OTDM: The cluster-median problem

Gabriel Zarate, Ximena Moure

January 2024

Contents

1 Introduction

In this report, we're tackling the cluster-median problem, a fundamental part of grouping data. The goal is to arrange data points into groups so that they're as close as possible to a central point within their group, called the median. This helps make sure that points in the same group are similar to each other but different from those in other groups. We use two main strategies: a precise mathematical method for finding the best solution and a quicker, smart approach using something called a minimum spanning tree. By comparing these strategies, we highlight the complexities and practical considerations of organizing data into groups effectively.

In this study, we explore the cluster-median problem by using different data sets, each with m data points and n attributes. We organize these points into k groups to ensure that points within the same group are alike. We also discuss the specific median used for a subset of points I, which is the point closest to all other points in I. This median is the central point in I that has the smallest sum of distances to all other points in I, using the matrix of Euclidean distances, D, for our calculations.

Mathematically, we can state it as follows: The used median for a subset of points $I \subseteq \{1, \ldots, m\}$, defined as the nearest point to all points of I:

$$r \text{ is the median of } I \text{ if } \sum_{i \in I} d_{ir} = \min_{j \in I} \sum_{i \in I} d_{ij}$$

Where $D = (d_{ij})$, $i = 1, \ldots, m$, $j = 1, \ldots, m$ is the matrix of Euclidean distances.

2 Datasets

For the dataset generation, which involves the use of randomized generation and sampling a seed was defined to ensure the reproducibility of the project.

All the code for the datasets generation is in the file 'dataGen.py'. This file generates the .txt files used for the heuristic solution and the .dat files used in AMPL for the formulation as an integer optimization problem.

2.1 Iris dataset

The Iris dataset contains 150 observations of iris flowers from three different species: Iris setosa, Iris virginica, and Iris versicolor. It is composed by four numeric features which are:

- Sepal Length: The length of the outer part of the flower (sepal).
- Sepal Width: The width of the outer part of the flower.
- Petal Length: The length of the inner part of the flower (petal).
- Petal Width: The width of the inner part of the flower.

For this particular project it was needed to work with a small number of points, so it was decided to sample 25 observations to work with. So finally for this case, the matrix A will be of dimensions 25x4. Figure 1 shows the distribution of this dataset by using the first two features to this dataset being able to be plotted.

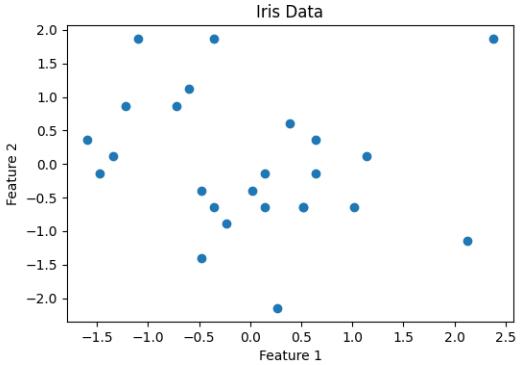

Figure 1: Iris Dataset Visualization

2.2 Moon dataset

The moon dataset is a synthetic dataset provided by the $sklearn.datasets.make_moons$ function in the Scikit-learn library. The dataset consists of two interleaving half circles, which resemble the shape of two moons, and is 2D (which means has two features). This creates a non-linearly separable pattern, which will be an interesting approach to evaluate on this project. For this specific case, a 25 observations dataset was generated with a small noise of 0.05 added. So finally for this case, the matrix A will be of dimensions 25x2. Figure 2 shows the distribution of this dataset.

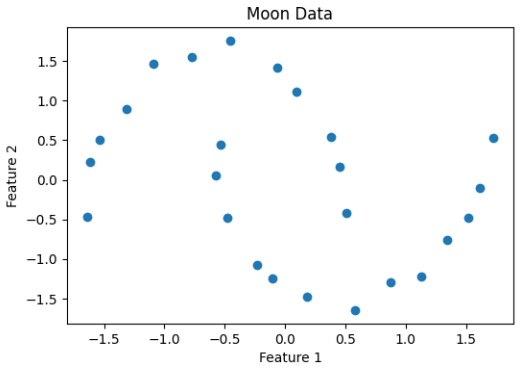

Figure 2: Moon Dataset Visualization

2.3 Blob dataset

To create the dataset, we used the make_blobs function from the Scikit-Learn library. The following parameters were specified:

- n_samples: 500.
  This parameter represents the number of data points in the dataset. In this case, we generated 500 data points.

- n_features: 2.
  The n_features parameter determines the number of features (or dimensions) for each data point. Our dataset has two features.

- centers: 3.
  The centers parameter defines the number of centers or clusters in the dataset. We generated data with three distinct clusters.

- cluster_std: 2.
  Additionally, we set the cluster_std parameter to 2, which controls the standard deviation of each cluster. This value affects the spread or dispersion of data points within each cluster.

- random_state: 42
  To ensure reproducibility, we used a specific random seed (random_state) with the value 42. This ensures that the dataset remains consistent across different runs.

The resulting dataset consists of 500 data points, each with two features, and it is organized into three distinct clusters. The clusters have varying degrees of dispersion, as controlled by the cluster_std parameter.

Matrix A dimension for this case is 500x2.

Figure 3 shows the distribution of this dataset.

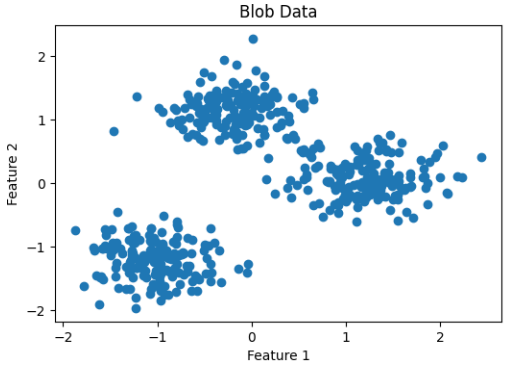

Figure 3: Blob Dataset Cluster Visualization

3 Formulation as an integer optimization problem

The integer optimization approach simplifies the cluster-median problem by assuming a maximum of m clusters, where $m - k$ are essentially placeholders and will not be populated. The median of a cluster is represented by its corresponding element index. We assign binary variables to represent cluster membership. The objective is to assign points to clusters in such a way that the sum of the distances from points to their cluster medians is minimized, thus forming k distinct and valid clusters. If a median for a cluster is designated, every point assigned to that cluster will contribute to the minimization of the overall distance. The formulation, which aims to optimize this assignment, is expressed as follows:

$$
\begin{aligned}
&\min \sum_{i=1}^{m} \sum_{j=1}^{m} d_{ij} x_{ij} \\
&\text{subject to} \\
&\sum_{j=1}^{m} x_{ij} = 1, \quad i = 1, \ldots, m \\
&\sum_{j=1}^{m} x_{jj} = k \\
&x_{ij} \geq x_{jj}, \quad i, j = 1, \ldots, m \\
&x_{ij} \in \{0, 1\}
\end{aligned}
$$

The last group of constraints can be formulated as follows:

$$
\begin{aligned}
&\min \sum_{i=1}^{m} \sum_{j=1}^{m} d_{ij} x_{ij} \\
&\text{subject to} \\
&\sum_{j=1}^{m} x_{ij} = 1, \quad i = 1, \ldots, m \\
&\sum_{j=1}^{m} x_{jj} = k \\
&m x_{jj} \geq \sum_{i=1}^{m} x_{ij}, \quad j = 1, \ldots, m \\
&x_{ij} \in \{0, 1\}
\end{aligned}
$$

3.1 Implementation

In order to solve the integer problem AMPL was used. The code for the .mod file can be seen in 1.

We chose to specify the parameter 'k' within the '.run' file for each dataset to avoid the need for creating distinct '.mod' files for individual datasets, where the sole distinction among them would be the 'k' parameter.

