COL761 - Data Mining

Assignment 2

Members

Rishi Jain - 2020CS10373
Chetan Gaikwad - 2020CS10335
Tanish Singh Tak - 2020EE10560

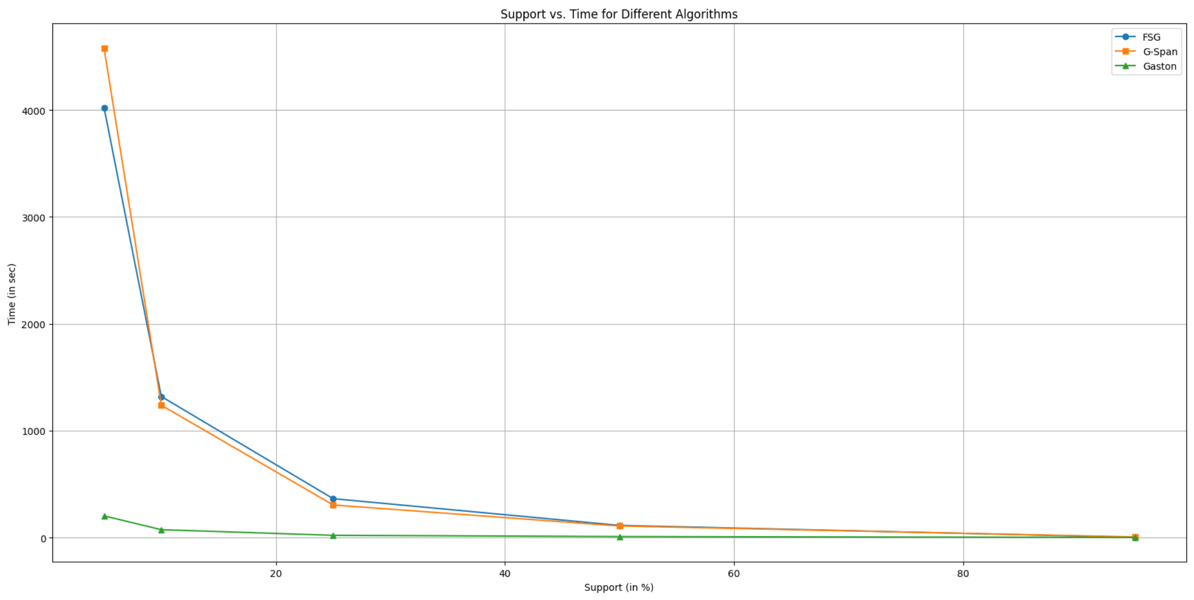
Date of Submission - 23 September 2023

1 Question 1

Plots for FSG, G-Span and Gaston at minSup = 5%, 10%, 25%, 50% and 95%

Table 1: Performance Comparison of FSG, G-Span, and Gaston

Algorithm	Observations		
	Support (in %)	User Time (in sec)	Number of Frequent Patterns
FSG	5	4021.8	46404
	10	1319.8	8549
	25	363.8	1062
	50	113.7	208
	95	2.2	3
G-Span	5	4579.09	46430
	10	1237.8	8555
	25	304.933	1066
	50	107.926	212
	95	5.77	5
Gaston	5	202.93	46424
	10	73.45	8549
	25	20.85	1062
	50	9.16	208
	95	1.13	3



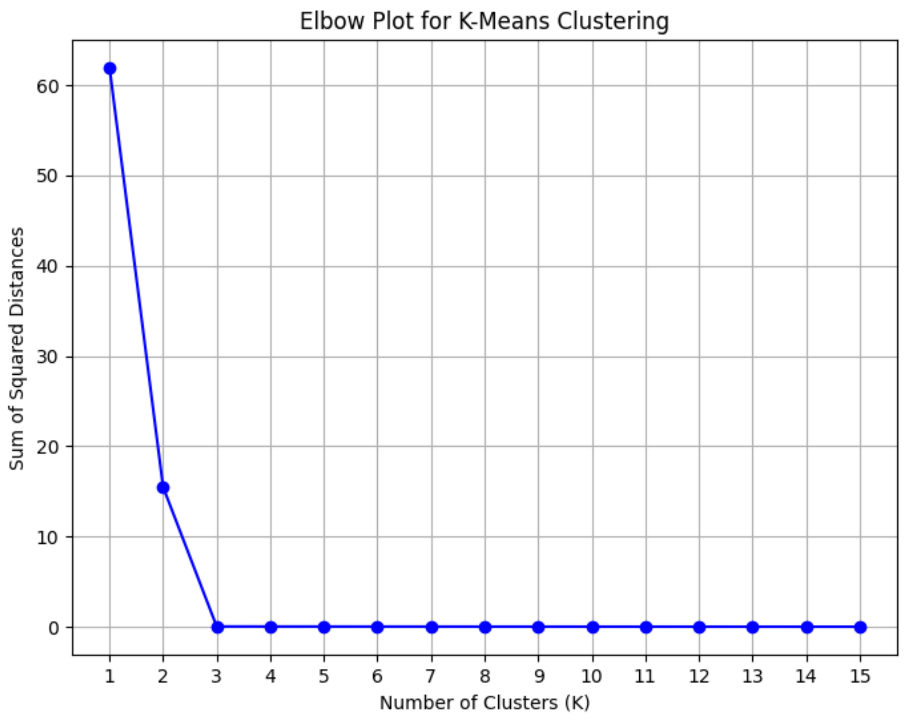
Trends and explaination:
We can observe that time taken for all algorithm decreases with increasing support. The key reasons for this observation are:
Reduced Search Space: As the support threshold increases, the number of potentially frequent subgraphs decreases. The search space for patterns to consider becomes smaller because only patterns that meet the higher support threshold are examined. This reduction in the search space leads to faster mining.
Fewer Candidates: Higher support leads to fewer potential candidates for frequent subgraphs. The algorithms spend less time generating and evaluating candidate subgraphs since many subgraphs will not meet the increased support requirement.
Less Pruning Overhead: Frequent subgraph mining algorithms often employ pruning techniques to eliminate patterns that cannot meet the support threshold. With a higher support threshold, more patterns are pruned early in the process, reducing the computational overhead associated with pruning.

Also it is observed that time taken by FSG and GSPAN is almost same but time taken by Gaston is quite low than others.
Why is Gaston much faster than other?
Efficient Enumeration of Paths and Trees: Gaston efficiently enumerates paths and trees as fragments first before considering general graphs with cycles. This strategy is effective for low support because paths and trees are simpler and more frequent structures compared to general graphs with cycles. By tackling the more frequent cases first, Gaston reduces the overall computational load.
Global Order on Cycle-Closing Edges: Gaston defines a global order on cycle-closing edges and generates cycles that are "larger" than the last one. This order helps prioritize the generation of larger cycles, potentially reducing the number of unnecessary smaller cycles to consider. This is especially advantageous for graphs with low support, where cycles might be relatively rare.
NP-Completeness Consideration: Gaston defers the handling of graphs with cycles (known to pose NP-completeness challenges) to the end of the mining process. This ensures that the more computationally intensive tasks are deferred until later, and the algorithm primarily focuses on simpler fragments initially.

These strategies collectively allow Gaston to efficiently handle graphs with low support by prioritizing the mining of simpler and more frequent structures first, and deferring the handling of potentially complex cycles until later in the process.

2 Question 2

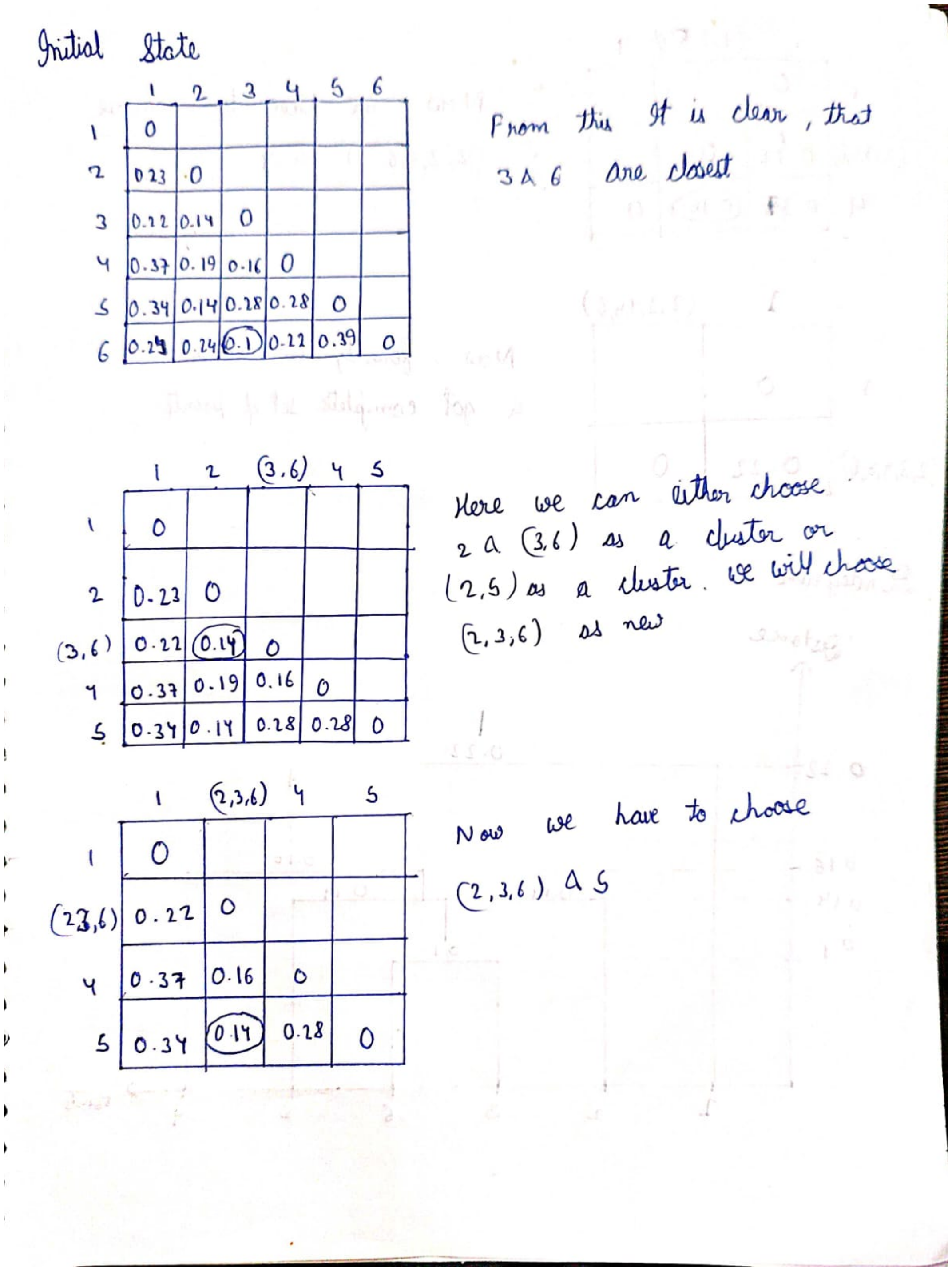
For this part we have chosen dimension as 5. The elbow plot is as follows:



From this plot we can see that the elbow is at 3. Thus, we choose 3 as the number of clusters.

3 Question 3

3.1 Draw the dendrogram for single linkage clustering on the data below. Show all the steps.



Initial State

	1	2	3	4	5	6
1	0					
2	0 23	0				
3	0.22	0.14	0			
4	0.37	0.19	0.16	0		
5	0.34	0.14	0.28	0.28	0	
6	0.29	0.24	0.1	0.22	0.39	0

From this It is clear, that 3 & 6 are closest

	1	2	(3,6)	4	5
1	0				
2	0.23	0			
(3,6)	0.22	0.14	0		
4	0.37	0.19	0.16	0	
5	0.34	0.14	0.28	0.28	0

Here we can either choose 2 & (3,6) as a cluster or (2,5) as a cluster. We will choose (2,3,6) as new

	1	(2,3,6)	4	5
1	0			
(2,3,6)	0.22	0		
4	0.37	0.16	0	
5	0.34	0.14	0.28	0

Now we have to choose (2,3,6) & 5

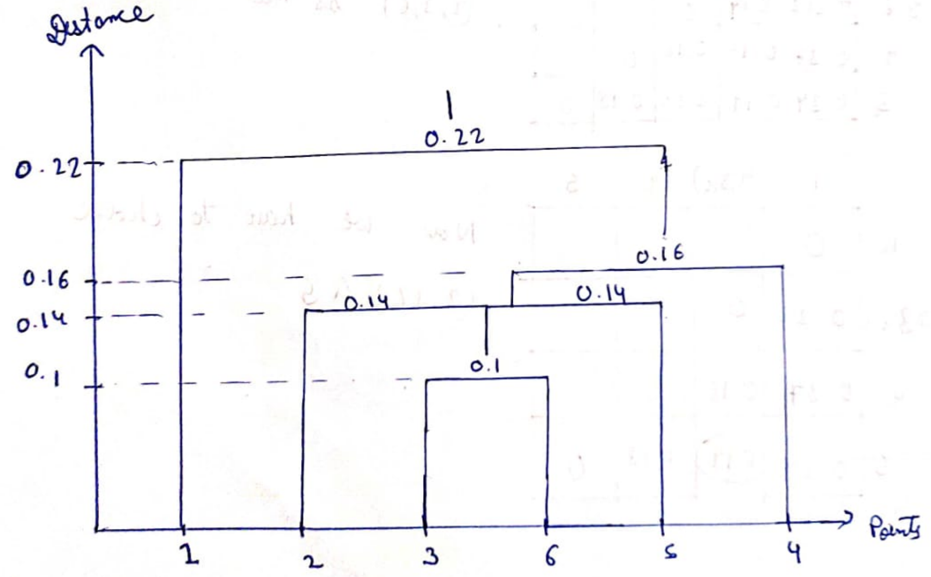
	1	(2,3,5,6)	4
1	0		
(2,3,5,6)	0.22	0	
4	0.37	0.16	0

Now we have to choose (2, 3, 5, 6) & 4

	1	(2,3,4,5,6)
1	0	
(2,3,4,5,6)	0.22	0

Now, finally we choose 1 & (2,3,4,5,6), & got complete set of points

Dendogram



3.2 What is the complexity of the fastest possible algorithm? Give your algorithm's pseudocode and complexity analysis.

The idea of our algorithm is that since we are adding an edge between the two closest clusters in each iteration, we will get a minimum spanning tree. Now instead of using Kruskal's algorithm to find the minimum spanning tree(MST), we will use Prim's algorithm and then add the edges in sorted order to get the dendrogram.
The reason for this is that Kruskal's algorithm has a complexity of $O(ElogV)$, where E is the number of edges. Since we are using single linkage clustering, the number of edges is $O(n^2)$, where n is the number of points. Thus, the complexity of Kruskal's algorithm is $O(n^2logn)$. If we use Prim's algorithm we can reduce the complexity to $O(n^2)$, which is the complexity of Prim's algorithm if we do not use a heap and do linear search to find the minimum edge.
The time complexity of any clustering algorithm based on pairwise distances between N points cannot be less than $O(N^2)$. This is because, in order to form clusters, we need to calculate the distance between each pair of points. With N points, there are $\frac{N\cdot(N-1)}{2}$ unique pairs, resulting in at least $O(N^2)$ distance calculations.
Thus, the theoretical lower bound for the time complexity of any clustering algorithm that relies on pairwise distances is $O(N^2)$.
This proves that our algorithm is the fastest possible algorithm.
Pseudocode:
Note:- d(p,q) is the distance between points p and q.

```
Applying Prim's algorithm:
$n \leftarrow$ number of points
$list\_points \leftarrow$ list of points
$visited \leftarrow$ array of size $n$ initialized to $False$
$distance\_with\_neighbour \leftarrow$ array of pairs of size $n$ initialized to $(\infty, None)$    $\triangleright$ stores the nearest neighbour of each point from the current set and the distance between them
$MST \leftarrow$ empty list
$current \leftarrow$ random point from $list\_points$
$visited[current] \leftarrow True$
$distance[current] \leftarrow 0$
$completed \leftarrow 1$
for $p$ in $list\_points$ do
    if $visited[p] == False$ and $distance[p] > d(current, p)$ then
        $distance[p] \leftarrow d(current, p)$
        $distance\_with\_neighbour[p] \leftarrow (d(current, p), current)$
    end if
end for
while $completed < n$ do
    $\min\_dist \leftarrow \infty$
    $next \leftarrow$ None
    for $p$ in $list\_points$ do
        if $visited[p] == False$ then
            if $distance[p] < \min\_dist$ then
                $\min\_dist \leftarrow distance[p]$
                $next \leftarrow p$
            end if
        end if
    end for
    $MST.append(((next, distance\_with\_neighbour[next][0]), distance\_with\_neighbour[p][1]))$
    $visited[next] \leftarrow True$
    $completed \leftarrow completed + 1$
    for $p$ in $list\_points$ do
        if $visited[p] == False$ and $distance[p] > d(current, p)$ then
            $distance[p] \leftarrow d(current, p)$
            $distance\_with\_neighbour[p] \leftarrow (d(current, p), current)$
        end if
    end for
end while
Create dendrogram starting by combining two points with minimum distance
```

```
$MST.sort(with\_key = lambdax : x[1])$
$levels\_cluster \leftarrow$ empty list of size $n$
for  do$p$ in $list\_points$:
    $levels\_cluster[0].append([p])$
end for
$level \leftarrow 1$
for $edge$ in $MST$ do
    $cluster1 \leftarrow$ cluster of $edge[0][0]$ in $levels\_cluster[level - 1]$
    $cluster2 \leftarrow$ cluster of $edge[0][1]$ in $levels\_cluster[level - 1]$
    $new\_cluster \leftarrow$ cluster1 $\cup$ cluster2
    $levels\_cluster[level].append(new\_cluster)$
    for $cluster$ in $levels\_cluster[level - 1]$ do
        if $cluster \neq cluster1$ and $cluster \neq cluster2$ then
            $levels\_cluster[level].append(cluster)$
        end if
    end for
end for
```

Time Complexity Analysis:
Prim's Algorithm to Construct MST
The time complexity of Prim's algorithm is analyzed as follows:

- **Initialization Step:** Initializing arrays takes $O(n)$ time.
- **Main Loop:** The main loop runs n times. Within each iteration, operations on each point in `list_points` are performed. For each point, we may update distances and choose the minimum distance, which takes $O(n)$ time.

Therefore, the total time complexity for Prim's algorithm is $O(n^2)$.
Creating Dendrogram from MST
The time complexity of creating the dendrogram from MST is analyzed as follows:

- **Sorting MST Edges:** Sorting the MST edges takes $O(n \log n)$ time.
- **Creating Levels of Clusters:** Iterating through the sorted MST edges and creating levels of clusters takes $O(n^2)$ time.

Therefore, the overall time complexity is $O(n^2)$ due to this step.