**Nantes, 14 December 2014**

# The cluster-median problem

Laboratory 3 – Optimization

EM - DMKM

**Group**:

- Gabriela Hernández Larios
- Maíra Machado Ladeira

## 1. Definition of the problem

The third laboratory assignment is about the classification of data according to the cluster-median problem. The cluster median problem is, roughly speaking, the problem of locating k "medians" points on a network with m points, so as to minimize the sum of all the distances from each point to its k-median and k ⊆ m. The approach of the cluster median problem in this assignment is stated as:

- Given a data matrix $A = (a_{ij}), i = 1, \dots, m,\ j = 1, \dots, n,$ of m points (or observations) and n attributes (variables), the goal is find $k$ median points, $p_l, l = 1, \dots, k$ , where $k$ is a predefined parameter and represents the number of clusters, so as to minimize the sum of distances from each point to the nearest $p_l$ median point. In the cluster-median problem $p \subseteq \{1, \dots, m\}$.
- The median point $p$ for a subset of points $\mathcal{J} \subseteq \{1, \dots, m\}$ is defined as the nearest point to all points of $\mathcal{J}$:

$$p\ is\ the\ median\ of\ \mathcal{J}\ if\ \sum_{i\,\epsilon\,\mathcal{J}} d_{i,p} = \min_{j\,\epsilon\,\mathcal{J}} \sum_{i\,\epsilon\,\mathcal{J}} d_{ij}$$

where $D = d_{ij}, i = 1, \dots, m, j = 1, \dots, m$ is the matrix of the Euclidean distances between the m points.

Nowadays, there exist many methods or approaches proposed for solving the cluster-median problem. On the present assignment only the following two methods were considered:

1. **Cluster-median problem formulated as an integer optimization problem.** This combinatorial optimization problem provides an acceptable solution to the cluster-median problem however it is considered as NP-hard on general networks and its solution is only viable when the number of points is not very large.
2. **Heuristic solution as a minimum spanning tree (MST) problem**. This problem may provide a reasonable good heuristic solution with less computational resources. However, this method implements a heuristic based on the linear separation of the points. On this method, the clusters are separated according to the biggest distances between the points and, therefore, it is more likely to find classification errors on this method.

The Figure 1 shows a set of 22 point classified in 6 clusters according to the cluster-median problem formulated as an integer optimization problem:

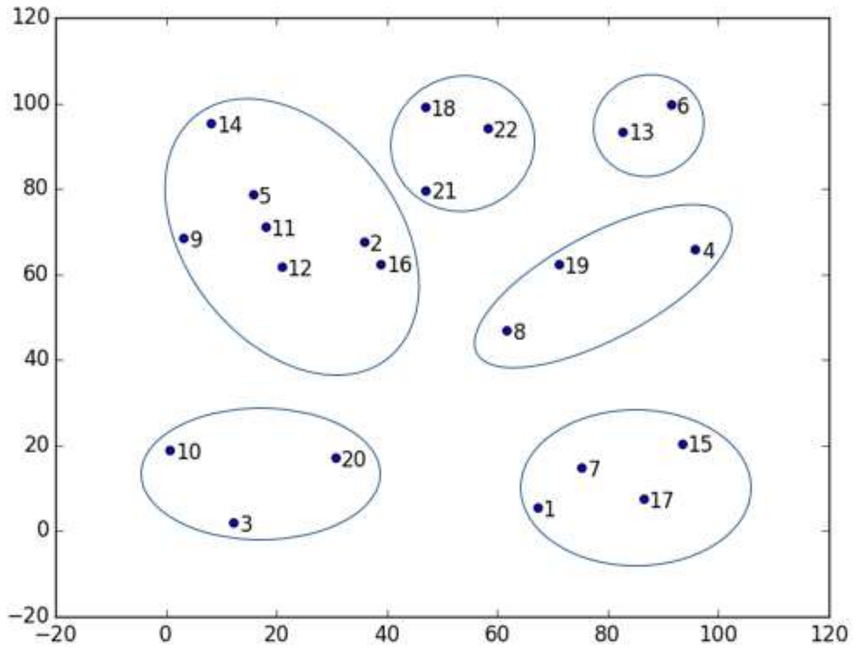


*Figure 1 –22 points into 6 clusters according to the cluster-median problem*

The Figure 2 shows the same 22 points classified in 6 clusters according to the cluster median problem implemented as a heuristic solution of minimum spanning tree.

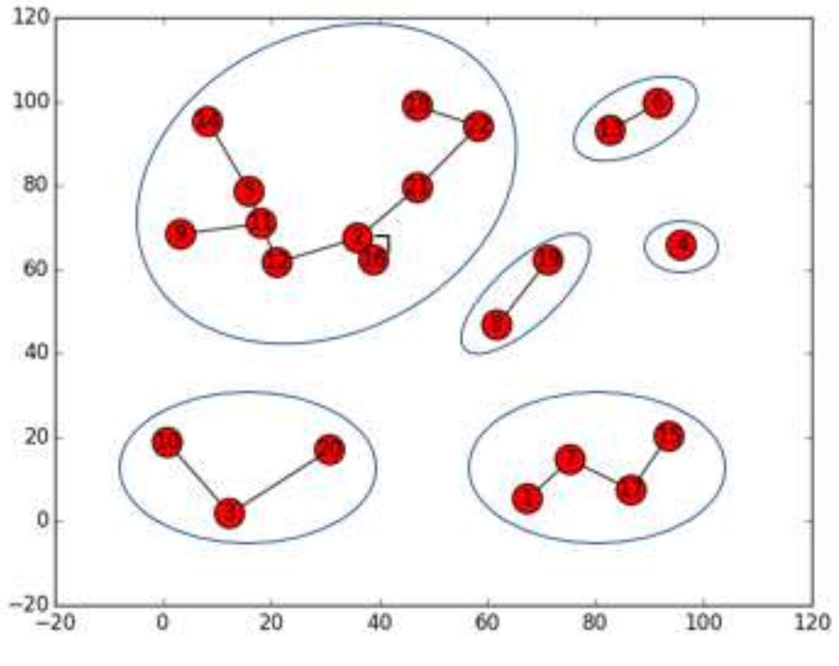


*Figure 2 - 22 points classified into 6 cluster according to the spanning tree heuristic*

On this laboratory, the 2 methods are implemented and the results obtained are compared.

## 2. Data set

Two different data sets were classified into clusters during this assignment:

- Randomly generated 2D points
- Sample from the Iris data

### 2.1. Generated points

The set of generated points was obtained with a self-developed python script. On the script, the points are randomly generated and the distance between them is calculated with the Euclidian distance. The script requires from the user the expected number of points and returns 3 files:

- InputDataAMPL.txt: file composed by the distances between the points. On this file each point has its distance to all the points from the data set, including itself. The file is organized on the format required by the .dat file from AMPL. A sample of this file can be seen below:

```
1 1   0.0
2 1   69.5952557543
3 1   55.173841357
4 1   66.7880931481
5 1   89.5452784683
6 1   97.1849469478
7 1   12.2230406587
8 1   41.7379563994
9 1   89.9087077713
10 1   67.8750932056
11 1   82.0226317901
12 1   72.8662452443
13 1   89.2538035632
14 1   107.552540652
15 1   30.101624
16 1   63.7094363138
17 1   19.3130360918
18 1   95.698452146
19 1   56.9771084571
20 1   38.3972967462
21 1   76.9265327459
22 1   89.2377297844
1 2   69.5952557543
2 2   0.0
```

- InputMatrix.txt: file that contains the distances between the points to be used by the spanning tree. This file has the format required by the spanning tree python script that will be described on the Section 4 of this report. Each line of the file has the distance of a point to all the other points, including itself. A sample of this file can be seen below:

```
0.0     69.5952557543     55.173841357     66.7880931481     89.5452784683     97.1849469478
12.2230406587     41.7379563994     89.9087077713     67.8750932056     82.0226317901
72.8662452443     89.2538035632     107.552540652     30.101624     63.7094363138
```

```
19.3130360918    95.698452146    56.9771084571    38.3972967462    76.9265327459
89.2377297844

69.5952557543    0.0    69.8008112202    59.9535306192    22.9665407149
64.1423763903    65.8616108723    33.1286246304    32.6692834507    60.0657150965
18.1734662117    15.9550941446    53.4954619164    39.1933404982    74.3136223907
5.90223094046    78.4810332855    33.27570499    35.6340422737    50.7438367635
16.4685329933    34.7928393723
```

- DataPoints.txt: file that contains the 2D coordinates to the points. This file is only auxiliary and can be used for plotting the point if necessary.

Besides the output files, the program plots the points in a plan and allow the user to save this plot as an image. The Figure 2 shows an example of data generated by this program.

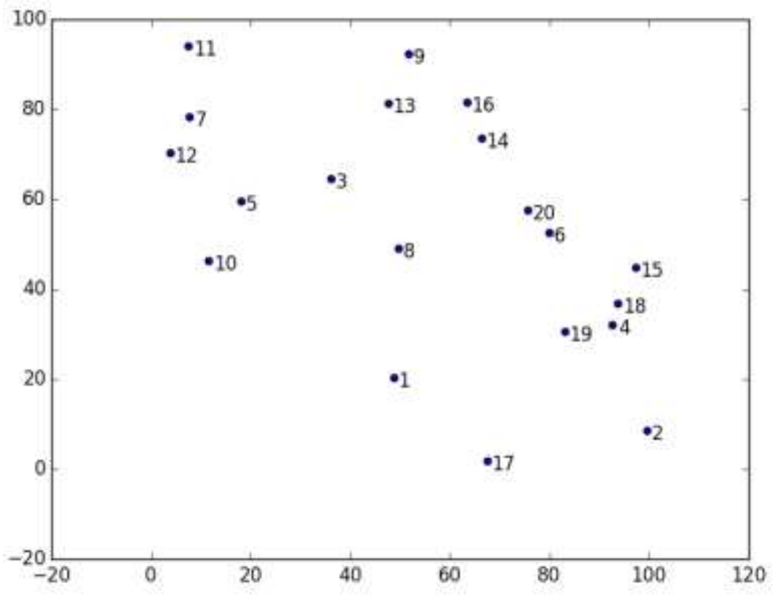


*Figure 3 - Example of 20 points generated*

## 2.2. Iris data set

The Iris data set[1] was chosen because it is a well-known data set for pattern recognition. The original data set contains 3 classes of 50 instances each one, each class referring to a type of Iris plant. Each point or observation of this data set is composed by 4 attributes: sepal length in cm, sepal width in cm, petal length in cm and petal width in cm, and according to these four attributes each point is classified into three different classes: Iris Setosa, Iris Versicolor and Iris Virginica. It is important to highlight, that according to the UCI repository information, one of

[1] http://archive.ics.uci.edu/ml/datasets/Iris

the classes is linearly separable from the other two, but the other two classes are not linearly separable from each other. The linearly separable class is not known in advanced.

The Figure 4 shows the point plotted by petal width and petal length while the Figure 5 shows the points by sepal length and sepal width.

