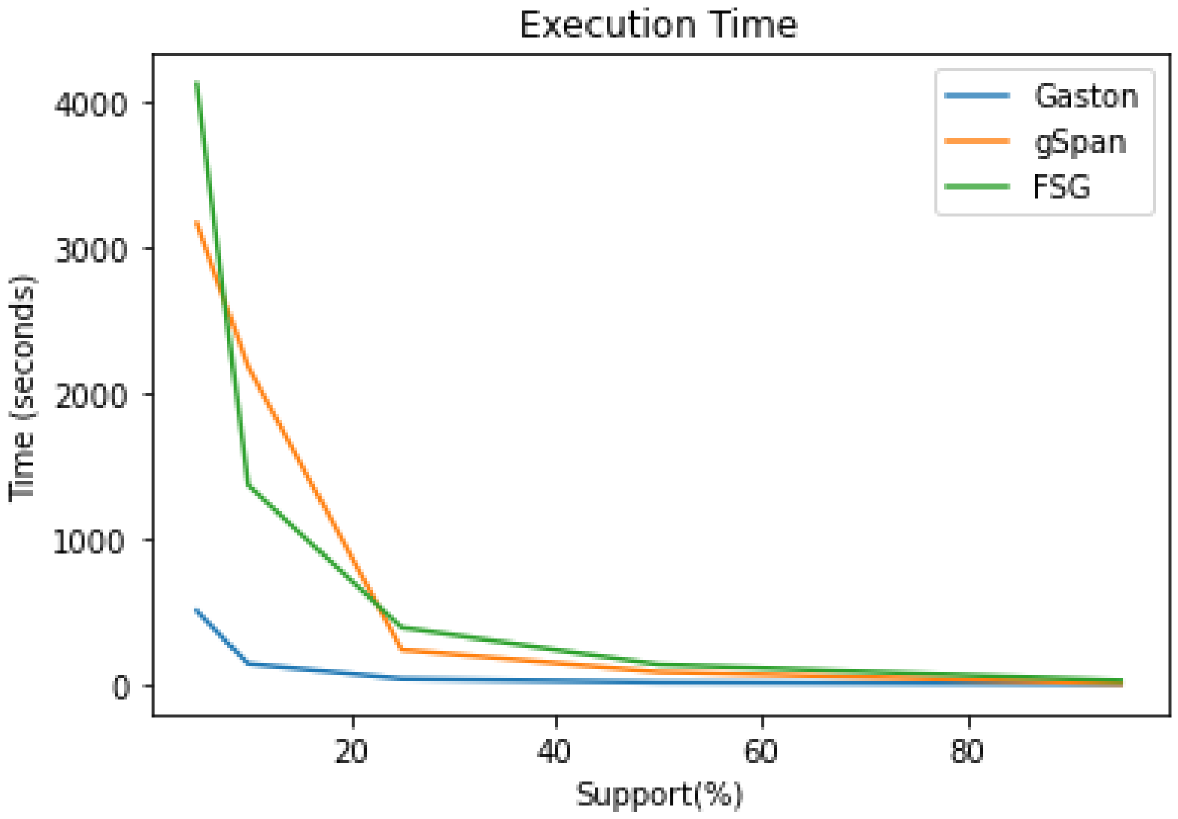
COL761
Assignment 2

Suditi Laddha
2023AIB2065
Anant Kumar Sah
2023AIB2068
Ajay Kumar Meena
2023AIB2083

September 24, 2023

1. Frequent Subgraph Mining



Why is Gaston faster than gSpan and FSG

1. **G**aston employs lexicographic order traversal for efficient exploration of the search space. It uses a compressed graph database representation, reducing memory requirements and improving I/O operations. Gaston typically includes optimized subgraph isomorphism algorithms for faster checking.Depending on the implementation, and can be parallelized for improved performance on multi-core processors or distributed systems. It uses efficient data structures and a compressed database representation, resulting in lower memory consumption and Gaston employs incremental techniques to avoid redundant computations during subgraph generation.

Why gSpan is Faster Than FSG (Frequent Subgraph Mining)

gSpan is often considered faster than FSG (Frequent Subgraph Mining) due to several key design features and optimizations that contribute to its improved efficiency:

1. **g**Span employs a BFS-based exploration strategy to generate frequent subgraphs. This approach allows gSpan to discover frequent subgraphs level by level, starting from smaller subgraphs and gradually expanding to larger ones. BFS can be more efficient than the depth-first search (DFS) strategy used by FSG, especially when searching for frequent subgraphs in a breadth-first manner.gSpan uses canonical labeling techniques to compare and prune redundant subgraphs efficiently. Canonical labeling assigns unique labels to isomorphic subgraphs, enabling gSpan to avoid

exploring isomorphic duplicates and reducing the search space dramatically.The level-wise search strategy of gSpan ensures that it explores frequent subgraphs of a particular size before moving on to larger ones. This helps in pruning the search space early, preventing the generation of unnecessary candidates, and reducing computational overhead. gSpan employs a compact data structure to store the graph database and frequent subgraphs. This efficient data structure minimizes memory consumption and allows for faster subgraph discovery. gSpan incorporates various pruning techniques based on the Apriori property, which helps eliminate infrequent subgraphs from consideration early in the search process. These pruning techniques reduce the number of candidates to be explored, contributing to its speed. gSpan uses a specialized graph database format that organizes the input graphs in a way that facilitates frequent subgraph discovery. This format enables efficient retrieval and manipulation of subgraphs, further improving performance. Depending on the implementation, gSpan can be parallelized effectively to take advantage of multi-core processors or distributed computing environments.

Conclusion:

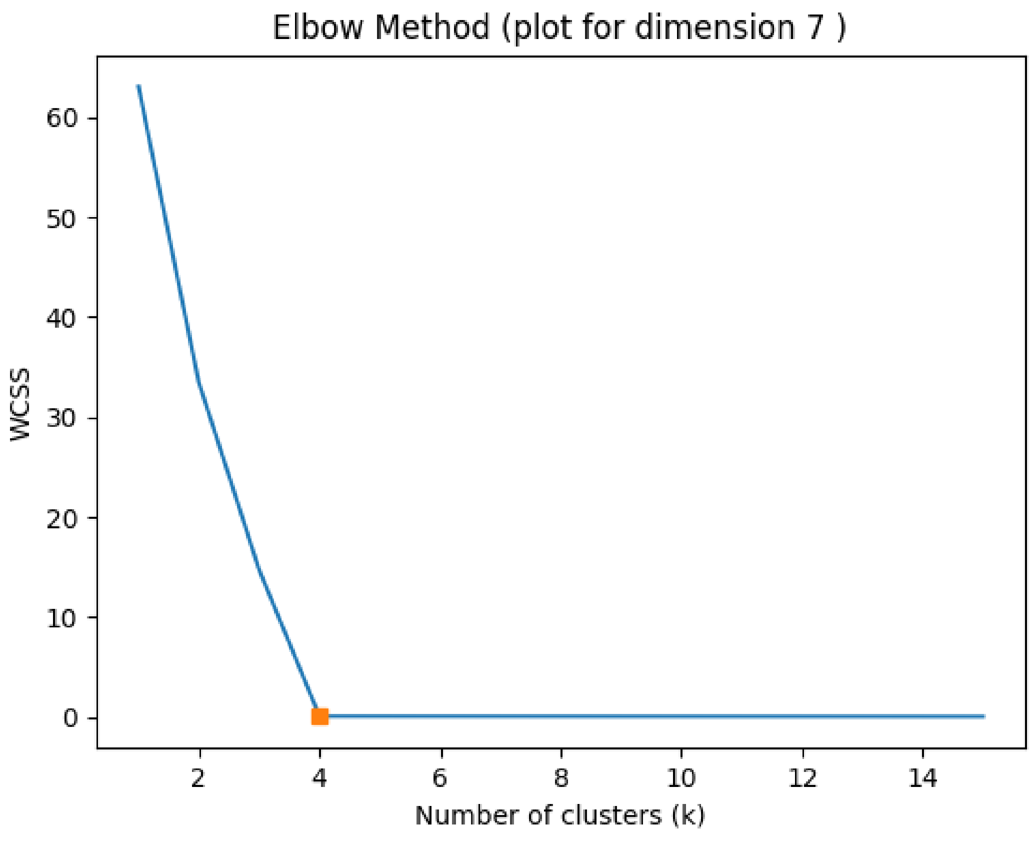
1. **F**ig illustrates the runtime of gSpan and FSG as support varies from 5 percent to 95 percent where time in seconds is marked on the y axis and the support value in percentage is marked in the x axis. For FSG, when the support is less than 5 percent, the process is aborted either because the main memory is exhausted or the runtime is too long. Fig-1 shows gSpan achieves better performance by 15-100 times in comparison with FSG.

