**Similarity Join Over Multiple Streams of Time-Series Data**

**Source code and presentation slides:** [**https://github.com/acash11/Big-Data-Analytics-Final**](https://github.com/acash11/Big-Data-Analytics-Final)

Andrew Cash - [acash11@kent.edu](mailto:acash11@kent.edu)

Erik Wood – [ewood21@kent.edu](mailto:ewood21@kent.edu)

**1. Introduction**

The Similarity Join Query is useful for finding correlations between different streams of data. Specific use cases include cleaning sensor data, finding patterns in stock prices, and internet traffic analysis.

The similarity join query (SJ) in the context of multiple time-series data streams has multiple applications in the field of data engineering. Given two time-series databases *R* and *S*, an SJ query will return all possible pairs of subsequences, <r, s>, in which the distance between r and s is less than or equal to a threshold variable, ε (epsilon). To dynamically maintain stream data, a sliding window structure is used, where old data is pushed out as new data is pushed in. To calculate each similar subsequence within each time series, a variety of approaches can be used. The simple one is brute force, where every possible subsequent is compared. Naturally, this is lacking in performance.

A more novel way may use indexing and dimensionality reduction. Fast subsequence matching in time-series databases (<https://dl.acm.org/doi/pdf/10.1145/191843.191925>) suggests using a sliding window over each stream of data, Si, and capturing its features using Discrete Fourier Transform. This is then used to index an R\*-tree, which allows for efficient access of the original data.

**2. Project Description**

Our method is a little bit different. The goal was to index each non-query series, Si, using a B+-tree. The B+-tree would allow for range searches over the data for effective pruning, and even supports the nature of time-series data, since the leaf nodes comprise a linked list.

Consider the first point in a query sequence q. If this point is the first coordinate in a sliding window, then any similar points must be within at least Epsilon distance away from it. This can be proven by considering the most extreme case of a legal similarity join. To calculate the distance between subsequences, we take the sum of their corresponding coordinator points. If one point is Epsilon distance away from its corresponding point, then all other points must be equal to their corresponding points. Essentially, any points past the radius Epsilon can be pruned.

Challenges and technical contributions (new problems or new solutions?) In our project we are running multiple similarity joins in real time on a data stream to queries results that have similarities.

The workload distribution for each member in your team: Is going to be 50/50 split because we only have two people in our group.

**3. Background**

Related papers: *Efficient Similarity Join Over Multiple Stream Time* Series by Xiang Lian, Student Member, IEEE, and Lei Chen, Member, IEEE

Software tools: Python, NumPy, Pandas, Scipy, Matplotlib.

Required hardware: A computer to run the queries, and a stream of data.

Related programming skills: Ability to use python and libraries in python.

**4. Problem Definition**

Formal (mathematical) definitions of problems: Whenever T\_i gets a new data item T\_i[t+1], Similarity join obtains and then outputs all subsequences from stream time series T\_i(1 <= j <= m) that epsilon-match with the new subsequence S\_new = T\_i[t - n + 2 : t + 1] from T\_i, where n = s \* w.

Challenges of tackling problems: Ease of implementing mathematical problems and getting results to be accurate.

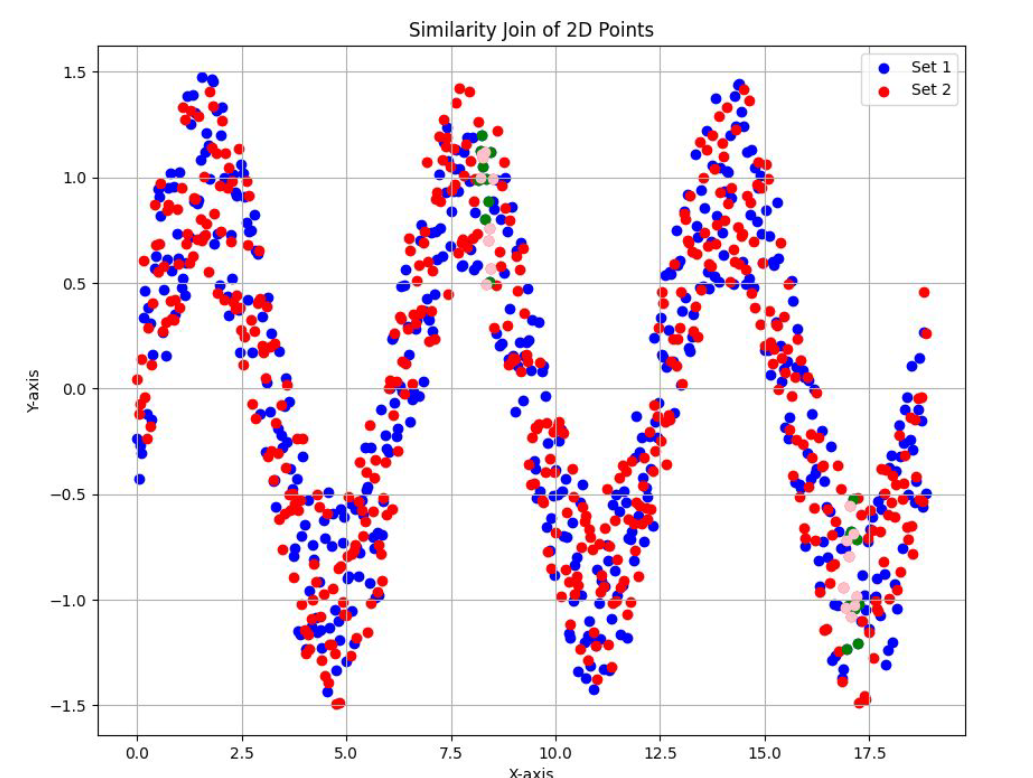
A brief summary of general solutions in your project: Taking a data stream and running a query over two time-series databases and returning all possible pairs of the subsequent data.

**5. The Proposed Techniques:**

Storing our data set in a B+-tree to allow efficient range searches and sequential access. This should provide a large pruning power for the similarity join.

**6. Visual Applications:**

For this project, we used matplotlib to show our data with an interactive graph. It clearly shows the similarity join on well-represented sets of data.



**7. Experimental Evaluations:**

Experimental Evaluations: Our data was a replication of a sine wave, which fits into the use case of the similarity join. The brute force algorithm outperformed our implementation, however with better optimization it should be able to be faster due to its pruning power.

Our parameters are a Threshold variable for similarity, The number of points generated in the data set, the noise factor in the creation fo the data set, and the size of the sliding window for the similarity join

**8. Future Work:**

Applying these concepts to live-time data streams, and creating a structure that can handle many real time updates.

**9. References:**

Efficient Similarity Join Over Multiple Stream Time Series by Xiang Lian, Student Member, IEEE, and Lei Chen, Member, IEEE.

Fast Subsequence Matching in Time-Series Database by Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos