OTDM Integer Optimization Lab Assignment: The cluster-median problem

Mariem Gandouz, Egon Ferri

20th of DECEMBER 2019

Contents

1 Introduction

In this report, we will discuss the general ideas surrounding the k-medians problem. We take a look at what k-medians attempts to solve and how it goes about doing so. Next, we will solve the k-medians problem with both an integer optimization approach (using AMPL) and a heuristic approach obtained with the minimum spanning tree procedure (Kruskal's algorithm, implement in AMPL). Finally, we'll close off our report with a comparison of the two solutions obtained, in terms of the objective function of the integer optimization problem and plot the results to procure a better visualization of the 2 procedures.

2 K-Medians problem

To make our discussion of K-Medians problem easier, we will first need to introduce a new notation for talking about Clustering or Cluster Analysis to procure a better knowledge base concerning this section. It is a powerful method in the area of Machine Learning known as Unsupervised Learning. It means the discovery of delimitable groups of data that have as many similarities as possible. The aim is to identify groups (classes/clusters) that belong together and to assign the data according to their belonging. A cluster or a class is the combination of similar objects with the greatest possible similarity. Cluster algorithms have the goal of automatically forming these related classes, which have as many similarities as possible. In other words, the attribution to a cluster is done in a way that different clusters have the greatest possible dissimilarity. For similarity or dissimilarity there are defined reasonable measures, e.g. distances. This is usually not trivial and must be done with great care, but for simplicity, the Euclidean distance can be considered in our problem.

K-medians is a widely famous approach to performing this clustering task. It attempts to minimize the sum of the distance between a center and the other points in one cluster. Given the number of cluster, the center of each cluster is randomly set to be the median of all points in that cluster. By looking for this minimal distance, it will finally reach a convergence, where medians don't change anymore.

The median for a subset of points $\mathcal{I} \subseteq \{1, \ldots, m\}$ is defined as the nearest point to all points of $\mathcal{I}$:

$$r \text{ is the median of } \mathcal{I} \text{ if } \sum_{i \in \mathcal{I}} d_{ir} = \min_{j \in \mathcal{I}} \sum_{i \in \mathcal{I}} d_{ij} \text{ (i.e. when a convergence is reached)}$$

where $D = (d_{ij}), i = 1, ..., m, j = 1, ..., m$ is the matrix of the distances for each pair of points. D is computed from a given data matrix A $= (a_{ij}\ ), i = 1, ..., m, j = 1, ..., n$ of m points and n variables; using, as mentioned above, the Euclidean distance.

3 Dataset

For the experiments we consider a simple and famous dataset; the Iris dataset. Since we wanted three quite separate clusters (to have a simple problem to check that everything works) and two features (to have the possibility to have an easy and immediate graphical representation), we filtered the data set in order to simplify the problem.

Figure 1: Full Iris dataset

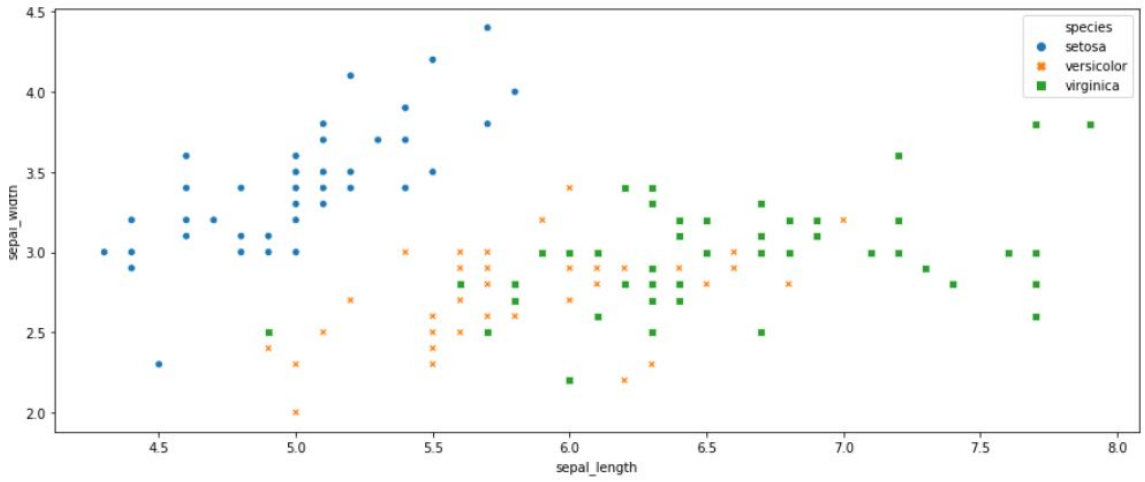


Figure 2: Filtered Iris dataset

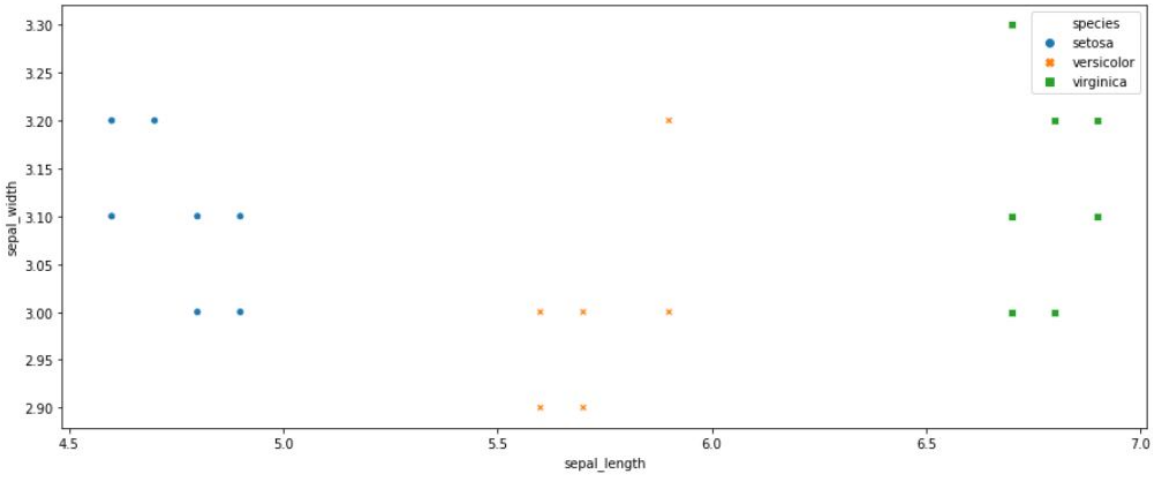


Figure 3: Points

	sepal_length	sepal_width	species
0	4.9	3.0	setosa
1	4.7	3.2	setosa
2	4.6	3.1	setosa
3	4.9	3.1	setosa
4	4.8	3.0	setosa
5	4.8	3.1	setosa
6	4.6	3.2	setosa
7	5.9	3.0	versicolor
8	5.6	2.9	versicolor
9	5.6	3.0	versicolor
10	5.9	3.2	versicolor
11	5.7	3.0	versicolor
12	5.7	2.9	versicolor
13	6.8	3.0	virginica
14	6.9	3.2	virginica
15	6.7	3.3	virginica
16	6.9	3.1	virginica
17	6.7	3.1	virginica
18	6.8	3.2	virginica
19	6.7	3.0	virginica

The related Euclidean distances matrix is:

Table 1: Euclidean distances to feed the optimizer

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0.00	0.28	0.32	0.10	0.10	0.14	0.36	1.00	0.71	0.70	1.02	0.80	0.81	1.90	2.01	1.82	2.00	1.80	1.91	1.80
0.28	0.00	0.14	0.22	0.22	0.14	0.10	1.22	0.95	0.92	1.20	1.02	1.04	2.11	2.20	2.00	2.20	2.00	2.10	2.01
0.32	0.14	0.00	0.30	0.22	0.20	0.10	1.30	1.02	1.00	1.30	1.10	1.12	2.20	2.30	2.11	2.30	2.10	2.20	2.10
0.10	0.22	0.30	0.00	0.14	0.10	0.32	1.00	0.73	0.71	1.00	0.81	0.82	1.90	2.00	1.81	2.00	1.80	1.90	1.80
0.10	0.22	0.22	0.14	0.00	0.10	0.28	1.10	0.81	0.80	1.12	0.90	0.91	2.00	2.11	1.92	2.10	1.90	2.01	1.90
0.14	0.14	0.20	0.10	0.10	0.00	0.22	1.10	0.82	0.81	1.10	0.91	0.92	2.00	2.10	1.91	2.10	1.90	2.00	1.90
0.36	0.10	0.10	0.32	0.28	0.22	0.00	1.32	1.04	1.02	1.30	1.12	1.14	2.21	2.30	2.10	2.30	2.10	2.20	2.11
1.00	1.22	1.30	1.00	1.10	1.10	1.32	0.00	0.32	0.30	0.20	0.20	0.22	0.90	1.02	0.85	1.00	0.81	0.92	0.80
0.71	0.95	1.02	0.73	0.81	0.82	1.04	0.32	0.00	0.10	0.42	0.14	0.10	1.20	1.33	1.17	1.32	1.12	1.24	1.10
0.70	0.92	1.00	0.71	0.80	0.81	1.02	0.30	0.10	0.00	0.36	0.10	0.14	1.20	1.32	1.14	1.30	1.10	1.22	1.10
1.02	1.20	1.30	1.00	1.12	1.10	1.30	0.20	0.42	0.36	0.00	0.28	0.36	0.92	1.00	0.81	1.00	0.81	0.90	0.82
0.80	1.02	1.10	0.81	0.90	0.91	1.12	0.20	0.14	0.10	0.28	0.00	0.10	1.10	1.22	1.04	1.20	1.00	1.12	1.00
0.81	1.04	1.12	0.82	0.91	0.92	1.14	0.22	0.10	0.14	0.36	0.10	0.00	1.10	1.24	1.08	1.22	1.02	1.14	1.00
1.90	2.11	2.20	1.90	2.00	2.00	2.21	0.90	1.20	1.20	0.92	1.10	1.10	0.00	0.22	0.32	0.14	0.14	0.20	0.10
2.01	2.20	2.30	2.00	2.11	2.10	2.30	1.02	1.33	1.32	1.00	1.22	1.24	0.22	0.00	0.22	0.10	0.22	0.10	0.28
1.82	2.00	2.11	1.81	1.92	1.91	2.10	0.85	1.17	1.14	0.81	1.04	1.08	0.32	0.22	0.00	0.28	0.20	0.14	0.30
2.00	2.20	2.30	2.00	2.10	2.10	2.30	1.00	1.32	1.30	1.00	1.20	1.22	0.14	0.10	0.28	0.00	0.20	0.14	0.22
1.80	2.00	2.10	1.80	1.90	1.90	2.10	0.81	1.12	1.10	0.81	1.00	1.02	0.14	0.22	0.20	0.20	0.00	0.14	0.10
1.91	2.10	2.20	1.90	2.01	2.00	2.20	0.92	1.24	1.22	0.90	1.12	1.14	0.20	0.10	0.14	0.14	0.14	0.00	0.22
1.80	2.01	2.10	1.80	1.90	1.90	2.11	0.80	1.10	1.10	0.82	1.00	1.00	0.10	0.28	0.30	0.22	0.10	0.22	0.00

4 Integer optimization problem

In this section, we will formalize k-median as an integer optimization problem. Let's keep going and suppose in our dataset, there are m examples, so m points if we can visualize them. Initially, we can simply think of these m points as m clusters, therefore there are m medians, the median actually is the point itself. Suppose there are only k clusters needed, then $(m-k)$ clusters are empty.
Now let's consider $i, j = 1, ..., m$; i, j represents two points. Since there are m clusters at the beginning, let j be the median of its own cluster, so we obtain the following form:

$$x_{ij} = \begin{cases} 1 & \text{if element } i \in \text{ cluster-}j \\ 0 & \text{otherwise.} \end{cases}$$

As point $j \in$ cluster-j (i.e. $\exists$ cluster-j), thus $x_{jj} = 1$ and $\sum_{j=1}^{m} x_{jj} = k$, as there should be exactly k clusters. Otherwise, the total distance would not be minimized.
These ideas are formalized in the following formulation of the cluster-median problem:

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

Another correct formulation is obtained if we replace the third set of m^2 constraints

$$x_{jj} \geq x_{ij} \quad i, j = 1, \ldots, m$$

with the set of m constraints

$$mx_{jj} \geq \sum_{i=1}^{m} x_{ij} \quad j = 1, \ldots, m$$

where m points $(i), m$ clusters $(j), d_{ij}$ is the distance between points i and j.