The different .run files can be found in the folder 'ampl_code'.

```
# Number of data points
param m;
# Number of features per data point
param n;

# Parameter to store the data points
param A {1..m, 1..n};

# Distance matrix
param D {i in 1..m, j in 1..m} :=
  sqrt(sum {l in 1..n} (A[i,l] - A[j,l])^2);

# Binary variables indicating if point i belongs to cluster j
var x {1..m, 1..m} binary;
```

```
15
16 # Objective: Minimize the sum of distances from points to their cluster
       medians
17 minimize TotalDistance: sum{i in 1..m}(sum{j in 1..m} D[i,j]*x[i,j]);
18
19 # Each data point must belong to one and only one cluster
20 subject to AssignmentConstraint {i in 1..m}:
21   sum {j in 1..m} x[i,j] = 1;
22
23 # Exactly k clusters must be chosen
24 subject to ClusterConstraint:
25   sum {j in 1..m} x[j,j] = k;
26
27 # A point can belong to a cluster only if the cluster exists
28 subject to ExistenceConstraint {i in 1..m, j in 1..m}:
29   x[j,j] >= x[i,j];
```

Listing 1: Code for cluster.mod

We also tested the implementation formulated with less constraints. See 2.

```
1  # Number of data points
2  param m;
3  # Number of features per data point
4  param n;
5
6  # Parameter to store the data points
7  param A {1..m, 1..n};
8
9  # Distance matrix
10 param D {i in 1..m, j in 1..m} :=
11   sqrt(sum {l in 1..n} (A[i,l] - A[j,l])^2);
12
13 # Binary variables indicating if point i belongs to cluster j
14 var x {1..m, 1..m} binary;
15
16 # Objective: Minimize the sum of distances from points to their cluster
       medians
17 #minimize TotalDistance: sum {i in 1..m, j in 1..m} (D[i,j] * x[i,j]);
18 minimize TotalDistance: sum{i in 1..m}(sum{j in 1..m} D[i,j]*x[i,j]);
19
20 # Each data point must belong to one and only one cluster
21 subject to AssignmentConstraint {i in 1..m}:
22   sum {j in 1..m} x[i,j] = 1;
23
24 # Exactly k clusters must be chosen
25 subject to ClusterConstraint:
26   sum {j in 1..m} x[j,j] = k;
27
28 #Formulation with less constraints
29 subject to c3 {j in 1..m}:
30         m*x[j,j] >= sum{i in 1..m} x[i,j];
```

Listing 2: Code for cluster.mod

3.2 Results

3.2.1 Iris dataset

The execution time was 0.074757 seconds and the objective function was 21.31205938. Figure 4 shows a section of the result of the AMPL code.

```
ampl: include ampl_code/clusterIris.run
SOLVING IRIS DATA
CPLEX 22.1.1.0: optimal integer solution; objective 21.31205938
104 MIP simplex iterations
0 branch-and-bound nodes
_total_solve_elapsed_time = 0.074757

### Assignment of points to clusters ###

x [*,*]
:    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19 :=
1    0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
2    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
3    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
4    0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
5    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
6    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
7    0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
8    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
9    0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
10   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
11   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
12   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
13   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
14   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
15   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
17   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
18   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
19   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
20   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
21   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
22   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
23   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
24   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0
25   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0

:   20  21  22  23  24  25    :=
1    0   0   0   0   0   0
2    0   0   0   1   0   0
3    0   0   0   0   0   0
4    0   0   0   0   0   0
5    0   0   0   0   0   0
6    0   0   0   1   0   0
7    0   0   0   0   0   0
8    0   0   0   0   0   0
```

Figure 4: Part of AMPL result for the iris dataset

Figure 5 depicts the visualization of the clusters obtained.