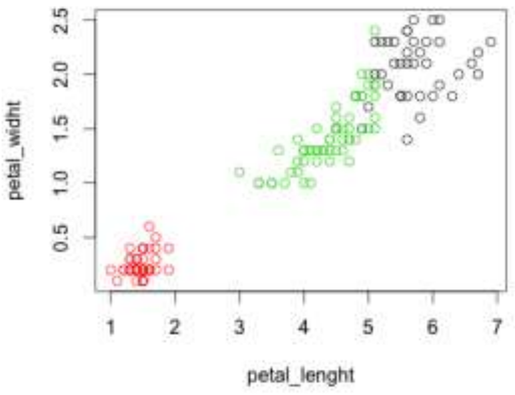


*Figure 4 - point plotted by petal width and length*

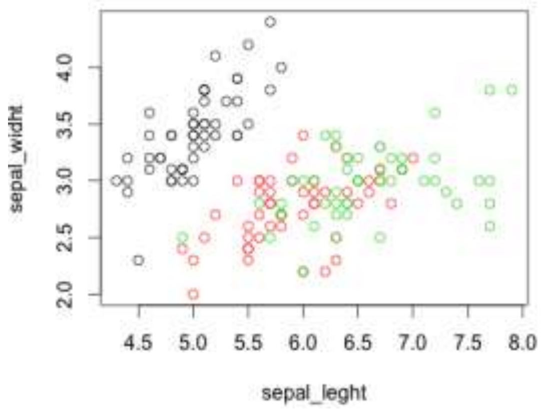


*Figure 5 - points plotted by sepal length and width*

Due to computational limitations for calculate the integer optimization problem with AMPL, using the students' version, a smaller sample of this population was needed. The sample of data used for tests can be seen on the table below, for obviously reasons the "classification" attribute was not considered:

| Data | Sepal length | Sepal Width | Petal Length | Petal Width | Classification |
|---|---|---|---|---|---|
| 1 | 4.9 | 3.1 | 1.5 | 0.1 | Iris-setosa |
| 2 | 5.4 | 3.7 | 1.5 | 0.2 | Iris-setosa |
| 3 | 4.8 | 3.4 | 1.6 | 0.2 | Iris-setosa |
| 4 | 4.8 | 3.1 | 1.6 | 0.2 | Iris-setosa |
| 5 | 5.4 | 3.4 | 1.5 | 0.4 | Iris-setosa |
| 6 | 5.2 | 4.1 | 1.5 | 0.1 | Iris-setosa |
| 7 | 6.0 | 2.9 | 4.5 | 1.5 | Iris-versicolor |
| 8 | 5.7 | 2.6 | 3.5 | 1.0 | Iris-versicolor |
| 9 | 5.5 | 2.4 | 3.8 | 1.1 | Iris-versicolor |
| 10 | 5.6 | 2.7 | 4.2 | 1.3 | Iris-versicolor |
| 11 | 5.7 | 3.0 | 4.2 | 1.2 | Iris-versicolor |
| 12 | 5.7 | 2.9 | 4.2 | 1.3 | Iris-versicolor |
| 13 | 6.2 | 2.9 | 4.3 | 1.3 | Iris-versicolor |
| 14 | 6.9 | 3.2 | 5.7 | 2.3 | Iris-virginica |
| 15 | 6.4 | 2.8 | 5.6 | 2.2 | Iris-virginica |
| 16 | 7.7 | 2.8 | 6.7 | 2.0 | Iris-virginica |
| 17 | 6.3 | 2.7 | 4.9 | 1.8 | Iris-virginica |
| 18 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |

|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 19 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |
| 20 | 6.4 | 3.1 | 5.5 | 1.8 | Iris-virginica |

The Figure 5 and the Figure 6 show this sample of points plotted by petal length and petal width and by sepal length and sepal width respectively. On the plots it is possible to clearly see to linear separation of one class for the dependency of the other 2.

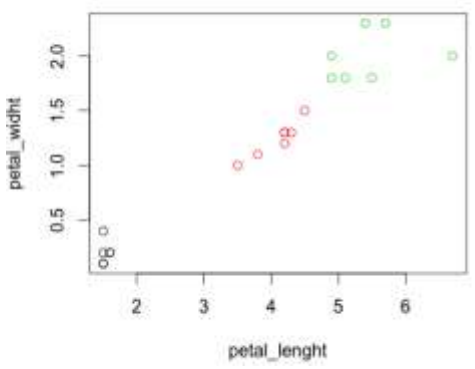


*Figure 6 - sample points plotted by petal length e width*

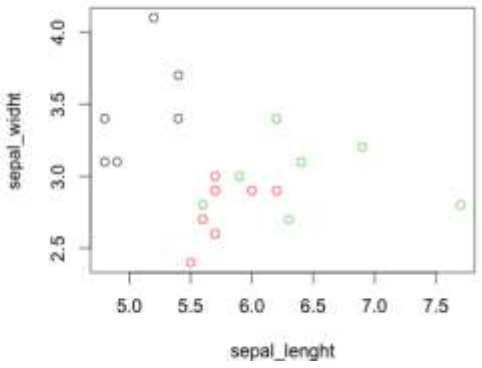


*Figure 7- sample points plotted by sepal length e width*

For getting the data on the necessary formats for the assignment, a self-developed python script was developed. This script generates, from the sample of data, the files Iris-AMPL.txt and Iris-st.txt for being used by AMPL and by the spanning tree script, respectively.

It is easy to notice that for this particular case, three clusters will be needed for classifying the data and the generated clusters should be:

- Cluster 1: points 1 to 6
- Cluster 2: points 7 to 13
- Cluster 3: points 14 to 20

It is not previously known which point should be the median for each cluster on the cluster-median problem.

## 3. Formulation as an integer optimization problem

The cluster-median problem can be formulated as an integer optimization with the objective function of minimize the sum of the distance of each point to the median of the cluster containing the point. On this formulation, the median of a cluster will be defined by a point belonging to the cluster that should have the minimum distance possible to the others points of the cluster.

Considering a set of random data, represented on Figure 8, that should be classified in 2 different clusters: the optimization problem will determine the 2 points that should be the medians of the generated clusters and which points should belong to each cluster. Each cluster will be named by its median. On the case where the number of the clusters is equal to the number of the points, each point will be the name of a cluster. On the case where only one cluster is selected, the median of this cluster will be the median of the data set.

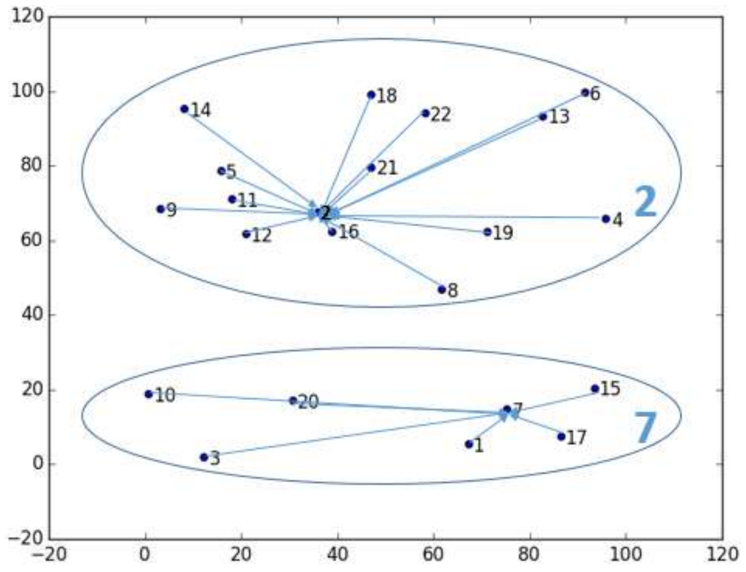


*Figure 8 - Median of a cluster*

As constraints for the optimization problem is possible to identify that:

- Each point should belong to one and only one cluster.
- The number of clusters must be exactly the requested (k).
- A point can belong to a cluster only if this cluster exists.
- A point can only belong or not belong to a cluster.

The table 1 shows the formulation of the problem and its constraints. For simplification of the notation, this assignment considers m points and m clusters, where m-k clusters are empty.

| | |
|---|---|
| $min \sum_{i=1}^{m} \sum_{j=1}^{m} d_{ij} x_{ij}$ | Minimum distance of all points to their cluster medians |
| $Subject\ to: \sum_{j=1}^{m} x_{ij}, i = 1..m$ | Every point must belong to one cluster |
| $\sum_{j=1}^{m} x_{ij} = k$ | Exactly k clusters |

| | |
|---|---|
| $x_{jj} \geq x_{ij}$ | A point may belong to a cluster only if it exists |
| $x_{ij} \in \{0,1\}$ | A point should only belong ($x_{ij} = 1$) or not belong ($x_{ij} = 0$) to a cluster |

*Table 1 -cluster-median as an integer optimization problem*

On this assignment, AMPL is used for solving the integer problem. For this, the problem is translated to AMPL and the .mod file can be seen below:

