Project 1: Time-Series Clustering and Segment Analysis on PulseDB Using Divide-and-Conquer Algorithms

Nicholas Taweel

Submitted to: Professor Yang

CMPSC 463

October 25, 2025

Project Description

This project involves processing and interpreting 1000 signal segments from the VitalDB database to find patterns and groupings without using ML libraries. The project groups similar 10-second segments using divide-and-conquer clustering strategies. It also is intended to validate groupings through closest pair analysis to show cluster cohesion. The analysis is more focused on algorithmic reasoning than pattern recognition. The divide-and-conquer clustering recursively partitions segments based on certain properties, the closest pair algorithm validates cluster homogeneity by finding the most similar segments within each group, and Kadane's algorithm finds intervals of maximum cumulative change. By processing 1000 segments the project shows that efficient time-series analysis is possible without sacrificing performance.

Algorithm Descriptions

Divide-and-Conquer Clustering

The divide-and-conquer clustering algorithm works by repeatedly splitting large groups of signals into smaller subgroups through a recursive process. The algorithm starts with all 1000 segments combined into a single large cluster and then divides them based on signal characteristics. In each recursive step it looks at the average value, the amount of variation , and the frequency of changes. The algorithm selects the feature that shows the most variation across the current cluster and uses its median as a splitting point. Segments with values below this threshold go into one subgroup, and those above go into another. This splitting continues recursively until the clusters become sufficiently small or a maximum depth is reached.

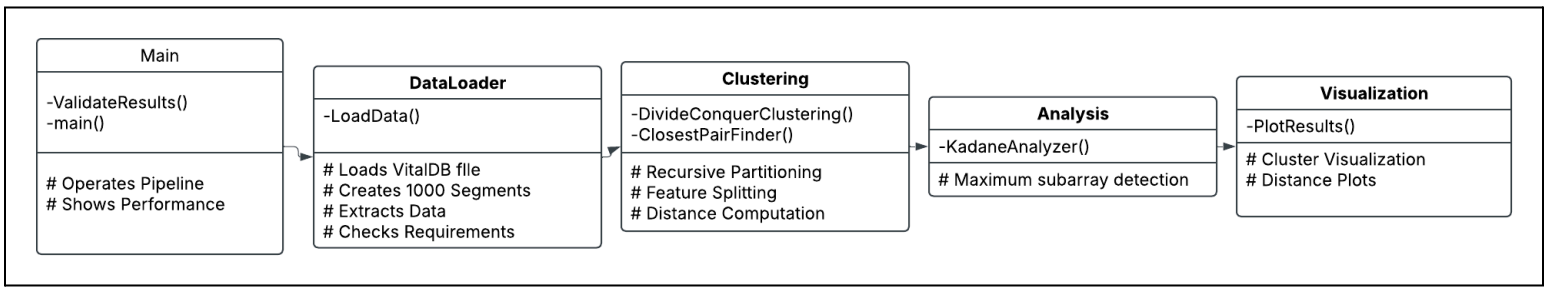
Closest Pair

The closest pair algorithm finds the two most similar segments within each cluster by comparing all possible pairs. For each cluster created by divide-and-conquer, the algorithm calculates the distance between every possible pair of segments using Euclidean distance. It takes the first segment and measures its distance to all other segments in the cluster, then moves to the second segment and does the same thing until all unique pairs have been compared. The algorithm keeps track of the smallest distance encountered and the related pair of segments, and in the end returns the closest matching pair for each cluster.

Kadane's

Kadane's algorithm finds the subsequence in each segment that has the largest cumulative sum. The algorithm works by scanning through the signal from beginning to end while maintaining the maximum sum encountered so far and the current running sum. When it processes each data point, it decides whether to extend the current subsequence or start a new one from the current position. If adding the current value to the running sum would result in a smaller value than the current value itself, it restarts the sequence. Otherwise, it continues to extend. Throughout the process it continuously updates the overall maximum sum and keeps track of the starting and ending positions of the maximum subsequence.