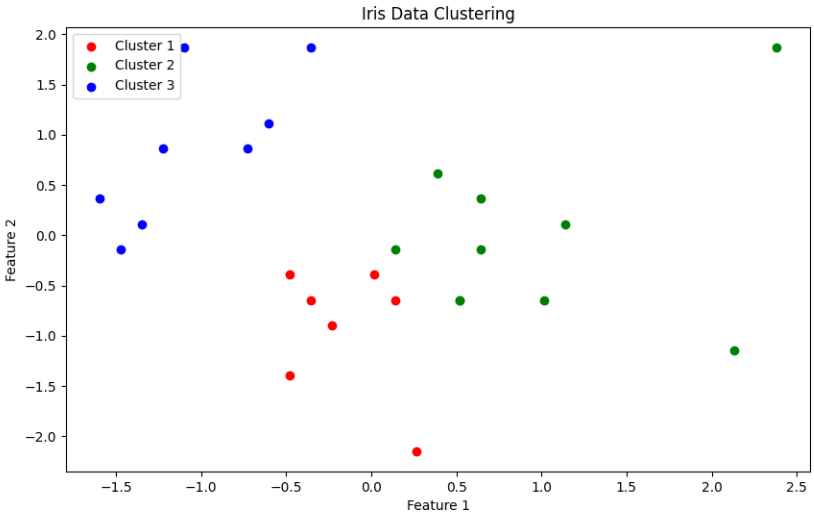

Figure 5: Iris result cluster visualization

3.2.2 Moon dataset

The execution time was 0.083285 seconds and the objective function was 22.99360788. Figure 6 shows a section of the result of the AMPL code.

```
ampl: include ampl_code/clusterMoon.run
SOLVING MOON DATA
CPLEX 22.1.1.0: optimal integer solution; objective 22.99360788
181 MIP simplex iterations
0 branch-and-bound nodes
_total_solve_elapsed_time = 0.083285

x [*,*]
:    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19 :=
1    0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
2    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
3    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
4    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
5    0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
6    0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
7    0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
8    0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
9    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
10   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
11   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
12   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
13   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
14   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
15   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
17   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
18   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
19   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
20   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
21   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
22   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
23   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0
24   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
25   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0

:   20  21  22  23  24  25    :=
1    0   0   0   0   0   0
2    0   0   0   0   1   0
3    0   0   0   0   1   0
4    0   0   0   0   1   0
5    0   0   0   0   0   0
```

Figure 6: Part of AMPL result for the moon dataset

Figure 7 depicts the visualization of the clusters obtained.

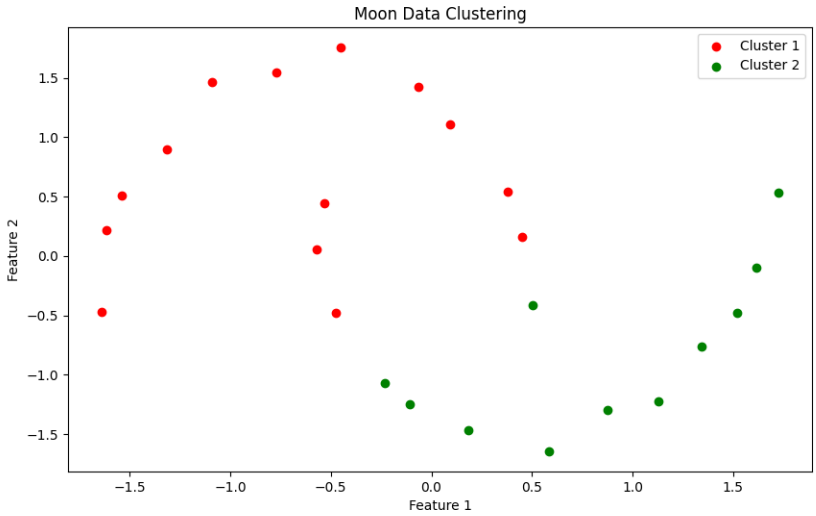

Figure 7: Moon result cluster visualization

3.2.3 Blobs dataset

The execution time was 38.3651 seconds and the objective function was 211.4307289. Figure 9 shows a section of the result of the AMPL code.