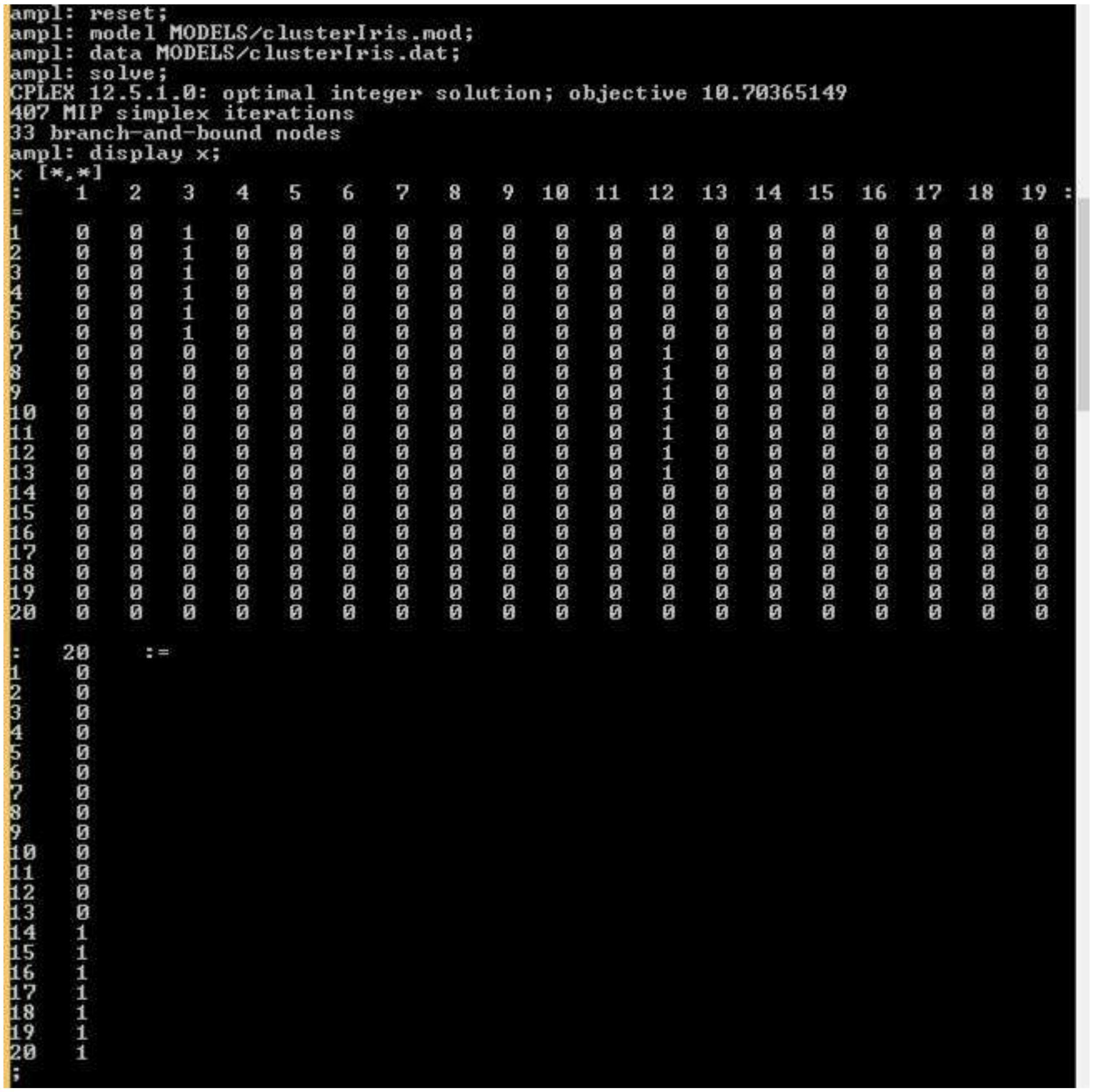
```
set m:= 1..20;
param k:=3;
param d {i in m, j in m};
var x {i in m, j in m} binary;

#objective function: minimize the distance of all points to their cluster medians
minimize objective_function: sum{i in m}(sum{j in m} d[i,j]*x[i,j]);

#Every point must belong to one cluster
subject to point {i in m}:
sum{j in m} x[i,j]=1;

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

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

The .dat file is available on the Appendix 1 due to its size.

The student's version of AMPL has a limitation of 500 variables and constraints. This way, the maximum points that can be used on the problem is m = 22, since each point has its distance calculated to each other point ($22 X 22 = 484$ variables). The iris data set, with 20 points, will be used as example on this section. As the minors solver cannot be used for solve integer problems, the cplex solver was used.

As the data used for testing the problem is a real world data with defined classifications, the only application possible is for 3 defined cluster. This way, the .mod file was initialized with m = 20 data points and k = 3 clusters.

The Figure 9 shows the results obtained with AMPL:



```
ampl: reset;
ampl: model MODELS/clusterIris.mod;
ampl: data MODELS/clusterIris.dat;
ampl: solve;
CPLEX 12.5.1.0: optimal integer solution; objective 10.70365149
407 MIP simplex iterations
33 branch-and-bound nodes
ampl: display x;
x [*,*]
:    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19 :
=
1    0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2    0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
3    0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
4    0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
5    0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
6    0   0   1   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   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   1   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

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

*Figure 9- results obtained with AML for Iris dataset*

As integer optimization problems are difficult to be solved, it requires a significant amount of calculations from AMPL. This way, the result is obtained in 407 interactions and for the current configuration, the value of objective function is approximated 10.7. That means that for the current set of data, the sum of the minimum distance between each point to its cluster is equal to 10.7.

As the Iris data set has 4 attributes, the data should be translated to a 4 dimensions space for being plotted. Considering the difficulties on doing that, the Figure 10 display the data clusters in a 2 dimensions space with random positions, in order to better visualize the clusters generated.

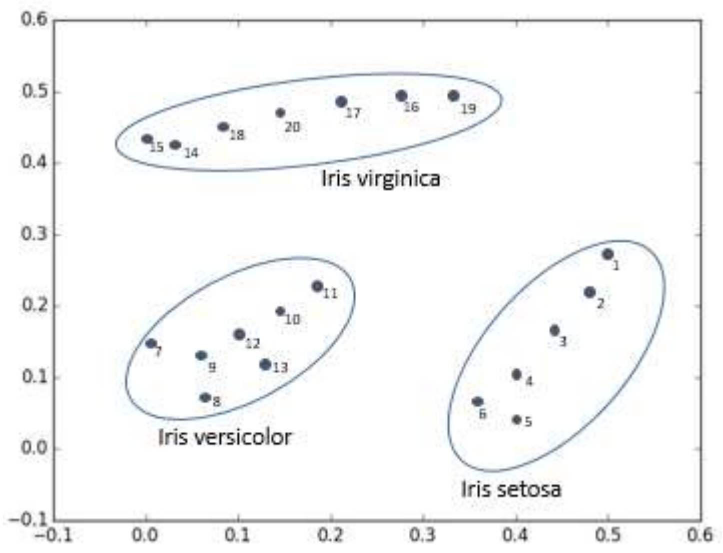


*Figure 10 -Randomly generated 2D plot of the clusters*

As it is possible to observe on Figure 10, the integer minimization problem divides the data on the expected classes without errors.

## 4. Minimum Spanning Tree Computation

On this assignment, PRIM's algorithm is used for building the minimum spanning tree and Python is used for the implementation.

For building the clusters using the MST method, the points represent the vertices in a graph and the distances between the points given by the matrix $D = d_{ij}, i = 1, \dots, m, j = 1, \dots, m,$ are the edges connecting the vertices. If k represents the number of clusters, then k-1 heaviest edges will be remove to obtain the closest or most similar vertices (points) into clusters.

A spanning tree is an acyclic, connected sub-graph of an undirected and connected graph, which has all the vertices (v) of the graph and contains only "v-1" edges, i.e. if there are 5 vertices (nodes) the spanning tree has 4 edges.

The idea behind PRIM's algorithm for computing the minimum spanning tree consists in having two sets of vertices, one containing the vertices already visited by the minimum spanning tree and the second one containing the vertices that have not been visited yet. It starts with a minimum spanning tree empty and with any random vertex. At every iteration it considers all the vertices that have been visited to find the minimum weight edge with the vertices that have not been visited, in other words, it finds the minimum weight edge that connect the two sets. Once the minimum edge was selected, before moving to the next iteration, it updates both sets, adding the new vertex to one set and removing it from the other set, and adds the new edge to the minimum spanning tree. The stop condition is reached when all the vertices have been visited.

**Implementation of Prim's Minimum Spanning Tree in Python**

For explaining how the Minimum Spanning Tree was computed in Python, the following data set of five points is given as an example.

$$\begin{aligned}
Point1 &= [12.4825214967, 75.1120197865]\\
Point2 &= [74.1104134845, 82.3644471288]\\
Point3 &= [82.7124823427, 39.2286058754]\\
Point4 &= [86.5583232908, 14.6986919349]\\
Point5 &= [37.7845195201, 64.2716008339]
\end{aligned}$$

The following 5x5 matrix represents the Euclidean distances for each pair of points. Each element of the matrix $d_{ij}$ represents the Euclidean distance between the vertex *i* and the vertex *j*.

$$D = \begin{bmatrix} 0.00 & 62.05 & 78.87 & 95.59 & 27.53 \\ 62.05 & 0.00 & 43.99 & 68.80 & 40.58 \\ 78.87 & 43.99 & 0.00 & 24.83 & 51.44 \\ 95.59 & 68.80 & 24.83 & 0.00 & 69.54 \\ 27.53 & 40.58 & 51.44 & 69.54 & 0.00 \end{bmatrix}$$

There are two important facts to take notice of the above matrix: this is a symmetric matrix, which means that it is equal to its transpose ($D = D^T$) and the diagonal is zeros, this mean that the distance between a point and itself is zero. These two important facts were taking into account for making easier the implementation of the algorithm in Python.

This “distance” (or weights) matrix represents the weighted edges in the graph and is the input of the minimum spanning tree function in the python program. The Figure 11 shows the undirected, connected graph for these points.

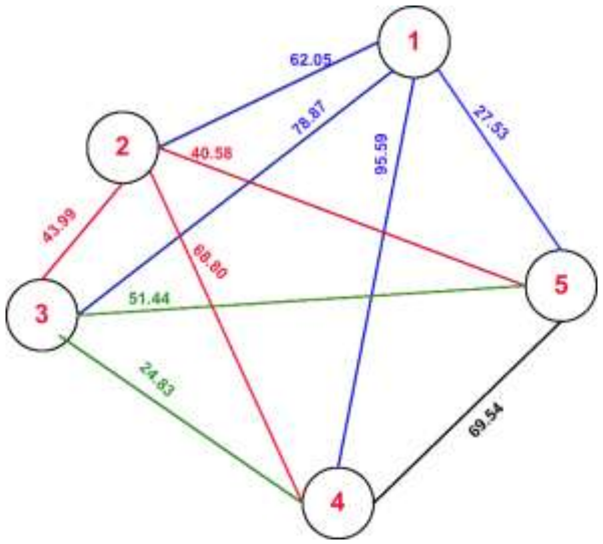


*Figure 11 - five points connected graph*

The implementation criteria will be explaining below using the 5 points data set as an example:

- The Python’s library numpy was used to work with matrices.
- The total number of vertices is obtained by computing the range of the matrix.

**vertices = 5**

- The spanning tree starts empty, and the library "Networkx" was used for drawing it in a fancy way.
  **spanning_tree_graph = []**
- A nested list also was defined to store the edges and their respective weights of the minimum spanning tree, so this edges-list starts empty as well. This nested list has an important role for creating the clusters.
  **spanning_tree_edges = []**
- There is one set of vertices, which contains only the visited vertices, and it always starts with the first point (vertex).
  **visited_vertices = [1]**
- The first visited edge was assigned to the first point for all the cases.
- There is one accumulator, which counts the number of visited edges; the accumulator starts with one, because it assumes that the first vertex (node) was already visited. This accumulator replaces the other set which contains the number of not visited edges.
  **number_visited = 1**
- The diagonal of the matrix is converted to infinite before computing the iterations.

$$D = \begin{bmatrix} \infty & 62.05 & 78.87 & 95.59 & 27.53 \\ 62.05 & \infty & 43.99 & 68.80 & 40.58 \\ 78.87 & 43.99 & \infty & 24.83 & 51.44 \\ 95.59 & 68.80 & 24.83 & \infty & 69.54 \\ 27.53 & 40.58 & 51.44 & 69.54 & \infty \end{bmatrix}$$

- The stop condition is reached when the accumulator is equal to the total number of vertices.
- The matrix is symmetric, so the same result is reached either if it is obtained using rows or columns.

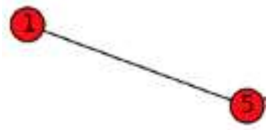
**Note:** In Python the indices start with 0 instead of 1. For explaining reasons, this fact will not be considered.