2. gSpan uses a new lexicographic ordering system and a depth-first search-based mining algorithm for efficient mining of frequent subgraphs in large graph database.Fig-1 shows that gSpan outperforms FSG by an order of magnitude and is capable to mine large frequent subgraphs in a bigger graph set with lower minimum supports than previous.

3. Gaston tends to perform better than FSG or gSpan on large and complex datasets where the number of nodes, edges, and subgraphs is substantial.This is also evident from fig-1. And this is due to that its graph database approach allows for efficient indexing and retrieval of subgraphs, making it suitable for datasets with a high degree of complexity. When frequent subgraphs are to be mined with a very low minimum support threshold (minSup), Gaston can be advantageous.

4. So in conclusion gSpan is well-suited for scenarios where you're primarily interested in finding highly frequent, simple subgraph patterns in relatively small to medium-sized datasets. Gaston's strengths lie in handling large and complex datasets, complex subgraph patterns, low support thresholds, and query-driven mining.

2. k-Means Clustering



1. k-means is an unsupervised machine learning algorithm used for clustering data points into groups or clusters based on their similarity. One of the critical parameters in k-means is the number of clusters (k) which is not known in apriori. Hence an elbow plot which is a visual tool was used to determine the optimal value of k for K-means clustering.

2. WCSS Calculation: For each value of k, the algorithm calculates the Within Cluster Sum of Squares Diatances(WCSS) between data points and their assigned cluster centroids. Within Cluster Sum of Squares measures how well data points are grouped around their respective centroids. Smaller WCSS values indicate tighter and more cohesive clusters.

3. Creating the Elbow Plot: Kmeans Module of Sckikit-learn library was used for Kmeans clustering. We ran K-means algorithm for a range of k values from a range 1,...,15 and calculated the corresponding WCSS for each k. The x-axis of the plot represents the values of k (the number of clusters), while the y-axis represents the corresponding SSD values.

4. Elbow Shape: As the value of k was increased, the WCSS typically decreased because more centroids were introduced(The plot often shows a decreasing curve). The ”elbow” in the plot is the point where the rate of decrease in WCSS starts to slow down. The point where the curve forms an ”elbow” shape is often considered

the optimal value of k. In our case the elbow point was different for different dimensional data sets. The figure shows the plot for a 7 dimensional data set and the elbow point can clearly be interpreted to be 4.

In some cases, say when the data set was 4 or 5 dimensional there was no clear and distinct elbow in the plot, making it challenging to choose an optimal k. In such situations, expert judgment and context are essential.

The elbow plot is a valuable tool for selecting an appropriate number of clusters in K-means, but it should be used in conjunction with other evaluation methods and domain knowledge to ensure meaningful and interpretable clustering results.

3. Dendogram

Fastest Algorithm

The fastest possible algorithms for single linkage agglomerative clustering are of order $O(n^2)$. One of these algorithms is the SLINK algorithm. It computes the dissimilarity matrix once or takes the dissimilarity matrix as input. The dissimilarity matrix stores all the dissimilarity measures between any two clusters. Here, the dissimilarity measure is the distance between two clusters.

As the dissimilarity matrix is constructed, an array which contains the minimum distance of a cluster from the first $n-1$ clusters is also constructed. Another array which contains the IDs of the clusters which are at the minimum distance from the initial $n-1$ clusters are also stored.

Because of these two distance and cluster ID arrays, the computation time of the algorithm reduces to $O(n^2)$ as the whole of dissimilarity matrix is not scanned again in any iteration. For a given iteration, only 2 rows of dissimilarity matrix are scanned to determine the new single linkage distance of the cluster formed from all other clusters. This process continues till all clusters are combined.

The Modified SLINK Algorithm is as follows:

```
$A_D \leftarrow \{\}$
$A_V \leftarrow \{\}$
**for** $i \in \{1, 2, ..., n\}$ **do**
    **for** $j \in \{1, 2, ..., n\}$ **do**
        // Calculate distance of all clusters (single points) from each other
        $d_{ij} \leftarrow$ distance between $c_i$ and $c_j$
    **end for**
    // Store minimum distance from cluster and minimum distance cluster ID
    $A_D \leftarrow A_D \cup \{\min_{j \neq i}\{d_{ij}\}\}$
    $A_V \leftarrow A_V \cup \{arg\min_{j \neq i}\{d_{ij}\}\}$
**end for**
$t \leftarrow 0$
**while** $t < n$ **do**
    // ID of one of two closest clusters
    $i \leftarrow arg\min A_D$
    // ID of closest cluster from $i$
    $j \leftarrow A_V[i]$
    // Update all clusters whose minimum distance cluster is $j$ to $i$
    **for** $k \in \{l | A_V[l] = j\}$ **do**
        $A_V[k] \leftarrow i$
    **end for**
    // Combine $j$ and $i$ by updating their single linkage distance from other clusters
    $d_{ik} \leftarrow \min\{d_{ik}, d_{jk}\} \forall k$
    $d_{ki} \leftarrow d_{ik} \forall k$
    // Update dissimilarity matrix
    remove $d_{jk}$ and $d_{kj} \forall k$
    // Update minimum dissimilarity and closest cluster arrays
    $A_D[j] \leftarrow \infty$
    $A_D[i] \leftarrow \min\{d_{ik} | d_{ik} \neq 0\}$
    $t \leftarrow t + 1$
end while
```
This algorithm take $O(n^2)$ time to construct the dissimilarity matrix, $O(n)$ time to create the minimum distance and the closest cluster arrays. For each iteration, the algorithm scans just two rows of the matrix, which is decreasing in size. Thus for all $n-1$ iterations, the time taken is less than $O(n^2)$. Thus, the time complexity of this algorithm is $O(n^2)$.

Implementation

Given following points:

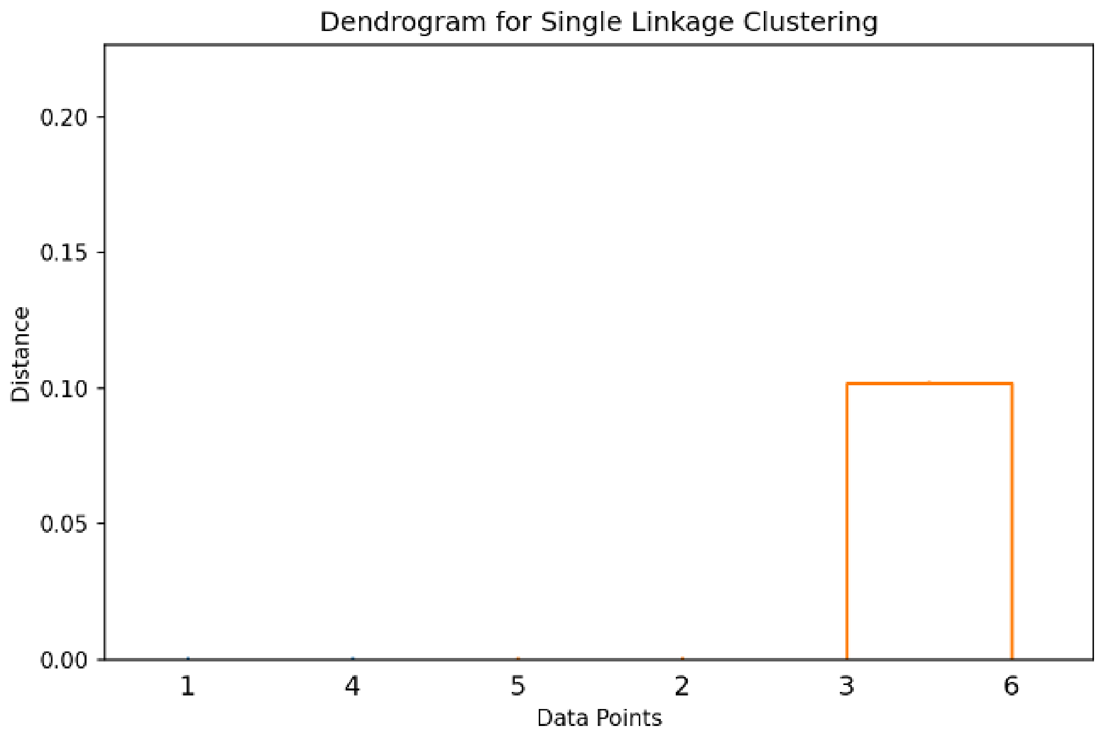
Point	x	y
1	0.40	0.53
2	0.22	0.38
3	0.35	0.32
4	0.26	0.19
5	0.08	0.41
6	0.45	0.30

The initial distance matrix is as follows:

Distance	1	2	3	4	5	6
1	0	0.2343	0.2159	0.3677	0.3418	0.2354
2		0	0.1432	0.1942	0.1432	0.2435
3			0	0.1581	0.2846	0.1020
4				0	0.2842	0.2195
5					0	0.3860
6						0

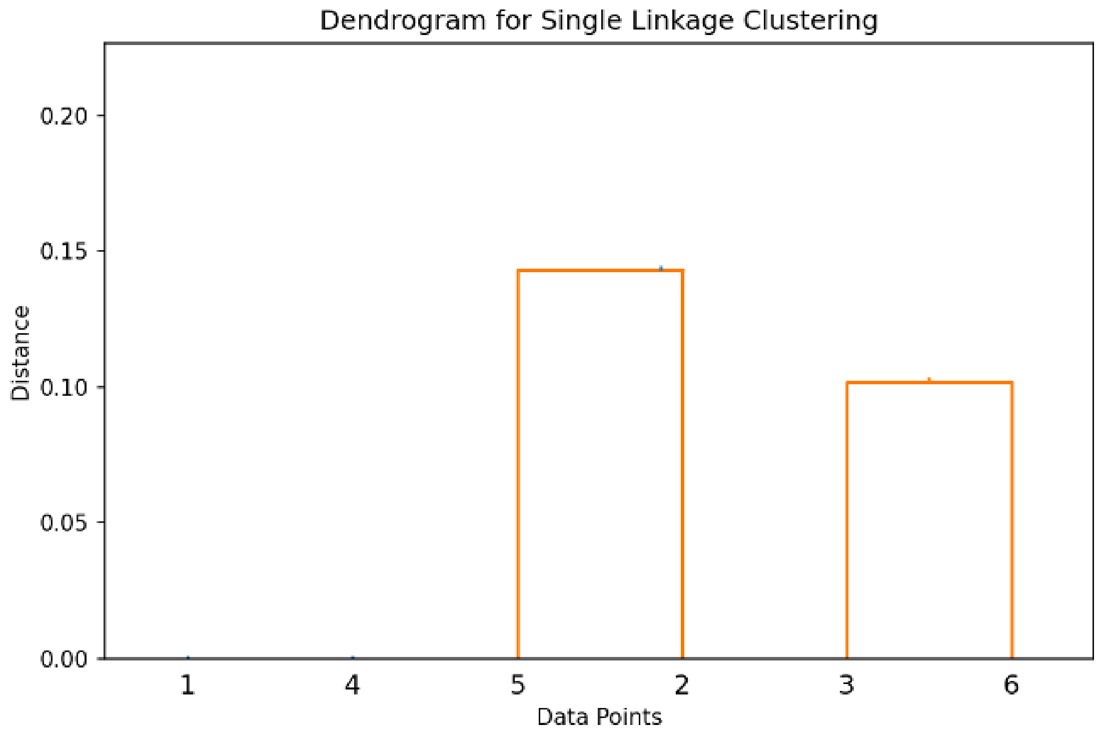
$$A_D = [0.2159\ 0.1432\ 0.1020\ 0.1581\ 0.1432]$$

$$A_V = [3\ 5\ 6\ 4\ 2]$$



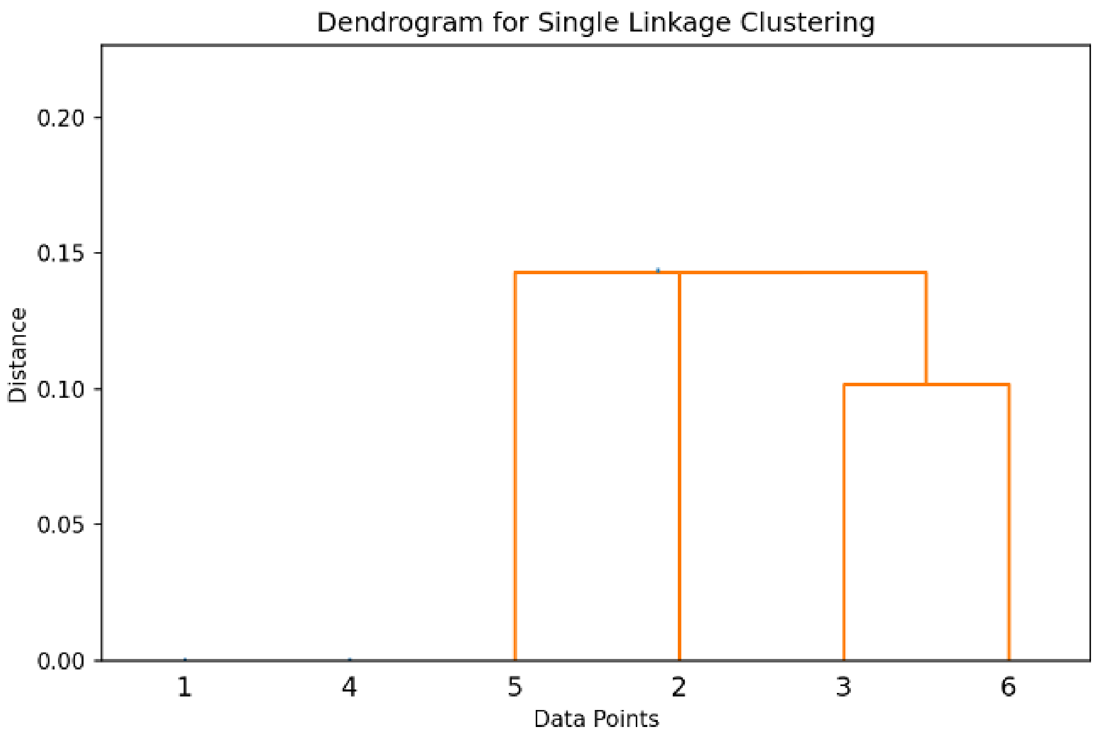
After clustering clusters 3 and 6, the updated distance and closest cluster arrays are:

$$A_D = [0.2159\ 0.1432\ 0.1432\ 0.1581\ 0.1432]$$
$$A_V = [3\ 5\ 3\ 4\ 2]$$



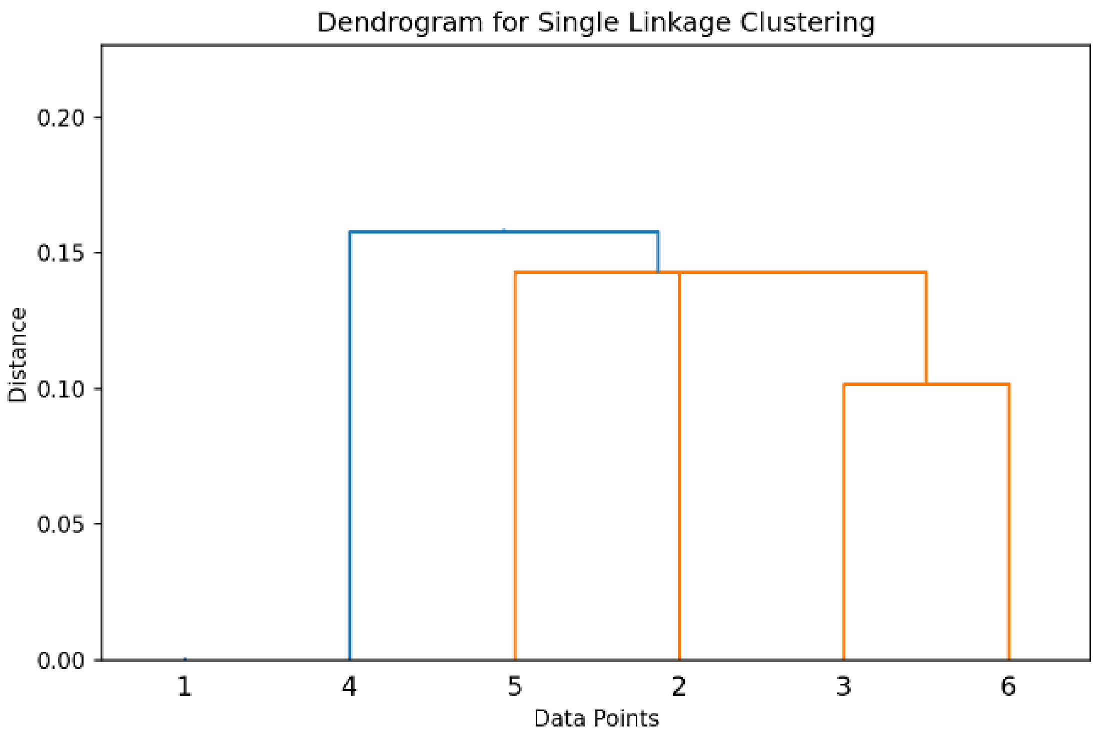
After clustering clusters 2 and 5, the updated distance and closest cluster arrays are:

$$A_D = [0.2159\ 0.1432\ 0.1432\ 0.1581\ 0.1432]$$
$$A_V = [3\ 2\ 3\ 4\ 2]$$



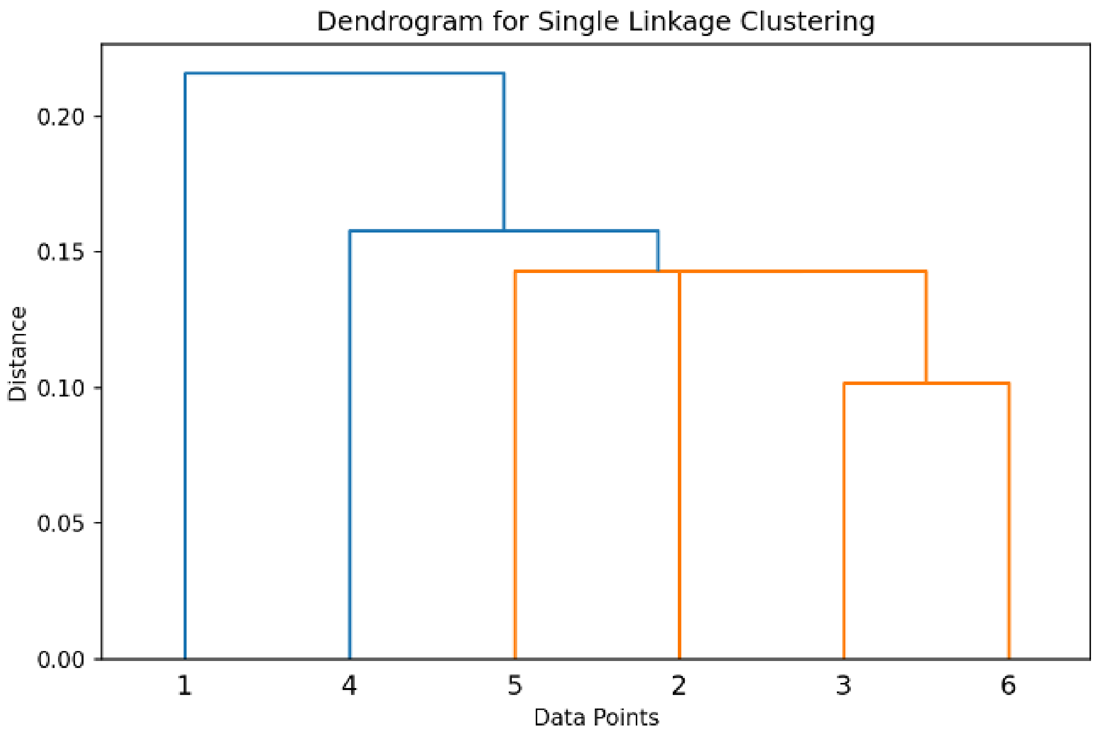
After clustering clusters 2, 3, 5 and 6, the updated distance and closest cluster arrays are:

$$A_D = [0.2159\ 0.1581\ \infty\ 0.1581\ \infty]$$
$$A_V = [2\ 4\ 2\ 4\ 2]$$



After clustering clusters 2, 3, 4, 5 and 6, the updated distance and closest cluster arrays are:

$$A_D = [0.2159 \; 0.2159 \; \infty \; \infty \; \infty]$$
$$A_V = [2 \; 1 \; 2 \; 2 \; 2]$$



After clustering clusters 1, 2, 3, 4, 5 and 6, the updated distance and closest cluster arrays are:

$$A_D = [0\ \infty\ \infty\ \infty\ \infty]$$
$$A_V = [1\ 1\ 1\ 1\ 1]$$