chaudhry, "Introduction to Stream Data Management," in *Stream Data Management, Advances in Database Systems*, Springer, 2006, pp. 1-13.

[21] W. Lindner and J. Meier, "Towards a Secure Data Stream Management System," *Lecture Notes in Computer Science*, pp. 114-128, 2006.

[22] M. Stonebraker, U. Çetintemel and S. Zdonik, "The 8 Requirements of Real-time Stream Processing," *ACM SIGMOD Record,* vol. 34, no. December, pp. 42-47,

2005.

[23] E. M. Knorr and R. T. Ng, "A Unified Approach for Mining Outliers," in *the 1997 conference of the Centre for Advanced Studies on Collaborative research*, Toronto, p. 11, 1997.

[24] M. Breunig, H.-P. Kriegel, R. Ng and J. Sander, "LOF: Identifying Density-Based Local Outliers," in *2000 ACM SIGMOD International Conference on Management of Data*, pp. 93-104, 2000.

[25] S. Papadimitriou, H. Kitagawa, P. B. Gibbons and C. Faloutsos, "LOCI: Fast Outlier Detection Using the Local Correlation Integral," Intel Research Laboratory, Pittsburgh, 2002.

[26] I. Assent, P. Kranen, C. Baldauf and T. Seidl, "AnyOut: Anytime Outlier Detection on Streaming Data," in *Database Systems for Advanced Applications*, vol. 7238, S. Lee, Z. Peng, X. Zhou, Y. Moon, R. Unland and J. Yoo, Eds., Springer Berlin / Heidelberg, 2012, pp. 228-242.

[27] E. Eskin, "Anomaly Detection over Noisy Data using Learned Probability Distributions," in *Seventeenth International Conference on Machine Learning*, San Francisco, pp. 255-262, 2000.

[28] D. Agarwal, "An Empirical Bayes Approach to Detect Anomalies in Dynamic Multidimensional Arrays," in *5th IEEE International Conference on Data Mining*, Washington DC, pp. 26-33, 2005.

[29] N. Tatbul, "Streaming Data Integration: Challenges and Opportunities," in *IEEE ICDE International Workshop on New Trends in Information Integration*, Long Beach, CA, pp. 155-158 , 2010.

[30] A. Motro, "Management of Uncertainty in Database Systems," in *Modern Database Systems: The Object Model, Interoperability, and Beyond*, New York, ACM Press/Addison-Wesley Publishing Co., 1995, pp. 457-476.

[31] F. Angiulli and F. Fassetti, "Distance-based outlier queries in data streams: the novel task and algorithms," *Data Mining and Knowledge Discovery*, vol. 20, no. 2, pp. 290-324, 2010.

[32] K. Ishida and H. Kitagawa, "Detecting Current Outliers: Continuous Outlier Detection over Time-Series Data Streams," in *Lecture Notes in Computer*

Science, Berlin , Springer Berlin / Heidelberg, 2008, pp. 255-268.

[33] K. Beyer, J. Goldstein, R. Ramakrishnan and U. Shaft, "When Is "Nearest Neighbor" Meaningful?," in *Proceedings of the 7th International Conference on Database Theory*, London, UK, pp. 217-235, 1999.

[34] C. Aggarwal, A. Hinneburg and D. Keim, "On the Surprising Behavior of Distance Metrics in High Dimensional Spaces," in *Proceedings of the 8th International Conference on Database Theory*, London, UK, pp. 420-434, 2001.

[35] D. W. Scott, Multivariate Density Estimation: Theory, Practice, and Visualization, New York: John Wiley & Sons Inc., 1992.

[36] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki and D. Gunopulos, "Online Outlier Detection in Sensor Data Using Non-parametric Models," in *32nd international conference on Very large data bases*, Seoul, pp. 187-198, 2006.

[37] S. Papadimitriou, J. Sun and C. Faloutsos, "Streaming pattern discovery in multiple time-series," in *VLDB Endowment*, Trondheim, Norway, pp. 697 -708, 2005.