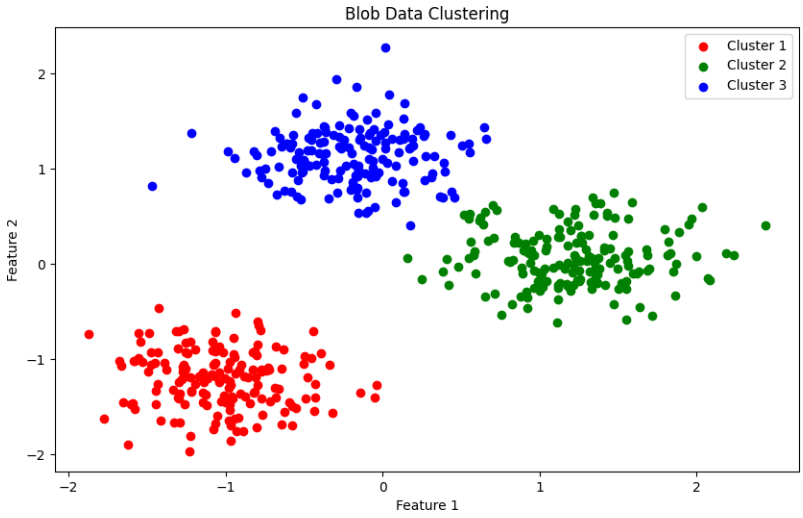

Figure 8: Moon result cluster visualization

```
ampl: include ampl_code/clusterBlob.run
SOLVING BLOBS DATA
CPLEX 22.1.1.0: optimal integer solution; objective 211.4307289
42621 MIP simplex iterations
0 branch-and-bound nodes
_total_solve_elapsed_time = 38.3651

### Assignment of points to clusters ###

x [*,*]
:     1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19 :=
1     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
3     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
4     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
5     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
6     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
7     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
8     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
9     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
10    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
11    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
12    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
13    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
14    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
15    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
16    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
17    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
18    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
19    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
20    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
21    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
22    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
23    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
24    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
25    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
26    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
27    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
28    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
29    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
30    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
31    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Figure 9: Part of AMPL result for the blobs dataset

Figure 8 depicts the visualization of the clusters obtained.

3.2.4 Formulation with less constraints

We conducted tests on both configurations (the one with less constraints and the other one) and obtained similar results; however, the second configuration required significantly more time for computation. We recognize that despite the second formulation having fewer constraints (using 'm' instead of 'm2'), this doesn't guarantee an improved formulation.

When running the code with the blobs dataset with less constraints the execution time was 1171.83 seconds , while with the other configuration the execution time was 38.3651 seconds.

4 Heuristic solution as a minimum spanning tree problem

4.1 Implementation

The implementation of the minimum spanning tree (MST) clustering was executed in Python, primarily utilizing the `networkx` library for its robust graph analysis capabilities. Notably, `networkx` includes a built-in `minimum_spanning_tree` function, which was crucial in the implementation. Supplementary libraries such as NumPy and pandas were also used mainly for data manipulation and computation.

The clustering function is designed to intake two parameters: the dataset as a NumPy array and a predetermined value of k, representing the desired number of clusters. The function's output is a list, with each element corresponding to the cluster assignment of the respective observation.

The function's logic unfolds in the following stages:

1. **Graph Construction**: A complete graph is generated from the dataset, where the edges' weights correspond to the Euclidean distances between data points. The Euclidean distances are efficiently computed using NumPy's `np.linalg.norm` function, which calculates the norm of the vector difference between two points.

2. **MST Computation**: The MST of the constructed graph is computed leveraging Kruskal's algorithm. This is facilitated by the `minimum_spanning_tree` function in `networkx`, where the algorithm parameter is explicitly set to "kruskal". This step effectively establishes the foundational structure for clustering.

3. **Edge Removal**: To form exactly k clusters, the $k-1$ edges with the greatest weights are excised from the MST. This process involves first sorting the edges in descending order of their weights and then systematically removing the top $k-1$ edges.

4. **Cluster Identification**: In the final stage, the resultant disconnected components of the MST are identified and delineated as distinct clusters.

The detailed code can be seen below. The full code can be found in the attached folder under the name 'mst.py'.

Listing 3: Code for clustering using MST

```
def find_clusters_with_mst_networkx(data, k):
    # Build a complete graph with nodes representing data points
    G = nx.complete_graph(len(data))

    # Add weights (distances) to the edges
    for i in range(len(data)):
        for j in range(i + 1, len(data)):
            # Euclidean distance between data points
            distance = np.linalg.norm(data[i] - data[j])
            G.add_edge(i, j, weight=distance)

    # Compute the minimum spanning tree of the graph
    mst = nx.minimum_spanning_tree(G, algorithm='kruskal')

    # Sort edges by weight in descending order and remove k-1 highest
        weighted edges
    edges = sorted(mst.edges(data=True), key=lambda x: x[2]['weight'],
        reverse=True)
    for i in range(k - 1):
        mst.remove_edge(edges[i][0], edges[i][1])

    # Identify clusters (connected components)
    clusters = list(nx.connected_components(mst))

    return [list(cluster) for cluster in clusters]
```

To be able to compute the objective function for the heuristic approach so that we can compare it with the AMPL code result we did the following (the code can be found in the file called `'mst.py'`):

- Identify the median (or medoid) of each cluster found by the MST heuristic.
- Compute the distance from each point in the cluster to this median.
- Sum these distances to obtain the total distance, similar to the objective function in the AMPL model.

4.2 Results

4.2.1 Iris

The Iris dataset clustering was set to use k=3, because it is known to have 3 real classes. The execution time was 0.003 seconds and the resultant clusters can be seen on Figure 10.

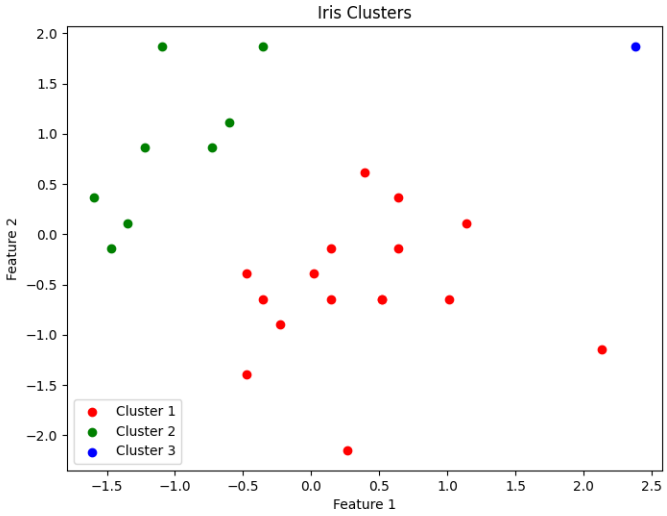

Figure 10: Iris Dataset Cluster Visualization

The objective function was 23.958149786835094.

4.2.2 Moon

The Moon dataset clustering was set to use k=2, because of its nature. The execution time was 0.002 seconds and the resultant clusters can be seen on Figure 11.

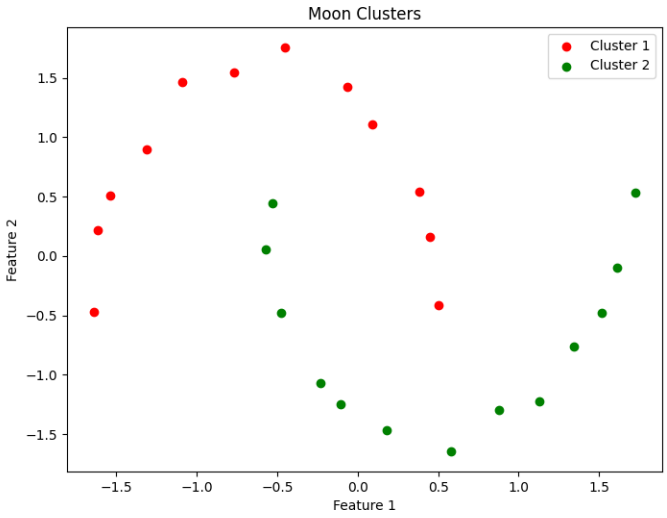

Figure 11: Moon Dataset Cluster Visualization

The objective function was 28.462678273585183.

4.2.3 Blobs

The execution time was 0.656424 seconds and the resultant clusters can be seen on Figure 12.

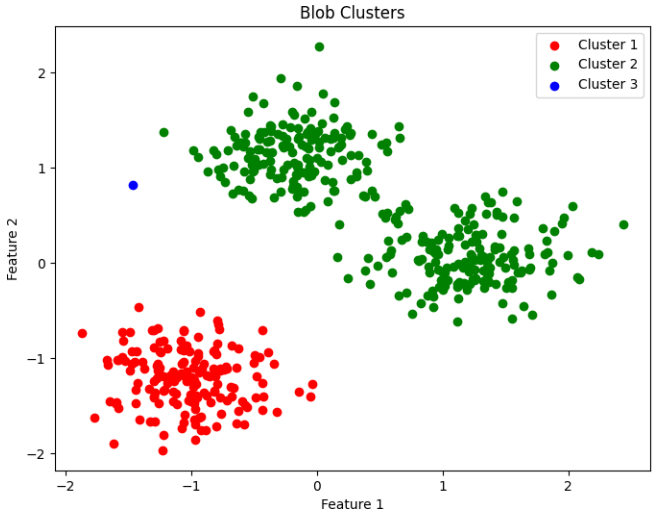

Figure 12: Blobs Dataset Cluster Visualization

The objective function was 382.9945462426345.

5 Comparison

As can be seen in the previous two sections (in the results of each method) we don‘t get the same cluster results from the formulation as an integer optimization problem and from the heuristic approach.

Some of the reasons can be the following ones:

- Objective function differences: The AMPL model minimizes the sum of the distances from points to their cluster medians. It's a direct optimization problem where we look for the absolute minimum. On the other hand, the heuristic method using a minimum spanning tree (MST) is an approximation that does not necessarily minimize the same objective function.
- Solution Quality: CPLEX aims for the optimal solution, whereas MST aims for a solution that is good enough and can be found quickly. The solution from CPLEX is likely to be closer to the true minimum sum of distances from points to their cluster medians, while the MST approach may yield a suboptimal clustering that does not minimize the objective function as well.
- Runtime and Complexity: CPLEX can handle complex models but may take more time to find an optimal solution for large datasets. MST is much faster but at the cost of potentially lower-quality solutions.
- Optimization Method: CPLEX uses branch-and-cut algorithms for integer programming which are very effective for finding globally optimal solutions within the search space defined by our model's constraints and objective function. It systematically explores feasible solutions and uses cutting planes to eliminate regions that do not contain an optimal solution. The minimum spanning tree (MST) approach used in Python with NetworkX is a heuristic that does not guarantee an optimal solution. It is a greedy algorithm that aims to connect all points with the minimum total distance but does not consider the distribution of points within clusters.

Table 1 shows the results from the different methods with the different datasets.

The data indicates that the AMPL solution consistently yields a superior objective function value. Furthermore, it becomes evident that the disparity between the optimal and heuristic solutions tends to widen as the dataset size grows.

Method	Dataset	Seconds	Objective Function
AMPL	Iris	≈ 0.07476	≈ 21.31205938
MST	Iris	≈ 0.003	≈ 23.9582
AMPL	Moon	≈ 0.08329	≈ 22.99361
MST	Moon	≈ 0.002	≈ 28.46268
AMPL	Blobs	≈ 38.3651	≈ 211.43073
MST	Blobs	≈ 0.6564	≈ 382.99455

Table 1: Comparison of AMPL and MST methods.

6 Conclusions

Throughout the course of the project, we conducted a comparative analysis between the integer optimization problem and the heuristic method using various datasets. As anticipated, the findings reveal that the optimal solution demands greater computational time compared to the heuristic approach; however, it consistently arrives at a more advantageous outcome.

7 Attachments

Folder 'ampl_code' conntaind the '.run' and 'mod' files used for solving the formulation as an integer optimization problem. Each dataset has it own '.run' file and then there is on '.mod' file for all. Additionally, there are is one file called 'cluster_lesss_constraints.mod' which is mostly the same as the other .mod file but it is the formulation with less constraints. Finally, there is the file file called 'cluster_blob_less_constraints.run' which runs the formulation with less constraints for the blob dataset.

Folder 'data' contains all the datasets used in the project. It has the .txt files used in the heuristic problem and the .dat files used in AMPL. All the files are generated using the file called 'dataGen.py'.

File 'mst.py' contains the implementation of the minimum spanning tree.

Folder 'images' contains all the images generated when running the 'mst.py' file.

Folder 'output' contains all the outputs generated when running the AMPL code. We use the output to be able to do some visualization of the clusters.

Folder 'visualization' contains the code to visualize the output of the AMPL code. It has one file per dataset.