Let's move on. Looking ahead, as we develop k-median as an integer optimization problem, next idea to watch out for is that we'll try to implement it in AMPL to obtain the optimal solution.

4.1 In AMPL

Let's solve our problem in AMPL:

Figure 4: Model

```
# PARAMETERS

# Points
param m;
set M:={0..m-1};

# Euclidean distances matrix
param D{M,M};

# Clusters
param k;

# VARIABLE

# Belonging matrix-> to which cluster belong point M?
var x{M,M} binary;

# OBJECTIVE FUNCTION

minimize obj: sum{i in M, j in M} D[i,j] * x[i,j];

# CONSTRAINTS
# Every point belongs to one cluster

subject to c_1 {i in M}: sum{j in M} x[i,j]=1;

# Exactly k clusters
subject to c_2 :sum{j in M} x[j,j]=k;

# A point may belong to a cluster only if the cluster exists
subject to c_3 {i in M, j in M}: x[j,j]>=x[i,j];
```

Figure 5: Data

```
data;
param m := 20;
param D :0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19:=

0 0.0 0.28284271247462855 0.31622776601684016 0.10000000000002558 0.09999999999999006 0.14142135623732055 0.360555127
1 0.28284271247462855 0.0 0.14142135623732055 0.2236067977499885 0.2236067977499726 0.14142135623729543 0.100000000000
2 0.31622776601684016 0.14142135623732055 0.0 0.3000000000000057 0.2236067977499726 0.19999999999999787 0.099999999995
3 0.10000000000002558 0.2236067977499885 0.3000000000000057 0.0 0.14142135623732055 0.09999999999999006 0.316227766016
4 0.09999999999999006 0.2236067977499726 0.2236067977499726 0.14142135623732055 0.0 0.10000000000002558 0.28284271247
5 0.14142135623732055 0.14142135623729543 0.19999999999999787 0.09999999999999006 0.10000000000002558 0.0 0.223606797
6 0.36055512754640245 0.10000000000002558 0.09999999999999006 0.31622776601684016 0.282842712474616 0.2236067977499726
7 1.0 1.2165525060596427 1.3038404810405309 1.0049875621120845 1.1000000000000003 1.1045361017187223 1.315294643796585
8 0.7071067811865476 0.9486832980505131 1.019803902718553 0.7280109889280526 0.8062257748298541 0.8246211251235319 1.
9 0.7000000000000065 0.9219544457292895 1.0049875621120916 0.7071067811865476 0.8000000000000048 0.8062257748298541 1.
10 1.0198039027185566 1.199999999999999 1.3038404810405282 1.004987562112088 1.118033988749895 1.1045361017187256 1.29
11 0.8000000000000048 1.0198039027185601 1.1045361017187287 0.8062257748298541 0.9000000000000012 0.9055385138137378 1
12 0.8062257748298585 1.04403065089106 1.118033988749895 0.8246211251235362 0.9055385138137418 0.9219544457292895 1.14
13 1.9 2.1095023109728976 2.202271554554523 1.9026297590440422 1.9999999999999982 2.002498439450078 2.2090722034374526
14 2.0099751242241797 2.2000000000000024 2.302172886644267 2.00249843945008 2.109502310972901 2.102379604162866 2.3 1.
15 1.8248287590894654 2.002498439450078 2.1095023109728994 1.8110770276274837 1.923538406167137 1.9104973174542796 2.1
16 2.002498439450078 2.202271554554526 2.3 1.9999999999999982 2.102379604162864 2.099999999999999 2.302172886644267 1.
17 1.8027756377319946 2.002498439450078 2.099999999999999 1.7999999999999985 1.9026297590440442 1.899999999999998 2.16
18 1.9104973174542796 2.099999999999999 2.202271554554524 1.9026297590440442 2.009975124224176 2.0024984394500764 2.26
19 1.7999999999999985 2.009975124224178 2.102379604162864 1.8027756377319946 1.9000000000000017 1.9026297590440442 2.1
param k:= 3
;
```

4.2 Computational results

Figure 6: Results

```
ampl: option solver cplex;
ampl: model cluster_median.mod;
ampl: data euclid_1.dat;
ampl: solve;
CPLEX 12.9.0.0: optimal integer solution; objective 2.678584445
59 MIP simplex iterations
0 branch-and-bound nodes
ampl: display x;
x [*,*]
:    0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18 :=
0    0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0
1    0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0
2    0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0
3    0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0
4    0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0
5    0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0
6    0   0   0   0   0   1   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   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   1   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   1
14   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
15   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
17   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
18   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
19   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
```

As we can easily see from the obtained matrix, the points are clustered as expected (since it was a very easy problem) and the points chosen as median-clusters are 5, 11, 18.

The objective function value (i.e. the minimal total distance) is 2.678584445.

5 Minimum spanning tree problem

Previously, we saw that our combinatorial optimization problem is a hard one, and its solution is only viable when the m points of previous sections is not very large. Nevertheless, sub-optimal solutions are sometimes easy to find. Consequently, there is much interest in approximation and heuristic algorithms that can find good or near optimal solutions to our optimization problem within reasonable running time. These approximation and heuristic algorithms carry no guarantee that an optimal solution will be found.

The minimum cost spanning tree problem is to find a spanning tree of minimum cost. By looking for this, some algorithm (e.g. Kruskal's or Prim's algorithm) can find k clusters after removing $k-1$ largest edges, since the goal is to partition m points in k clusters of "similar points".

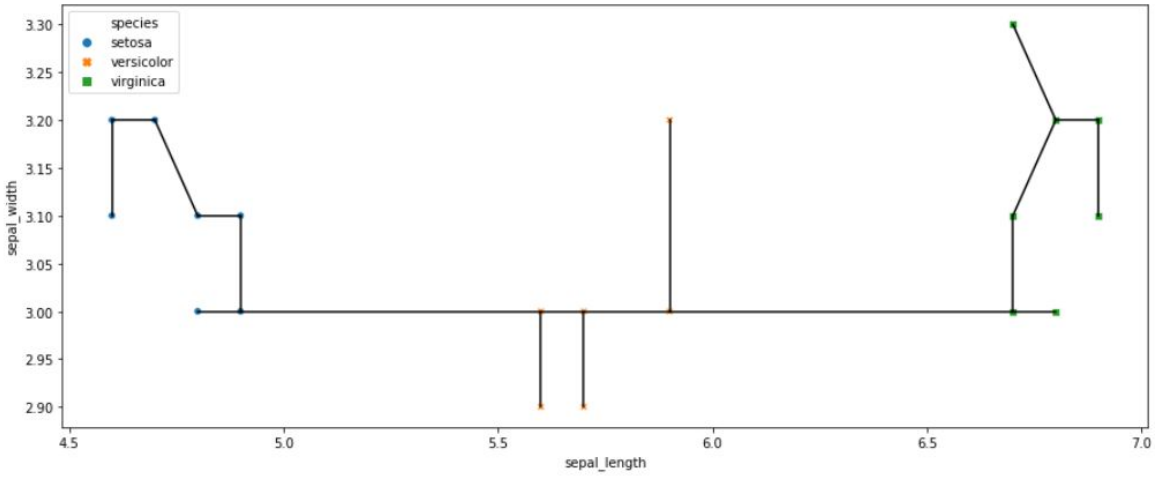
5.1 Implementation

To compute the minimum spanning tree of our graph, we choose to use the Kruskal's greedy algorithm via the scipy package of Python.

This is as easy as it goes:

```
X = csr_matrix(euclid)
Tcsr = minimum_spanning_tree(X)
minimum_spanning=Tcsr.toarray().astype(float)
```

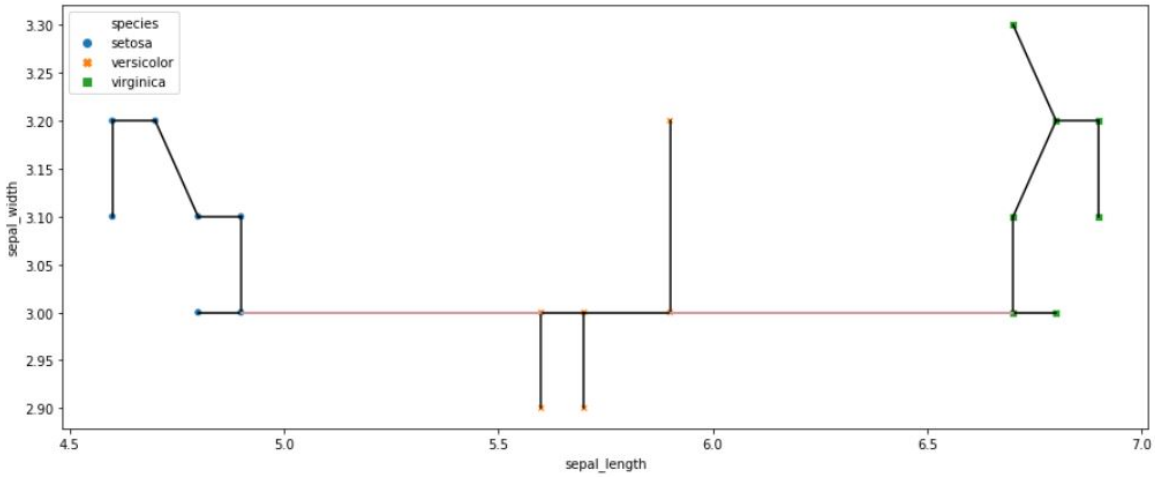
Figure 7: Minimum spanning tree



5.2 Computational results

To get our approximate solution we only need to cut out the most heavy edges, meaning that every edge in the spanning tree of minimum cost must have the least cost.

Figure 8: Approximate solution using the minimum spanning tree



5.3 Results comparison

As expected, since the problem is trivial, even this approximative method returns exactly the same solution.

The objective function will obviously be the same, but it's interesting to see how we can calculate it from this point.

We have to separate the connected components, find the clusters medians, and calculate the Euclidean distance from each point to his prototype. Then we sum up all distances, and we obtain the objective function. (We actually computed it, and it's the same).

6 Complicating things a bit

So, let's try transforming our problem into a more complicated one, to see if we can experiment some differences within the 2 methods.

6.1 New dataset

To build our new dataset, we begin again from the same Iris dataset, but we select points a little bit closer between them, and not clearly well separated in clusters.

Figure 9: New filtered Iris dataset

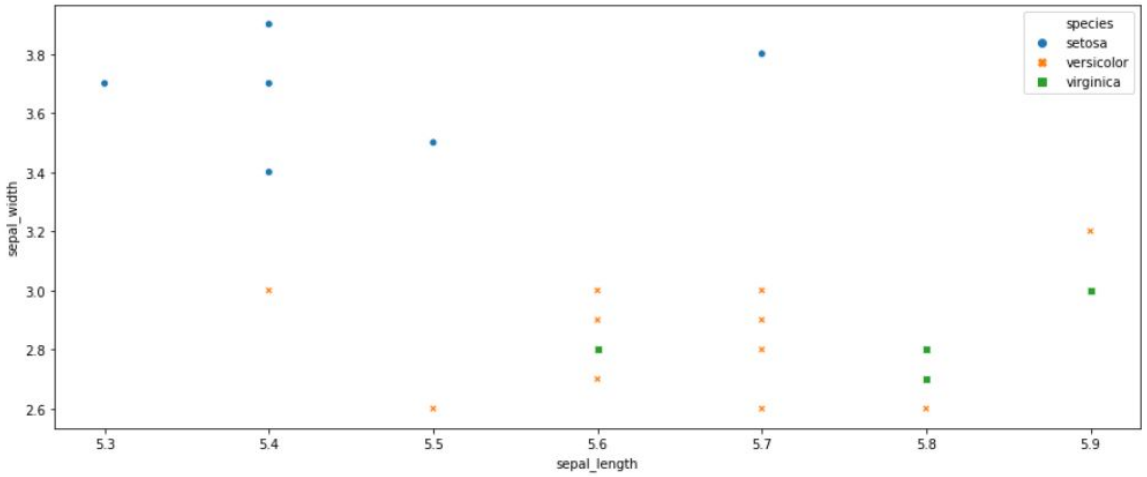


Figure 10: New points

	sepal_length	sepal_width	species
0	5.4	3.9	setosa
1	5.4	3.7	setosa
2	5.7	3.8	setosa
3	5.4	3.4	setosa
4	5.5	3.5	setosa
5	5.3	3.7	setosa
6	5.7	2.8	versicolor
7	5.9	3.0	versicolor
8	5.6	2.9	versicolor
9	5.6	3.0	versicolor
10	5.8	2.7	versicolor
11	5.9	3.2	versicolor
12	5.7	2.6	versicolor
13	5.4	3.0	versicolor
14	5.5	2.6	versicolor
15	5.8	2.6	versicolor
16	5.6	2.7	versicolor
17	5.7	3.0	versicolor
18	5.7	2.9	versicolor
19	5.8	2.7	virginica
20	5.8	2.8	virginica
21	5.6	2.8	virginica
22	5.9	3.0	virginica

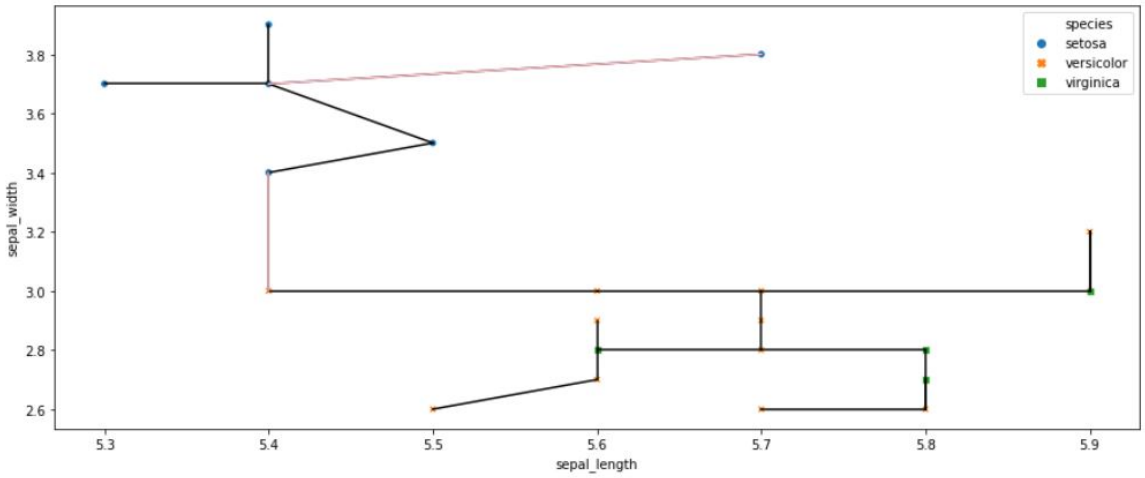
6.2 AMPL solution

Figure 11: Results

```
ampl: option solver cplex;
ampl: model cluster_median.mod;
ampl: data euclid_2.dat;
ampl: solve;
CPLEX 12.9.0.0: optimal integer solution; objective 3.686775909
116 MIP simplex iterations
0 branch-and-bound nodes
ampl: display x;
x [*,*]
:    0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18 :=
0    0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
1    0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2    0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
3    0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
4    0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
5    0   1   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   1   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   1   0
8    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
9    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
10   0   0   0   0   0   0   0   0   0   0   1   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   1   0
12   0   0   0   0   0   0   0   0   0   0   1   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   1   0
14   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0
15   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0
16   0   0   0   0   0   0   0   0   0   0   1   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   1   0
18   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0
19   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0
20   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0
21   0   0   0   0   0   0   0   0   0   0   1   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   1   0
```

6.3 Heuristic solution

Figure 12: Approximative solution using the minimum spanning tree



The objective function value is: 4.19280267866939

6.4 Comments

This time, we experiment some differences. In the first case we have a cluster that collects all the setosa Iris, and two clusters that mix the Versicolors and the Viriginicas.

In the second case, the method of the minimum spanning tree creates one big cluster with all the Versicolors and the virginicas, dividing the Setosas in two clusters.

It's important to remember that the species labels are just a reference for us and something that we are using to color things a bit, since we are not using them in anyway in our analysis, and, moreover, we are taking samples from them in a guided way. Especially in this second experiment, where we take subsets at the edge of the sample to have a confuse set of points.

Since we are not using labels, the only thing that we are minimizing is the cost function of the integer problem model, and we see that this time we have a worse performance applying our heuristic.

7 Conclusion

In this project, we have given an overview of how to solve the K-Medians problem, on one hand by implementing and solving the integer optimization problem using AMPL; and on the other by solving a minimum spanning tree problem using a path optimality theorem called Kruskal's greedy algorithm.
As we compared the computational results for solving our problem using these 2 procedures, we can say that K-Medians clustering is explorative. There are hardly any validation possibilities. It is thus only an optimization and the prerequisite is the knowledge of our respective dataset.