[38] W. Wu and L. Gruenwald, "Research issues in mining multiple data streams," in *Proceedings of the First International Workshop on Novel Data*, Washington, DC, pp. 56-60, 2010.

[39] C. Franke and M. Gertz, "ORDEN: Outlier Region Detection and Exploration in Sensor Networks," in *35th SIGMOD international conference on Management of data*, Rhode Island, pp. 1075-1078, 2009.

[40] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing and S. B. Zdonik, "the Design of the Borealis Stream Processing Engine," in *the 2nd Biennial Conference on Innovative Data Systems Research*, Asilomar, pp. 277-289, 2005.

[41] L. Gurgen, C. Roncancio, C. Labbé, A. Bottaro and V. Olive, "SStreaMWare: A Service Oriented Middleware for Heterogeneous Sensor Data Management," in *the 5th international conference on Pervasive services*, Sorrento, pp. 121-130, 2008.

[42] A. Dobra, M. Garofalakis, J. Gehrke and R. Rastogi, "Sketch-Based Multi-query Processing over Data Streams," *Advances in Database Technology, Lecture Notes in Computer Science,* pp. 551-568, 2004.

[43] J.-H. Hwang, S. Cha, U. Cetintemel and S. Zdonik, "Borealis-R: A Replication-Transparent Stream Processing System for Wide-Area Monitoring Applications," in *the 2008 ACM SIGMOD International Conference on Management of Data*, pp. 1303-1306, 2008.

[44] J. Krämer and B. Seeger, "Semantics and Implementation of Continuous Sliding Window Queries Over Data Streams," *ACM Transactions on Database Systems,* vol. 34, no. April, pp. 1-49, 2009.

[45] B. Mozafari, H. Thakkar and C. Zaniolo, "Verifying and Mining Frequent Patterns from Large Windows over Data Streams," in *the 2008 IEEE 24th International Conference on Data Engineering*, pp. 179-188, 2008.

[46] Y. Kim and U. Kim, "WSFI-Mine: Mining Frequent Patterns in Data Streams," *Advances in Neural Networks, Lecture Notes in Computer Science*, pp. 845-852, 2009.

[47] S. Sadik and L. Gruenwald, "DBOD-DS: Distance Based Outlier Detection for Data Streams," in *21st International Conference on Database and Expert Systems Applications*, Bilbao, Spain, pp. 122-136, 2010.

[48] S. Papadimitriou, J. Sun and C. Faloutsos, "Streaming pattern discovery in multiple time-series," in *31st international conference on Very large data bases*, pp. 697-708, 2005.

[49] E. M. Knorr and R. T. Ng, "Algorithms for Mining Distance-Based Outliers in Large Datasets," in *24rd International Conference on Very Large Data Bases*, pp. 392-403, 1998.

[50] E. Knorr, R. Ng and V. Tucakov, "Distance-based outliers: algorithms and applications," *the International Journal on Very Large Data Bases,* vol. 8, no. 3, pp. 237-253, 2000.

[51] S. Ramaswamy, R. Rastogi and K. Shim, "Efficient Algorithms for Mining Outliers from Large Data Sets," in *the 2000 ACM SIGMOD international conference on Management of data*, Dallas, Texas, USA, pp. 427-438, 2000.

[52] T. Zhang, R. Ramakrishnan and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases," in *1996 ACM SIGMOD international conference on Management of data*, Montreal, Canada, pp. 103-114 , 1996.

[53] D. M. Hawkins, Identification of Outliers, London: Chapman and Hall Ltd, 1980.

[54] F. Angiulli and F. Fassetti, "Detecting Distance-based Outliers in Streams of Data," in *Sixteenth ACM conference on Conference on information and knowledge management*, Lisbon, pp. 811-820, 2007.

[55] B. Sheng, Q. Li, W. Mao and W. Jin, "Outlier Detection in Sensor Networks," in *8th ACM international symposium on Mobile ad hoc networking and computing*, Montreal, pp. 219-228, 2007.

[56] K. Sequeira and M. Zaki, "ADMIT: Anomaly-based Data Mining for Intrusions," in *Eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, Edmonton; Alberta, pp. 386-395, 2002.

[57] P. K. Chan, M. V. Mahony and M. H. Arshad, "A Machine Learning Approach to Anomaly Detection," Florida Institute of Technology, Melbourne, 2003.

[58] J. Zhang, "Advancement of Outlier Detection: A Survey," *ICST Transactions on Scalabale Information Systems*, vol. 13, no. 01, 2013.

[59] S. Papadimitriou, H. Kitagawa, P. B. Gibbons and C. Faloutsos, "LOCI: Fast Outlier Detection Using the Local Correlation," in *19th International Conference on Data Engineering*, Bangalore, pp. 315-326, 2003.

[60] S. Sadik and L. Gruenwald, "Online outlier detection for data streams," in *Proceedings of the 15th Symposium on International Database Engineering & Applications*, Lisboa, Portugal, pp. 88-96, 2011.

[61] K. Ni, N. Ramanathan, M. N. H. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen and M. Srivastava, "Sensor Network Data Fault Types," *ACM Transactions on Sensor Networks*, vol. 5, no. May, pp. 1-29, 2009.

[62] X. Lu, T. Yang, Z. Liao, M. Elahi, W. Liu and H. Wang, "Incremental outlier detection in data streams using local correlation integral," in *Proceedings of the 2009 ACM symposium on Applied Computing*, New York, NY, USA, pp. 1520-1521, 2009.

[63] M. Shuai, K. Xie, G. Chen, X. Ma and G. Song, "A Kalman Filter Based

Approach for Outlier Detection in Sensor Networks," in *2008 International Conference on Computer Science and Software Engineering*, pp. 154-157, 2008.

[64] D.-I. Curiac, O. Banias, F. Dragan, C. Volosencu and O. Dranga, "Malicious Node Detection in Wireless Sensor Networks Using an Autoregression Technique," in *Third international conference on Networking and Services*, pp. 83-89, 2007.

[65] V. Puttagunta and K. Kalpakis, "Adaptive Methods for Activity Monitoring of Streaming Data," in *International Conference on Machine Learning and Applications*, Las Vegas, pp. 197-203, 2002.

[66] H. Madsen, Time Series Analysis, Boca Raton: Chapman & Hall/CRC, 2008.

[67] S. Efromovich, Nonparametric Curve Estimation – Methods, Theory, and Applications, New York: Springer-Verlag New York Inc., 1999.

[68] I. C. Oh, Forecasting Volatility, Norman: University of Oklahoma Thesis Library, 2004.

[69] N. N. Vijayakumar and B. Plale, "Missing Event Prediction in Sensor Data Streams Using Kalman Filters," in *Knowledge Discovery from Sensor Data*, Boca Raton, CRC Press, 2009, pp. 149-170.

[70] J. Laurikkala, M. Juhola and E. Kentala, "Informal Identification of Outliers in Medical Data," in *Fifth International Workshop on Intelligent Data Analysis in Medicine and Pharmacology*, Berlin, pp. 20-24, 2000.

[71] H. E. Solberg and A. Lahti, "Detection of Outliers in Reference Distributions: Performance of Horn's Algorithm," *General Clinical Chemistry*, pp. 2326-2332, 2005.

[72] S. Ando and E. Suzuki, "Detection of Unique Temporal Segments by Information Theoretic Meta-clustering," in *15th ACM SIGKDD international conference on Knowledge discovery and data mining*, Paris, pp. 59-68, 2009.

[73] R. Blender, K. Fraedrich and F. Lunkeit, "Identification of Cyclone-track Regimes in the North Atlantic," *Quarterly Journal of the Royal Meteorological Society*, pp. 565-579, 1997.

[74] R. T. Ng and J. Han, "Efficient and Effective Clustering Methods for Spatial Data Mining," in *20th International Conference on Very Large Data Bases*, pp. 144-

155, 1994.