Roughly speaking, what the iterator does is: it first obtains the index of the minimum distance along the vertices already visited, adds this edge to the spanning tree, updates the visited vertices set by appending the new vertex, makes infinite all the distances between the visited vertices, in such a way that these vertices cannot be visited again, and finally it increments the number of visited vertices.
Let's see the above description in detail using the worked example:
**First iteration**
Step 1: Checks condition.
number_visited ≠ vertices
1≠5
Step 2: Indices of the minimum distance along the visited vertices: 1.

$$D = \begin{bmatrix} \infty & 62.05 & 78.87 & 95.59 & 27.53 \\ 62.05 & \infty & 43.99 & 68.80 & 40.58 \\ 78.87 & 43.99 & \infty & 24.83 & 51.44 \\ 95.59 & 68.80 & 24.83 & \infty & 69.54 \\ 27.53 & 40.58 & 51.44 & 69.54 & \infty \end{bmatrix}$$

new_edge = [1,5]

new_vertix = 5
Step 3: Updates spanning three and visited_vertices.
spanning_tree_graph:



spanning_tree_edges=[[1,5,27.53]]
visited_vertices=[1,5]
Step 4: Makes infinite all the distances between the visited vertices: 1 and 5.

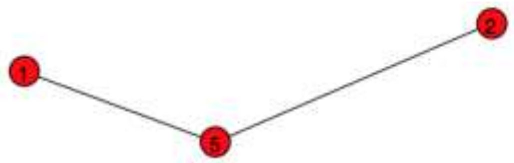
$$D = \begin{bmatrix} \infty & 62.05 & 78.87 & 95.59 & \infty \\ 62.05 & \infty & 43.99 & 68.80 & 40.58 \\ 78.87 & 43.99 & \infty & 24.83 & 51.44 \\ 95.59 & 68.80 & 24.83 & \infty & 69.54 \\ \infty & 40.58 & 51.44 & 69.54 & \infty \end{bmatrix}$$

Step 4: Increments the accumulator.
number_visited =2

**Second iteration**
Step 1: Checks condition.
number_visited ≠ vertices
2≠5
Step 2: Index of the minimum distance along the vertices 1 and 5.

$$D = \begin{bmatrix} \infty & 62.05 & 78.87 & 95.59 & \infty \\ 62.05 & \infty & 43.99 & 68.80 & 40.58 \\ 78.87 & 43.99 & \infty & 24.83 & 51.44 \\ 95.59 & 68.80 & 24.83 & \infty & 69.54 \\ \infty & 40.58 & 51.44 & 69.54 & \infty \end{bmatrix}$$

new_edge = [5,2]
new_vertix = 2
Step 3: Updates spanning three and visited_vertices.
spanning_tree_graph:



spanning_tree_edges = [[1,5,27.53], [5,2,40.58]]
visited_vertices = [1,5,2]
Step 4: Makes infinite all the distances between the visited vertices: 1, 5 and 2.

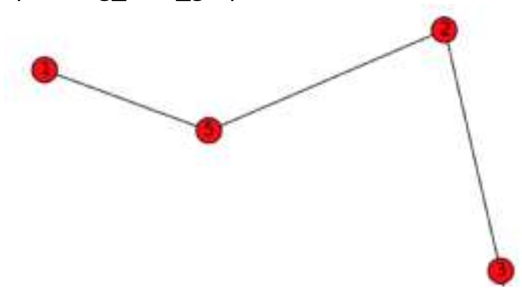
$$D = \begin{bmatrix} \infty & \infty & 78.87 & 95.59 & \infty \\ \infty & \infty & 43.99 & 68.80 & \infty \\ 78.87 & 43.99 & \infty & 24.83 & 51.44 \\ 95.59 & 68.80 & 24.83 & \infty & 69.54 \\ \infty & \infty & 51.44 & 69.54 & \infty \end{bmatrix}$$

Step 4: Increments the accumulator.
number_visited = 3

**Third iteration**
Step 1: Checks condition.
number_visited ≠ vertices
3≠5
Step 2: Index of the minimum distance along the vertices 1, 5 and 2.

$$D = \begin{bmatrix} \infty & \infty & 78.87 & 95.59 & \infty \\ \infty & \infty & 43.99 & 68.80 & \infty \\ 78.87 & 43.99 & \infty & 24.83 & 51.44 \\ 95.59 & 68.80 & 24.83 & \infty & 69.54 \\ \infty & \infty & 51.44 & 69.5 & \infty \end{bmatrix}$$

new_edge = [2,3]
new_vertix = 3
Step 3: Updates spanning three and visited_vertices.
spanning_tree_graph:



spanning_tree_edges = [[1,5,27.53], [5,2,40.58], [2,3, 43.99]]
visited_vertices = [1,5,2,3]
Step 4: Makes infinite all the distances between the visited vertices: 1,5,2 and 3.

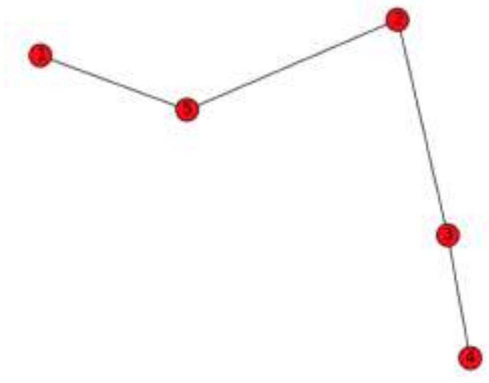
$$D = \begin{bmatrix} \infty & \infty & \infty & 95.59 & \infty \\ \infty & \infty & \infty & 68.80 & \infty \\ \infty & \infty & \infty & 24.83 & \infty \\ 95.59 & 68.80 & 24.83 & \infty & 69.54 \\ \infty & \infty & \infty & 69.54 & \infty \end{bmatrix}$$

Step 4: Increments the accumulator.
number_visited = 4

**Fourth iteration**
Step 1: Checks condition.
number_visited ≠ vertices
4≠5
Step 2: Index of the minimum distance along the vertices 1, 5, 2 and 3.

$$D = \begin{bmatrix} \infty & \infty & \infty & 95.59 & \infty \\ \infty & \infty & \infty & 68.80 & \infty \\ \infty & \infty & \infty & 24.83 & \infty \\ 95.59 & 68.80 & 24.83 & \infty & 69.54 \\ \infty & \infty & \infty & 69.54 & \infty \end{bmatrix}$$

new_edge = [3,4]
new_vertix = 4
Step 3: Updates spanning three and visited_vertices.
spanning_tree_graph:

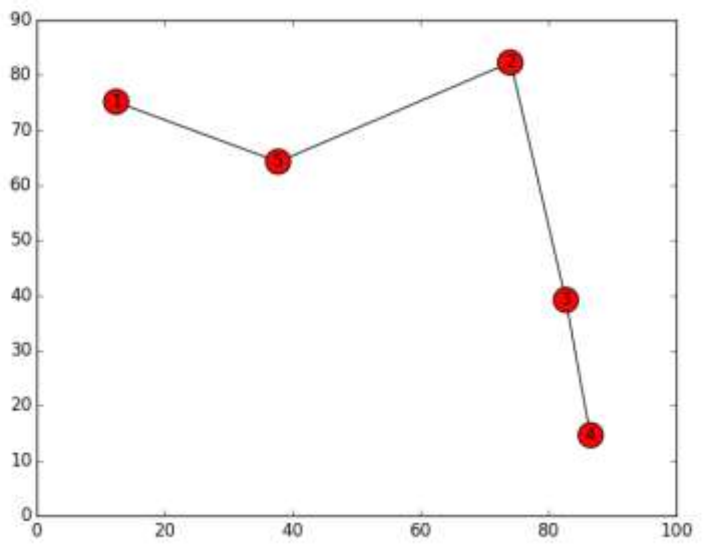


spanning_tree_edges = [[1,5,27.53], [5,2,40.58], [2,3,43.99], [3,4,24.83]]
visited_vertices = [1,5,2,3,4]
Step 4: Makes infinite all the distances between the visited vertices: 1,5,2,3,4.

$$D = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Step 4: Increments the accumulator.
number_visited = s5

**Fifth iteration**
Step 1: Checks condition.
number_visited ≠ vertices
Condition not satisfied: 5 ≠5
Stop condition reached.



**Building the clusters**
The following assumptions have been made in order to build k clusters:

- The user is asked to enter the desired number of clusters taking into consideration that no more than *m* clusters can be done if *m* is the total number of points.
- The number of edges to be removed is equal to the number of desired clusters minus one.
- The nested list (spanning_tree_edges), which contains the spanning tree edges with their weights, is used to create the clusters.

Let's see how the clusters are built using again the 5 data points' example and supposing that 3 clusters are required:
K=3

First, the spanning_tree_edges=[[1,5,27.53], [5,2,40.58], [2,3,43.99], [3,4,24.83]] nested list is converted to a matrix:

```
spanning_tree_edges = [[1,5,27.53]
                       [5,2,40.58]
                       [2,3,43.99]
                       [3,4,24.83]]
```

The rows are sorted by their weight:

```
spanning_tree_edges = [[3,4,24.83]
                       [1,5,27.53]
                       [5,2,40.58]
                       [2,3,43.99]]
```

Then the weights' column is extracted as a list:
Weights = [24.83

27.53
40.58
43.99]

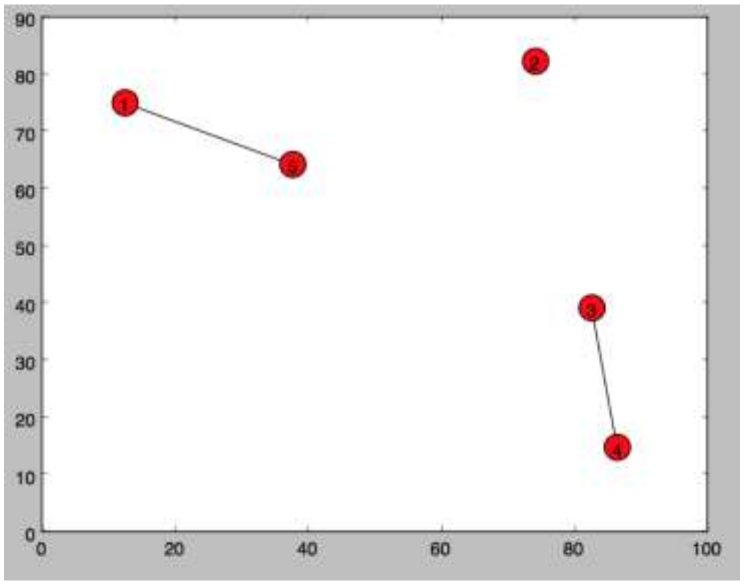
According to the number of desired clusters, it will take the k-1 maximum values of the weights' column. Since 3 clusters are desired, then the 2 maximum weights from the list are picked and placed in a new list. This new list represents the edges whose weights are the biggest, so are the candidates edges to be removed.

maximum_weights = [40.58, 43.99]

From this new list, the minimum weight is selected:

minimum_weight = 40.58

Using the "Networkx" library, the edges whose weight is bigger or equal to the minimum of the k-1 maximum weighted edges, in this case the edges whose weight is bigger than 40.58, are removed and the clusters are formed.
For this example, the edges corresponding to the weights >= 40.58 are [5,2,40.58] and [2,3,43.99], therefore those are removed.