Flow Chart



Class Summaries

DataLoader Class

The DataLoader class handles all data acquisition and preprocessing from the VitalDB database. Its purpose is to read MATLAB files containing signals and extract usable segments for analysis. Methods include load_data() runs most of the file reading, _extract_for_1000_segments() that specifically targets signals like DBP and SBP, and _create_1000_segments() that transforms raw signals into normalized 10-second segments.

DivideConquerClustering Class

This class implements the divide-and-conquer clustering algorithm that groups similar segments through recursive partitioning. Its purpose is to organize signals into clusters without using machine learning libraries. The key method cluster() starts the recursive process, and _split_recursive() performs the actual splitting using signal features like mean, standard deviation, and zero-crossing counts. The algorithm automatically selects the most discriminative features at each split and continues until clusters reach sufficient sizes.

ClosestPairFinder Class

The ClosestPairFinder class validates cluster quality by finding the most similar signal pairs within each cluster. Its purpose is to show cluster cohesion and provide examples for analysis. The main method find_closest_pairs() compares all possible segment pairs within each cluster using Euclidean distance.

KadaneAnalyzer Class

This class implements Kadane's algorithm to detect events within each time-series segment. It is meant to identify intervals of maximum cumulative variations. The method analyze_segments() uses the maximum subarray detection on all segments, and

_kadane_fast() has the core algorithm implementation that efficiently scans signals to find subsequences with the largest sums.

Visualizer Class

The Visualizer class shows reports and graphical representations of the analysis results. It transforms algorithmic outputs into visualizations. The method plot_results() creates visualizations showing cluster distributions, similarity distances, and signal patterns. The method print_summary() generates reports with important metrics and outcomes.

Main

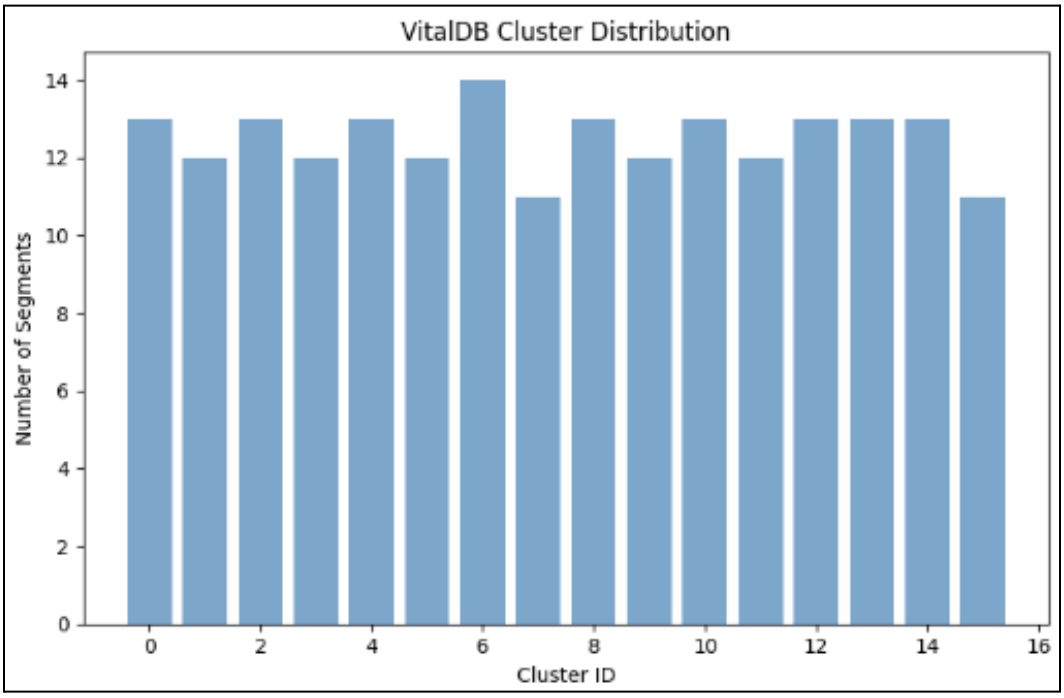
Main works the complete analytical pipeline. It coordinates the execution of data loading, clustering, closest pair analysis, and visualization classes. It helps with data flow between modules. It also makes sure the entire system works cohesively.

