In this research study, we implemented in Java a density-based outlier detection algorithm called Orion, evaluated its performance, and compared the results with several other outlier detection algorithms for data streams.

**Introduction**

An outlier is defined as an observation that has different values compared to other observations in a dataset. Outlier detection algorithms for data streams have a vital role in Big Data applications. In these applications, data points arrive over time in a window, which makes the problem of detecting outliers becomes more complex due to the special characteristics of data streams.

Orion is a density-based outlier detection algorithm for data streams [1]. Orion detects outliers through three phases:

1. Find a p-dimension Z0 that minimizes the stream density of a data point Dn using an evolutionary algorithm.
2. Compute the stream density and k-integral metrics of Dn projected on Z0:
   - The stream density of a data point Dn along a p-dimension Z0 is the percentage of neighbors of Dn within the scaled neighborhood distance.
   - The k-integral of Dn is the integral that includes k percent of the data points along Z0.
3. Perform co-clustering based on the two metrics to detect the outliers.

**Methods**

We implemented Orion in Java with JDK 11. We used the Watchmaker framework for evolutionary computation in phase two and the Expectation-Maximization (EM) clustering algorithm provided by WEKA framework in phase three [2].

We used two datasets from the UCI machine learning repository and OpenML Repository for performance analysis:

1. Mulcross dataset (282,144 records, 4 dimensions)
2. Tao dataset (575,468 records, 3 dimensions)

We compared Orion with three existing distance-based outlier detection algorithms: Abstract-C, Micro-Cluster Based Algorithm (MCOD), and Approximate-Storm [3][4].

We measured the outlier detection performance in terms of precision, recall, and Jaccard coefficient (JC). In addition, we also studied the execution time of Orion to gain more insight into its scalability.

**Comparison of Orion and Other Algorithms**

We evaluated the performance of Orion, compared to existing algorithms, using a suite of data streams. The results were evaluated using precision, recall, and Jaccard coefficient.

**Performance Evaluation of Orion**

Although Co-Clustering is shown as the most expensive operation in figure 5, the operation only runs once for each window. On the other hand, evolutionary computation, stream density estimator, and k-integral estimator are executed for each data point in a window.

This makes evolutionary algorithm the most expensive operation inside Orion, which contributes to 80.12% of the total execution time for each window on average.

**Discussions & Conclusions**

- The results indicate that Orion has better performance than Abstract-C, Approximate-Storm, and Micro-Cluster, especially in the case of the Mulcross dataset.
- Although Orion has a very low recall rate on the Tao dataset, it has a significantly higher precision rate than other algorithms, which means that the amount of normal data points falsely predicted as anomaly by Orion is extremely low. On the other hand, low recall rate suggests that Orion failed to detect most of the outliers in the dataset.
- The performance evaluation indicates that Orion is not scalable. The evolutionary computation (phase 1) is the most expensive phase and it seems to have a non-linear growth rate.
- Any further work on optimizing Orion need to focus on reducing the growth rate of the first phase. Solving the non-scalable issue of Orion could use GPU parallel computing to increase performance.

**References**