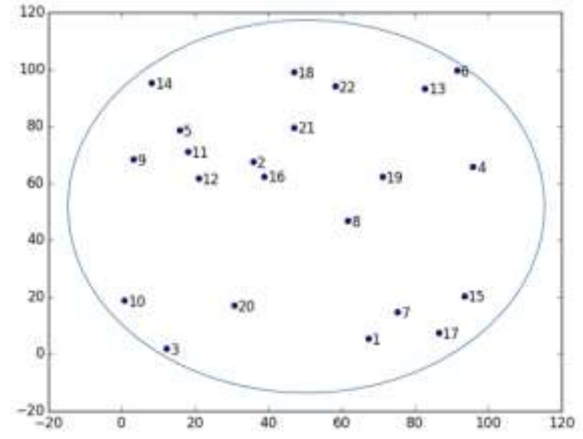
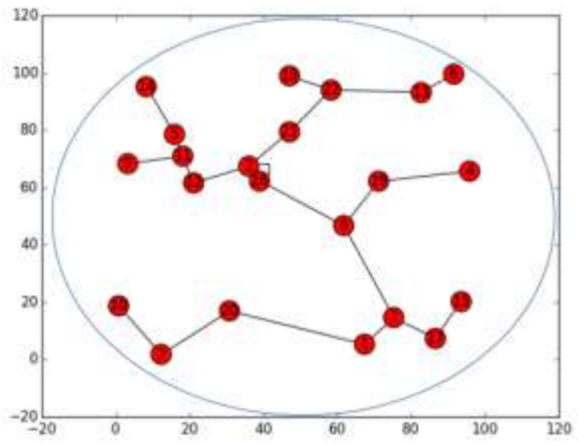
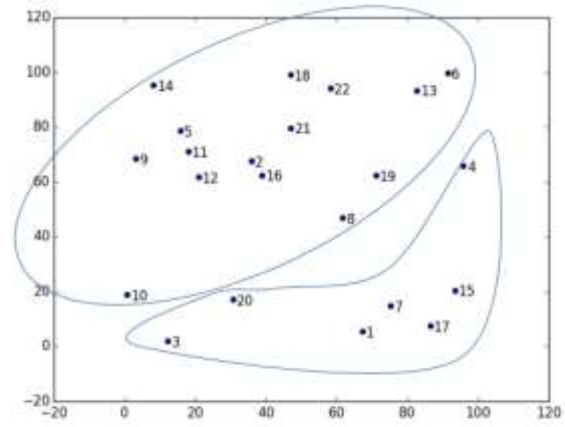
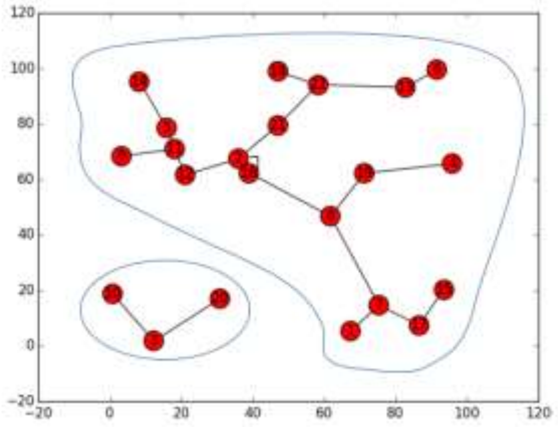
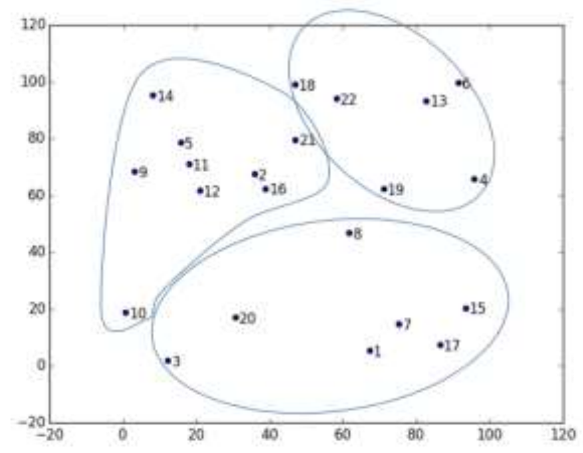
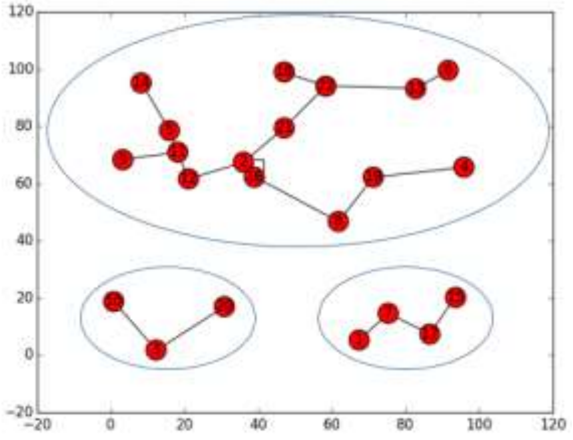
Please note that the points are actually in their right positions, so it is easy to see that the clustering was well done.