[75] G. Sheikholeslami, S. Chatterjee and A. Zhang, "WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases," in *24rd International Conference on Very Large Data Bases*, pp. 428-439, 1998.

[76] W. Lee and D. Xiang, "Information-Theoretic Measures for Anomaly Detection," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, p. 130, 2001.

[77] C. Aggarwal and P. Yu, "Outlier detection for high dimensional data," in *2001 ACM SIGMOD international conference on Management of data*, Santa Barbara, CA, USA, pp. 37-46, 2001.

[78] I. Mechelen, H.-H. Bock and P. D. Boeck, "Two-mode Clustering Methods: A Structured Overview," *Statistical Methods in Medical Research*, vol. 3, no. 5, pp. 363-394, October 2004.

[79] D. Ashlock, "Designing Simple Evolutionary Algorithms," in *Evolutionary Computation for Modeling and Optimization*, Guelph, Ontario, Springer, 2006, pp. 33-65.

[80] S. Guha and N. Koudas, "Approximating a Data Stream for Querying and Estimation: Algorithms and Performance Evaluation," in *18th International Conference on Data Engineering.*, pp. 567-576, 2002.

[81] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss, "Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries," in *27th International Conference on Very Large Data Bases*, pp. 79-88, 2001.

[82] D. W. Scott, Multivariate Density Estimation: Theory, Practice, and Visualization, New York: John Wiley. &. Sons, Inc., 1992.

[83] L. Gruenwald, H. Chok and M. Aboukhamis, "Using Data Mining to Estimate Missing Sensor Data," in *the Seventh IEEE International Conference on Data Mining Workshops*, pp. 207-212, 2007.

[84] T. J. Brailsford, J. H. Penm and D. R. Terrell, "Selecting the Forgetting Factor in Subset Autoregressive Modelling," *Journal of Time Series Analysis*, pp. 629-650, 2002.

[85] J. Fan and J. S. Marron, "Fast Implementations of Nonparametric Curve

Estimators," *Journal of Computational and Graphical Statistics*, pp. 35-56, 1994.

[86] S. Papadimitriou, J. Sun and P. Yu, "Local Correlation Tracking in Time Series," in *Sixth International Conference on Data Mining*, pp. 456-465, 2006.

[87] Y. Zhu and D. Shasha, "StatStream: statistical monitoring of thousands of data streams in real time," in *28th international conference on Very Large Data*, Hong Kong, China, pp. 358-369, 2002.

[88] S. Burdakis and A. Deligiannakis, "Detecting Outliers in Sensor Networks Using the Geometric Approach," in *IEEE 28th International Conference on Data Engineering*, Arlington, Virginia USA, pp. 1108-1119, 2012.

[89] R. Taylor, "Interpretation of the Correlation Coefficient: A Basic Review," *Journal of Diagnostic Medical Sonography*, vol. 6, no. 1, pp. 35-39, 1990.

[90] V. Cangelosi, Basic Statistics: A Real World Approach, West, 1983.

[91] R. Johnson and D. Wichern, Applied Multivariate Statistical Analysis, Prentice Hall, 2007.

[92] D. H. West, "Updating mean and variance estimates: an improved method," *Communications of ACM*, vol. 22, no. September, pp. 532-535, 1979.

[93] M. Kutner, C. Nachtsheim, J. Neter and W. Li, Applied Linear Statistical Models, Boston: McGraw-Hill, 2005.

[94] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," in *the nineteenth annual ACM symposium on Theory of computing*, New York, New York, USA, pp. 1-6, 1987.

[95] "OU Supercomputer Resources," 2009. [Online]. Available: http://www.oscer.ou.edu/resources.php. [Accessed 17 May 2010].

[96] A. Frank and A. Asuncion, *UCI Machine Learning Repository*, Irvine, California: University of California, Irvine, 2010.

[97] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. Mcclung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham and M. Zissman, "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation," in *2000 DARPA Information Survivability Conference and Exposition*, pp. 12-26, 2000.

[98] A. Lazarevic and V. Kumar, "Feature bagging for outlier detection," in *ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 157-166, 2005.