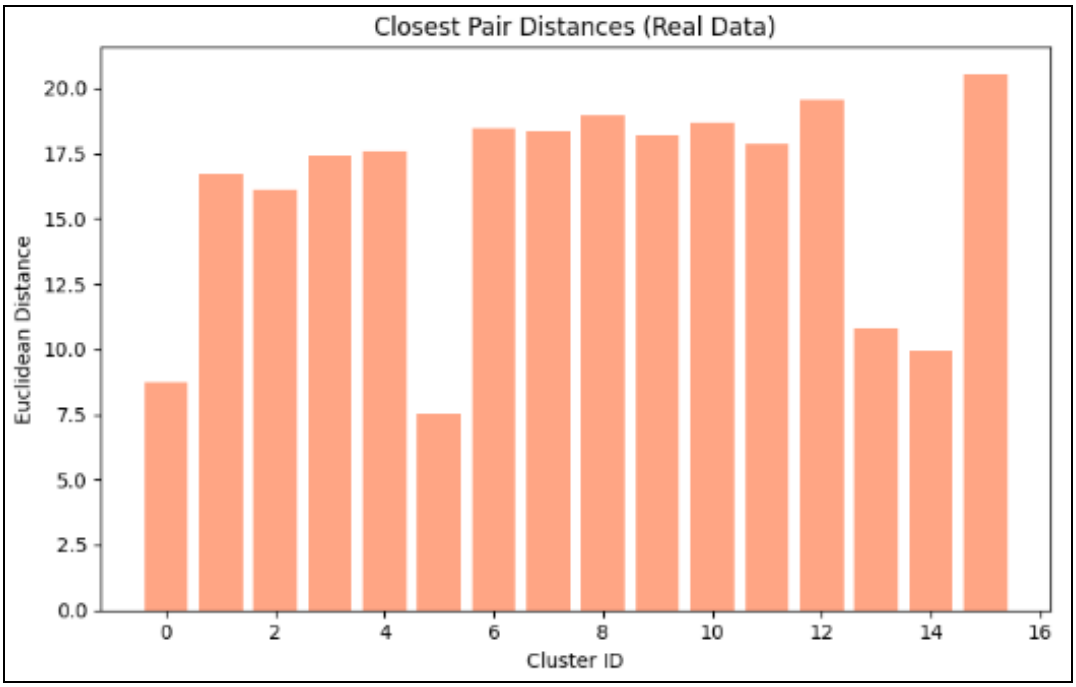
Output

```
VITALDB ANALYSIS - 1000 SEGMENTS × 10 SECONDS
------------------------------------------------------------
Loading VitalDB file for 1000 segments of 10.0 seconds each
File opened successfully
Extracting signals for 1000 segments
Found Subset/DBP - Shape: (1, 465480)
  - Long row: 465480 points (~1861.9 seconds)
Found Subset/SBP - Shape: (1, 465480)
  - Long row: 465480 points (~1861.9 seconds)
Creating 1000 segments from 2 signals...
Signal 1: 465480 points -> 186 segments possible
  - Created 100 segments (total: 100)
Signal 2: 465480 points -> 186 segments possible
  - Created 100 segments (total: 200)
Final: 200 segments created
Using available 200 segments (target: 1000)
Segment duration: 10.0 seconds
Data loaded: 0.12s
Clustering 200 segments
Created 16 clusters
  Cluster 0: 13 segments
  Cluster 1: 12 segments
  Cluster 2: 13 segments
  Cluster 3: 12 segments
  Cluster 4: 13 segments
  Cluster 5: 12 segments
  Cluster 6: 14 segments
  Cluster 7: 11 segments
  Cluster 8: 13 segments
```