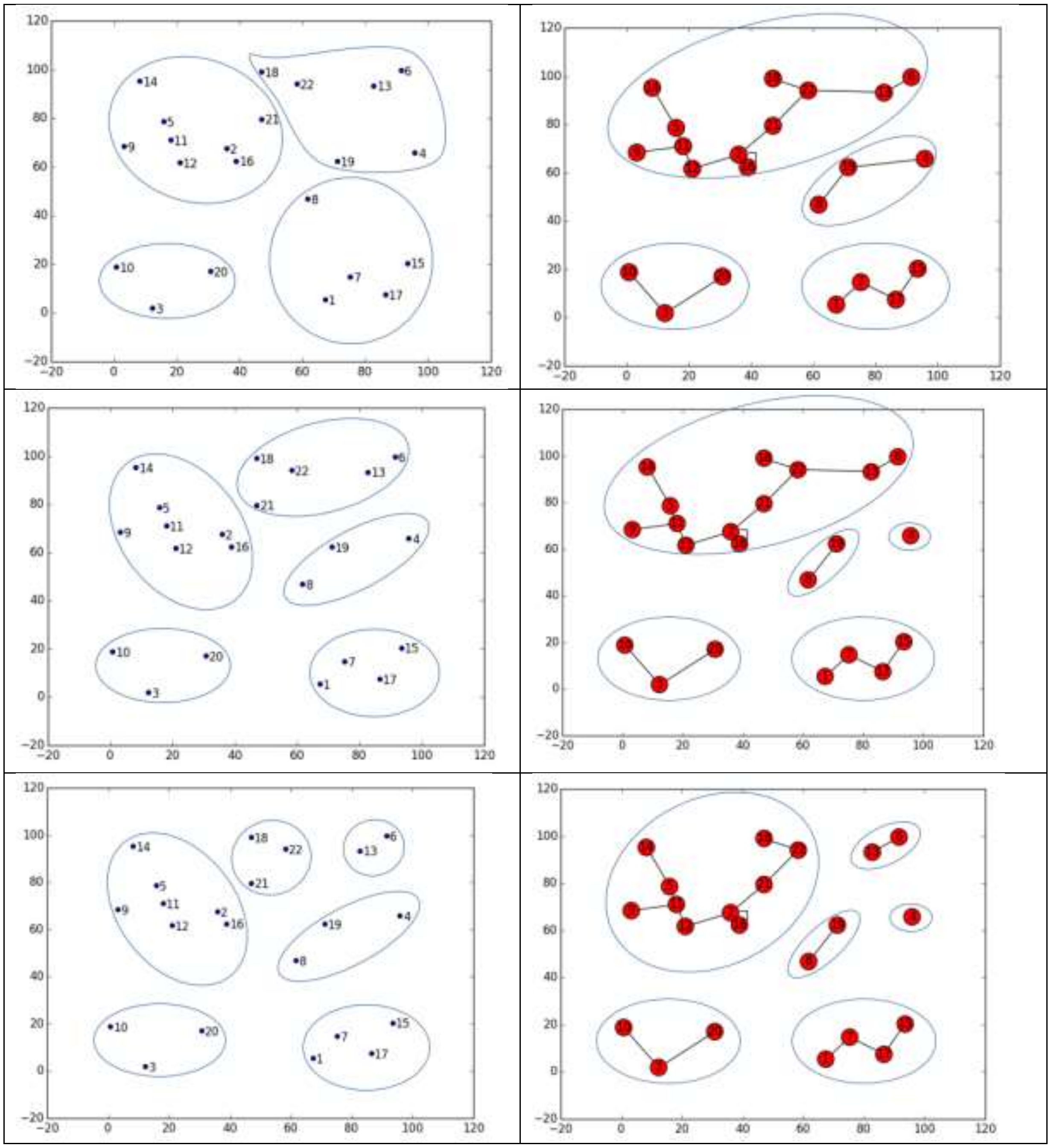
## 4. Comparison of the two solutions

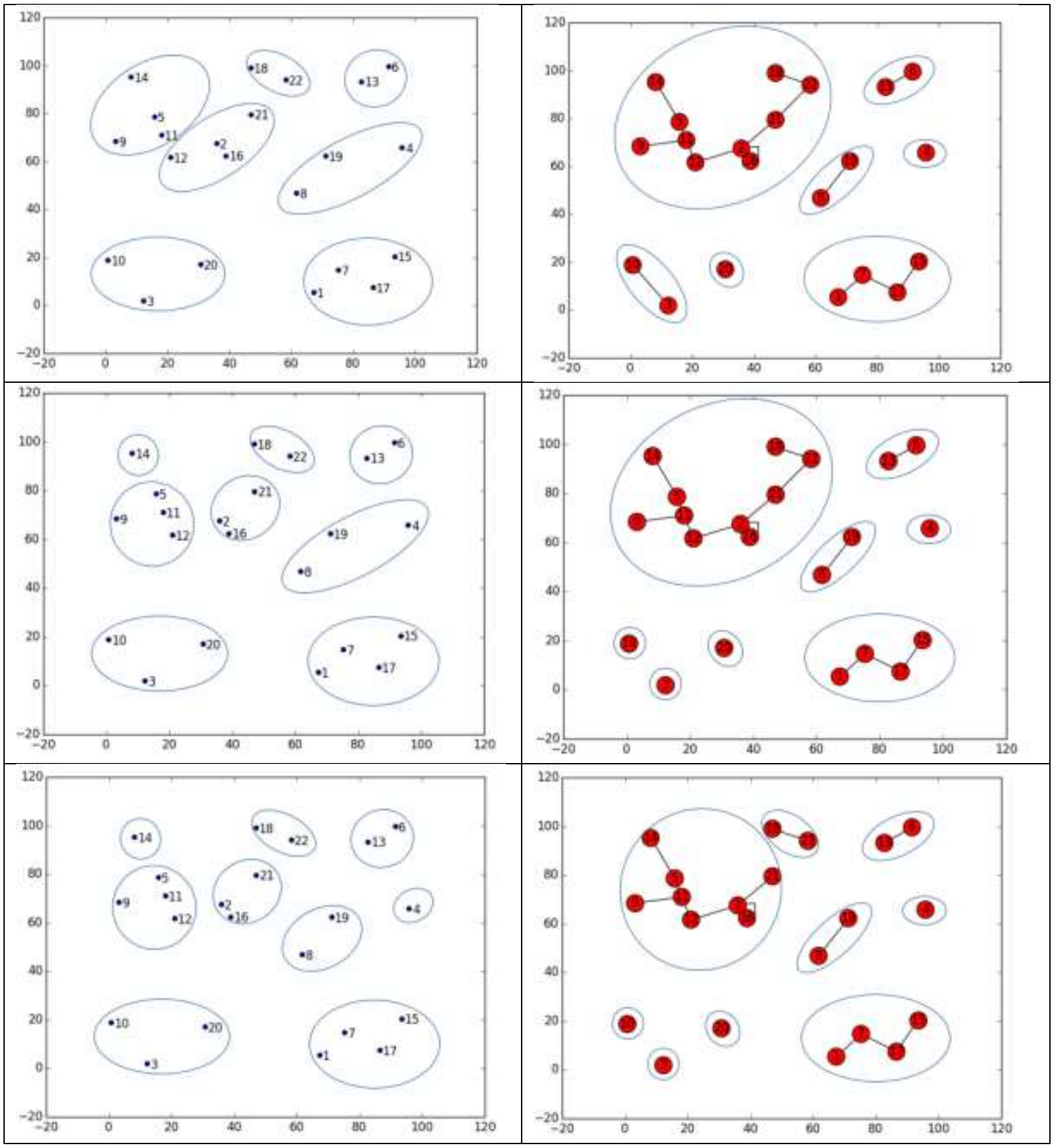
The comparison of the solutions is done on 3 different sets of data:

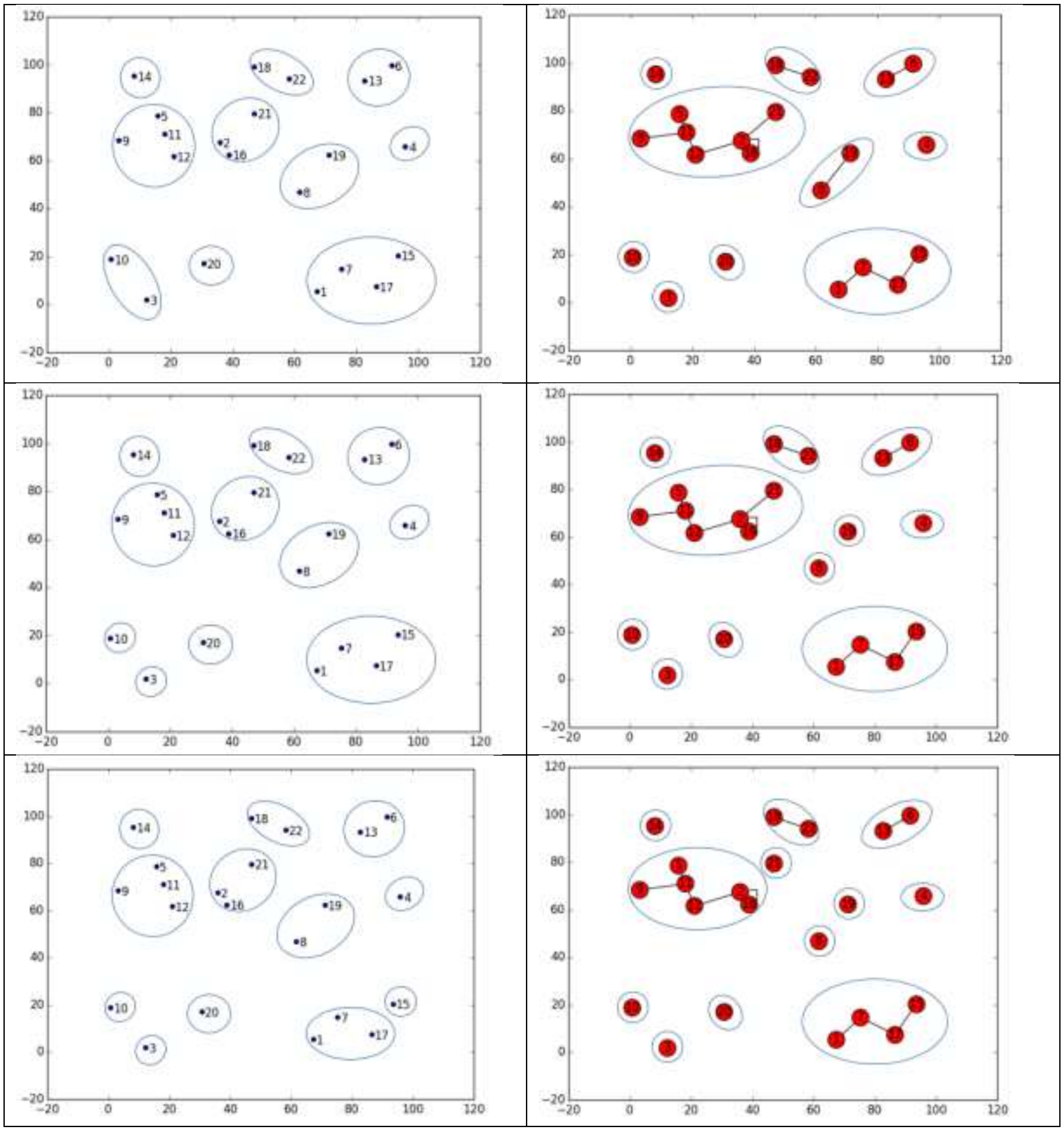
- **22 random generated points:**

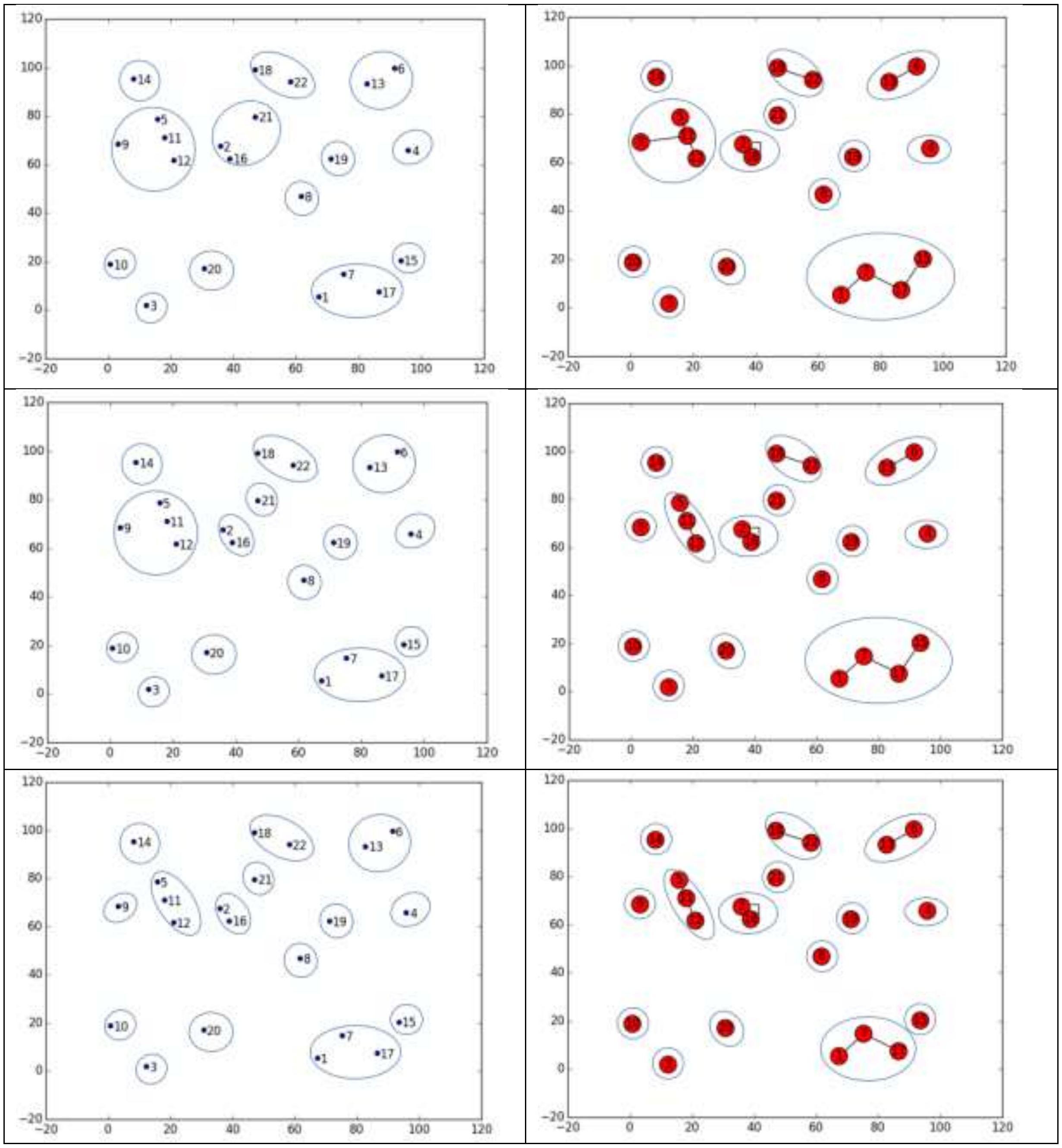
For the generated points it is possible to generate from 1 to 22 clusters. The results of the optimization problem and of the spanning tree for each k can be visualized below:

| **AMPL: Integer Optimization Problem** | **Spanning Tree** |
|---|---|
|  |  |
|  |  |
|  |  |

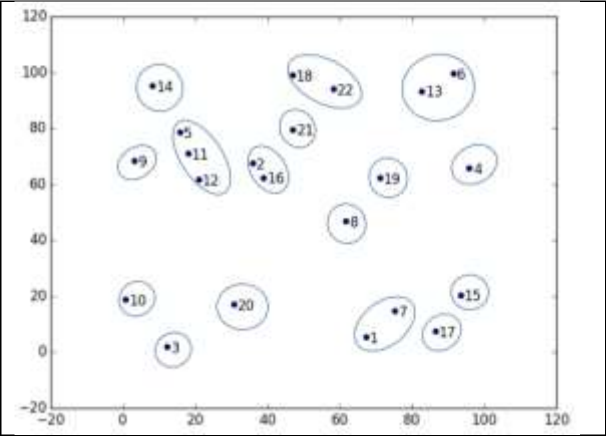
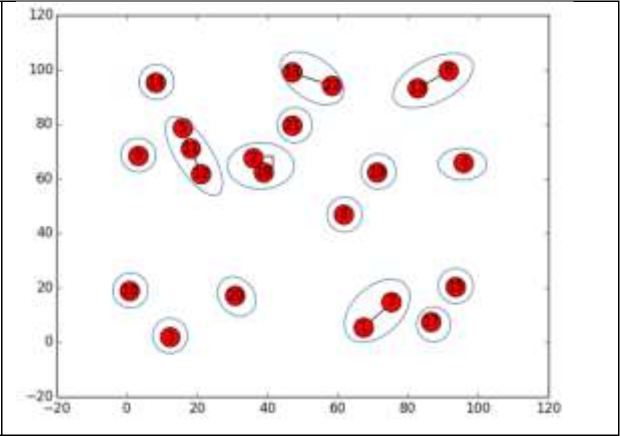
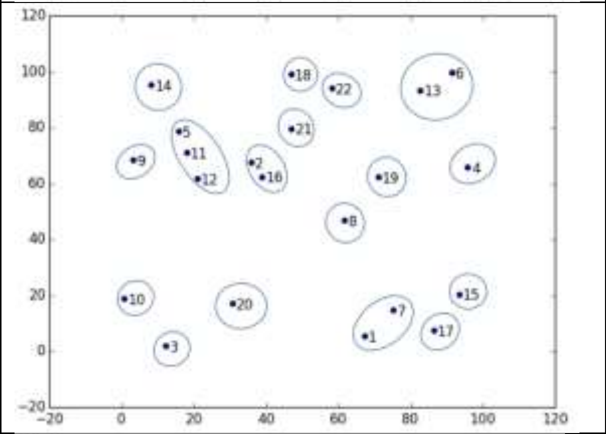
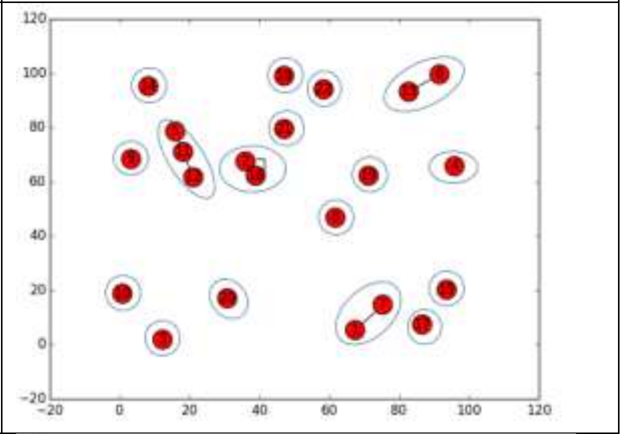
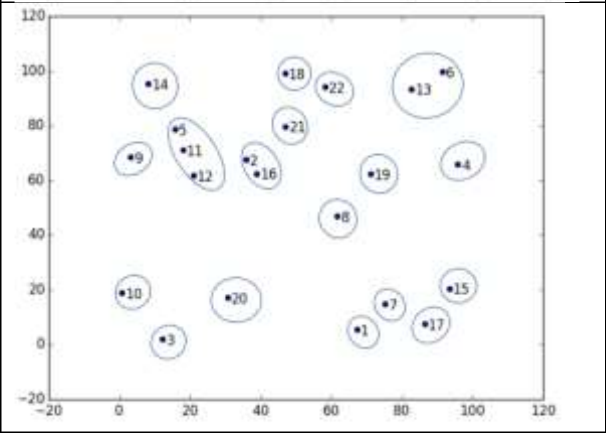
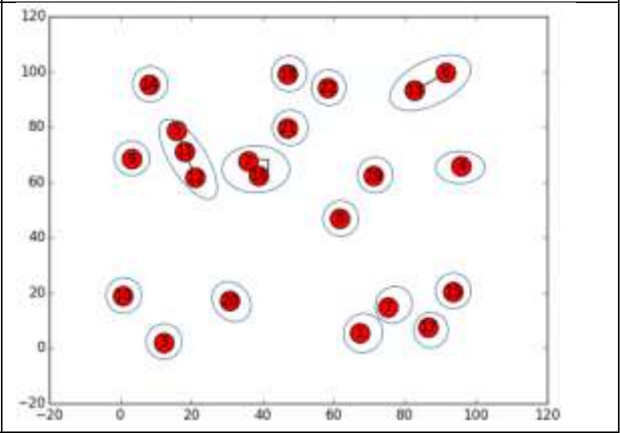

120
100
80
60
40
20
0
−20
−20
0
20
40
60
80
100
120
14
5
11
9
12
2
16
21
18
22
13
6
4
19
8
10
20
3
15
7
1
17

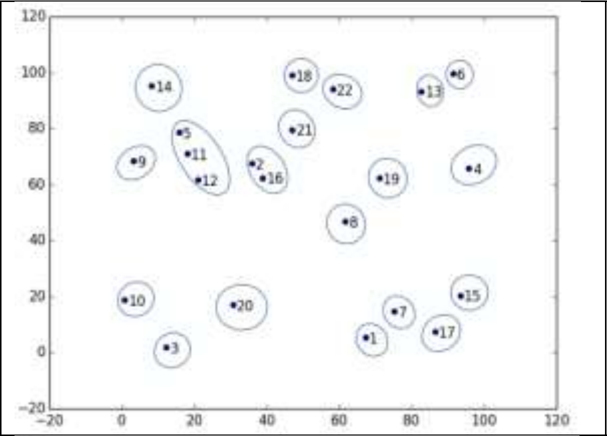
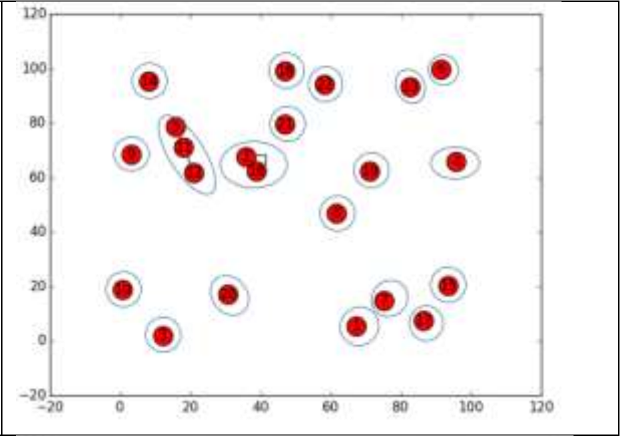
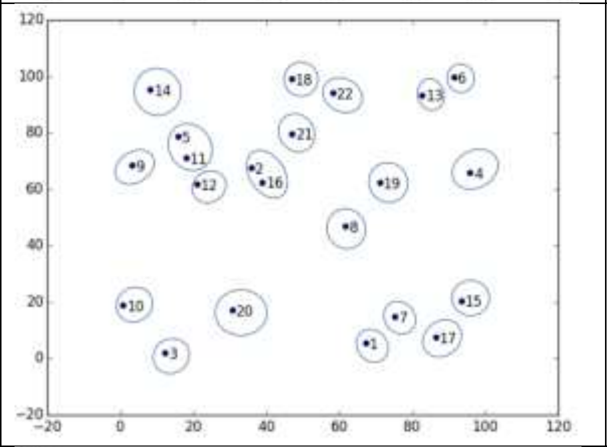
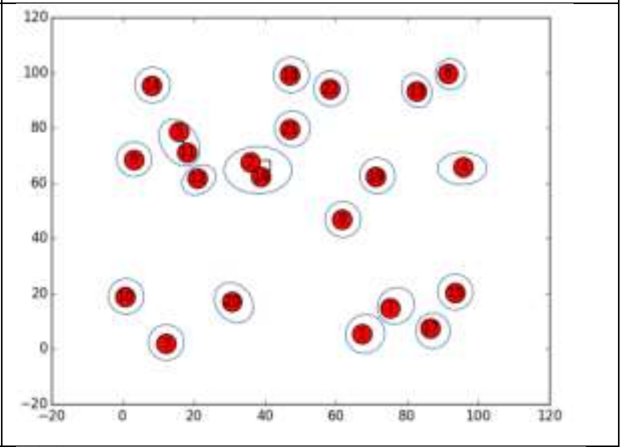
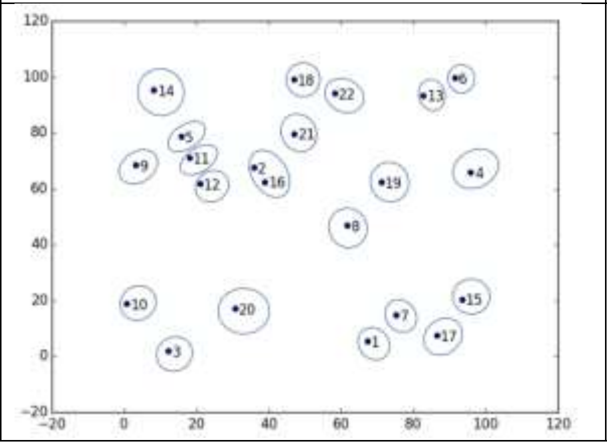
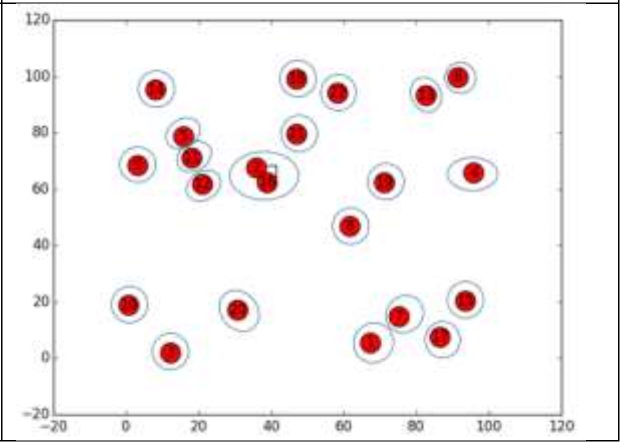