[99] A. Tsymbal, "The problem of concept drift: definitions and related work," Department of Computer Science, Trinity College Dublin, Dublin, Ireland, 2004.

[100] C. Franke and M. Gertz, "Detection and Exploration of Outlier Regions in Sensor Data Streams," in *2008 IEEE International Conference on Data Mining Workshops*, Washington, DC, pp. 375-384, 2008.

[101] V. Niennattrakul, E. Keogh and C. A. Ratanamahatana, "Data Editing Techniques to Allow the Application of Distance-Based Outlier Detection to Streams," in *the 2010 IEEE International Conference on Data Mining*, Washington, DC, USA, pp. 947-952, 2010.

[102] R. Kistler, E. Kalnay, W. Collins, S. Saha, G. White, J. Woollen, M. Chelliah, W. Ebisuzaki, M. Kanamitsu, V. Kousky, H. v. d. Dool, R. Jenne and M. Fiorino, "The NCEP–NCAR 50-Year Reanalysis," *Bulletin of the American Meteorological Society*, pp. 247-267, 2001.

[103] E. Keogh, J. Lin and W. Truppel, "Clustering of Time Series Subsequences is Meaningless: Implications for Previous and Future Research," in *Third IEEE International Conference on Data Mining*, p. 115 , 2003.

[104] W. P. Elderton and N. L. Johnson, System of Frequency Curves, London: Cambridge University Press, 1969.

[105] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein and R. Varma, "Query Processing, Resource Management, and Approximation ina Data Stream Management System," Stanford InfoLab, 2002.

[106] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss and M. A. Shah, "TelegraphCQ: Continuous Dataflow Processing," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, San Diego, p. 668, 2003.

[107] C. C. Aggarwal, J. Han, J. Wang and P. S. Yu, "On Demand Classification of Data Streams," in *Proceedings of the tenth ACM SIGKDD international*

conference on Knowledge discovery and data mining, Seattle, pp. 503-508, 2004.

[108] G. Hulten, L. Spencer and P. Domingos, "Mining Time-changing Data Streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, San Francisco, pp. 97-106, 2001.

[109] M. Last, "Online Classification of Nonstationary Data Streams," *Intelligent Data Analysis*, vol. 6, no. April, pp. 129-147, 2002.

[110] D. J. Abadi, D. P. Carney, U. Cetintemel, M. F. Cherniack, C. Convey, C. Erwin, E. Galvez, M. Hatoun, A. S. Maskey, A. Rasin, A. Singer, M. R. Stonebraker, N. Tatbul, Y. Xing, R. Yan and S. B. Zdonik, "Aurora: A Data Stream Management System," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, San Diego, p. 666, 2003.

[111] I. Botan, G. Alonso, P. M. Fischer, D. Kossmann and N. Tatbul, "Flexible and Scalable Storage Management for Data-intensive Stream Processing," in *the 12th International Conference on Extending Database Technology: Advances in Database Technology*, Saint Petersburg, pp. 934-945, 2009.

[112] M. Cammert, J. Kramer, B. Seeger and S. Vaupel, "A Cost-Based Approach to Adaptive Resource Management in Data Stream Systems," *IEEE Transactions on Knowledge and Data Engineering*, pp. 230-245, 2008.

[113] S. Chen, H. Wang, S. Zhou and P. S. Yu, "Stop Chasing Trends: Discovering High Order Models in Evolving Data," in *the 2008 IEEE 24th International Conference on Data Engineering*, pp. 923-932, 2008.

[114] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu and M. Doo, "SPADE: The System S Declarative Stream Processing Engine," in *the 2008 ACM SIGMOD international conference on Management of data*, pp. 1123-1134, 2008.

[115] C. Heinz, J. Kramer, T. Riemenschneider and B. Seeger, "Toward Simulation-Based Optimization in Data Stream Management Systems," in *the 2008 IEEE 24th International Conference on Data Engineering*, pp. 1580-1583, 2008.