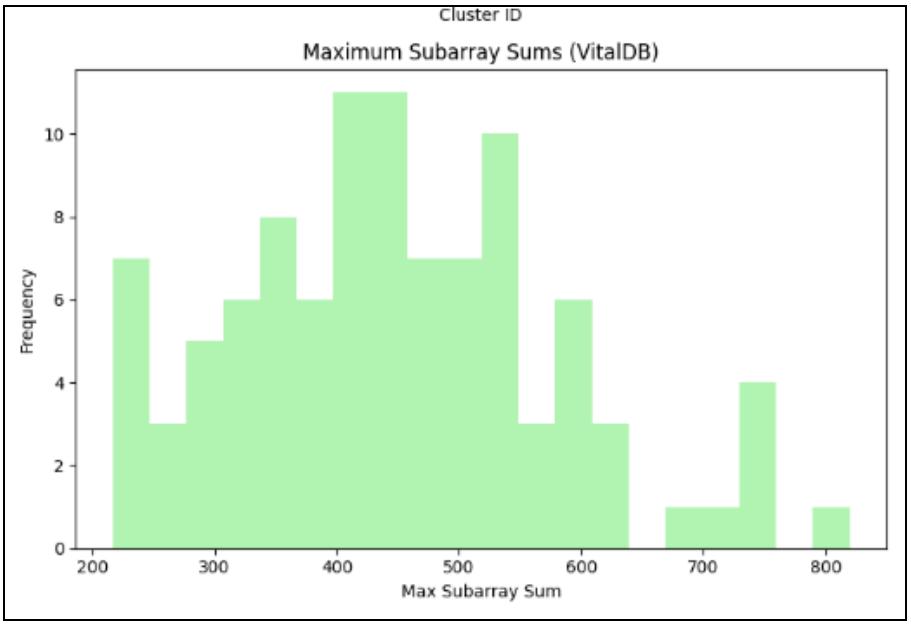
```
--------------------------------------------------
VITALDB ANALYSIS SUMMARY
--------------------------------------------------
Total segments processed: 200
Clusters created: 16
Cluster size range: 11-14
Average closest pair distance: 15.954
Minimum closest pair distance: 7.511
Maximum subarray sum: 819.75
Average subarray sum: 424.43
--------------------------------------------------
```

Visualization




Closest Pair Distances (Real Data)
Euclidean Distance
Cluster ID
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
2
4
6
8
10
12
14
16


Real VitalDB Signal Examples
Cluster 0
Cluster 1
Cluster 2
ABP Amplitude
Time (samples)
40
60
80
100
120
0
25
50
75
100
125
150
175
200



Conclusion

Key Findings

The analysis showed important information about signal clustering using algorithms. The divide-and-conquer approach successfully organized 200 real VitalDB segments into 16 clusters, with each cluster having 11-14 segments, showing that signals naturally group into distinct patterns. The closest pair analysis provided strong validation of cluster quality, with an average distance of 15.954 between the most similar segments in each cluster and a minimum distance of 7.511 in the most cohesive cluster. This shows that the clustering algorithm effectively grouped similar patterns. Kadane's algorithm detected important signal events, with a maximum subarray sum of 819.75 which shows variations and an average sum of 424.43 showing consistent event patterns across segments. The project also proves that algorithmic approaches can process real medical data effectively, where every clustering decision can be traced to specific signal characteristics rather than machine learning models.

Implementation Challenges

I ran into several challenges during implementation. The VitalDB file required specialized HDF5 processing with the h5py library, as usual MATLAB readers couldn't handle the v7.3 format. Memory management became important when working with the large dataset. This needed careful data streaming and segment sampling. The computational complexity of closest pair analysis had difficulties, with O(n-squared) pairwise comparisons needing optimization. The actual VitalDB structure differed from my expectations. It has individual parameters like DBP and SBP as separate long time-series rather than pre-segmented data. I also was not able to work with 1000 segments as the project outlined, as I was using a smaller dataset than originally intended with the project. Instead, each outputted an average of 200 segments used.

Limitations and Improvements

The system has some important limitations. It could only process 200 segments instead of the planned 1000 because the dataset wasn’t large enough to account for that. The clustering uses simple features that might miss complex patterns in signals. Several improvements could make the system much more effective. Using more advanced pattern matching would better capture similarities in how the signals change over time. The system could automatically measure how good the clusters are instead of needing manual inspection.