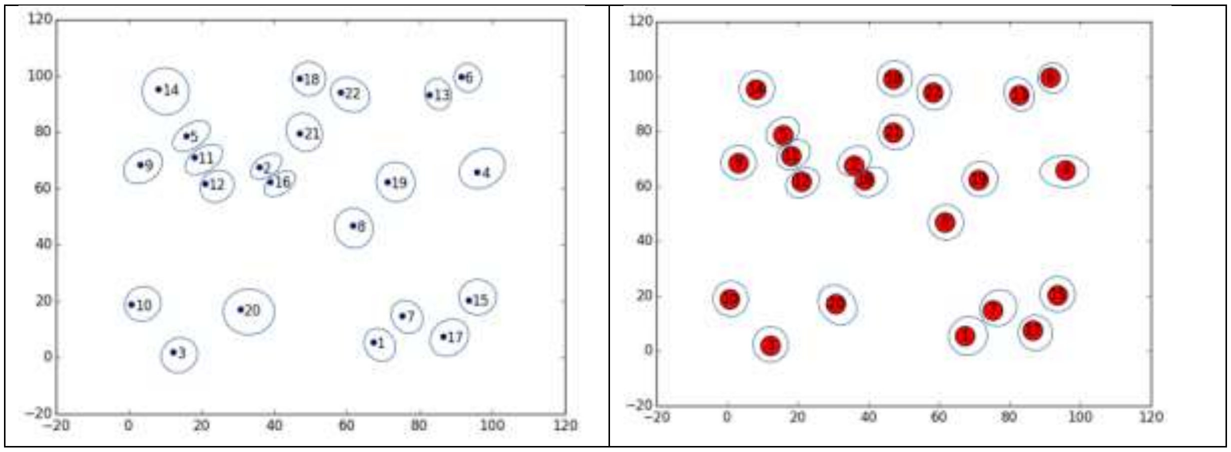

14
18
22
6
13
5
11
9
12
21
2
16
19
8
4
10
3
20
15
7
1
17



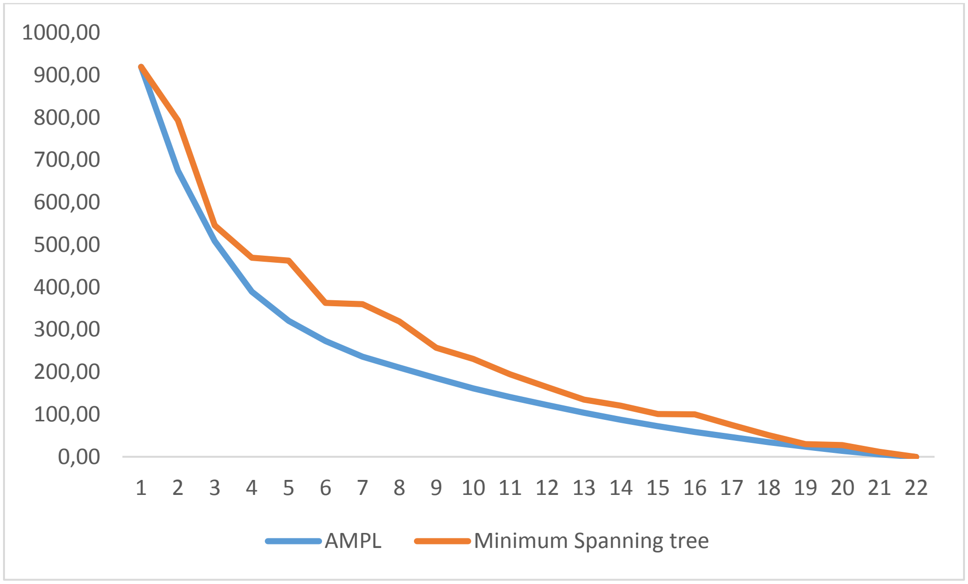
120
100
80
60
40
20
0
−20
−20
0
20
40
60
80
100
120
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22


14
18
22
13
6
5
11
12
9
2
16
21
19
4
8
10
20
15
7
17
1
3


14
18
22
13
6
5
11
12
9
2
16
21
19
4
8
10
20
15
7
17
1
3


14
18
22
13
6
5
11
12
9
2
16
21
19
4
8
10
20
15
7
17
1
3




As it is visible on the results above, the minimization problem and the spanning tree formulation don't always produce the same results. That occurs because the minimization problem returns the clusters with the minimum distance between the points of the cluster and its median, while the spanning tree returns the clusters with the biggest linear separation between them. In other words, the minimization problem returns the clusters with the most similar elements grouped together while the spanning tree returns the clusters with the biggest distance between them. On the Graphic 1 it is possible to compare the sum of the distances between a point and it means on the optimization problem (blue line) and on the spanning tree heuristic (orange line), it means the two different objective functions obtaining using both methods. The Graphic shows that the sum is always minimum on the optimization problem, which shows that the clusters obtained by the optimization problem have their elements more similar (more close) between themselves than the clusters obtained by the spanning tree.



*Graphic 1 - Sum of medians AMPL vs Minimum Spanning Tree*

Analyzing the case k = 2, where AMPL has a different solution from the spanning tree, it is possible to observe that the clusters classified with the spanning tree are more linearly separable than the clusters classified with AMPL. This fact can be observed on Figure 12, below. By the image it is possible to realize that the blue plan has a bigger weight than the orange, in other words, the points separated by the blue plan are more linearly separated than the ones classified by the orange plan. However, when analyzing the Graphic 1 it is possible to realize that the sum of the means of the points separated by the orange plan is smaller than the sum of the means separated by the blue point, which indicates that the clusters generated by the orange plane are more similar between each other than the plans generated by the blue plan.

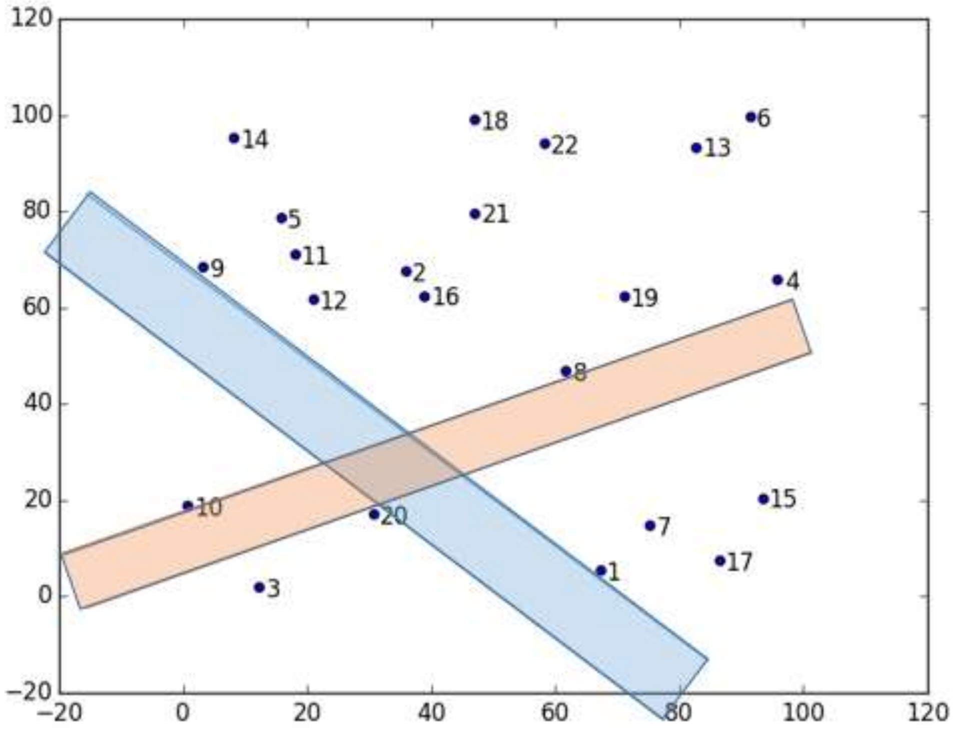
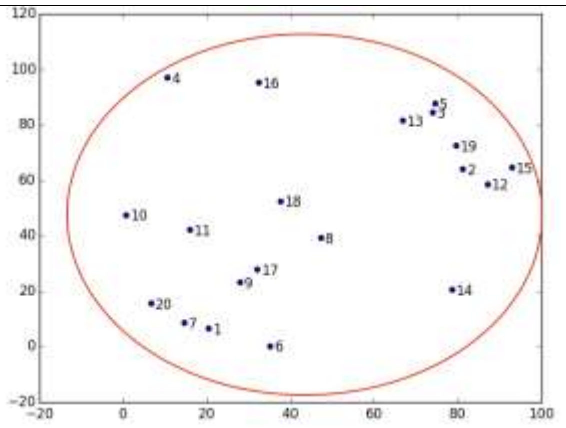
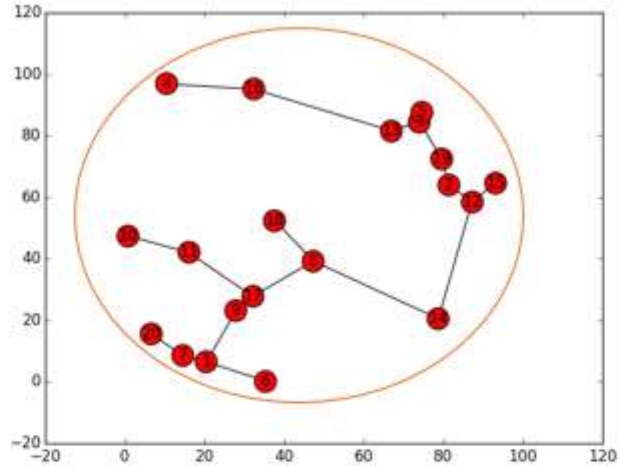
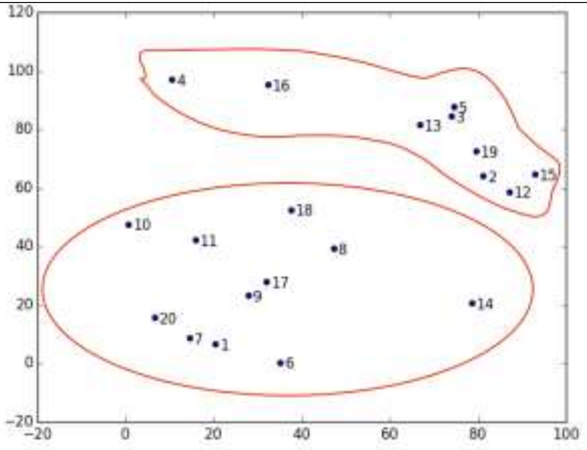
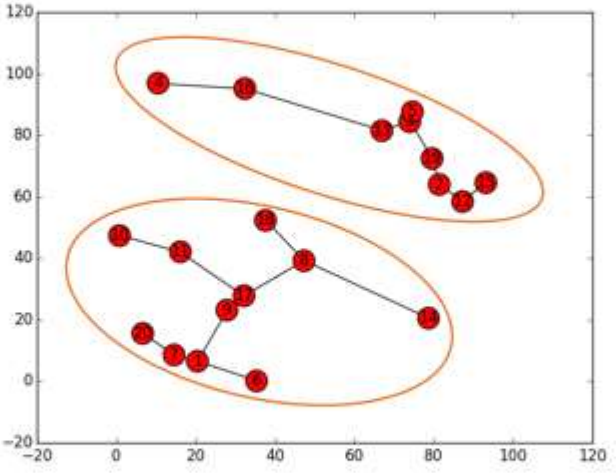
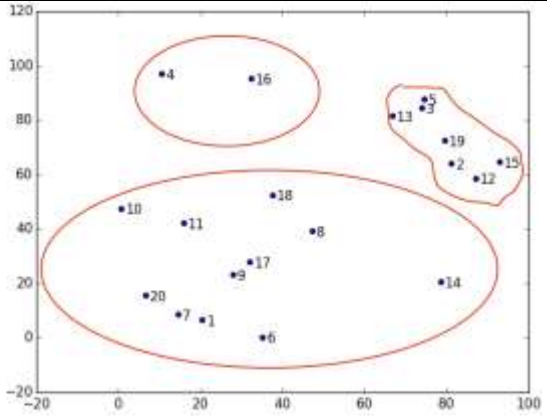
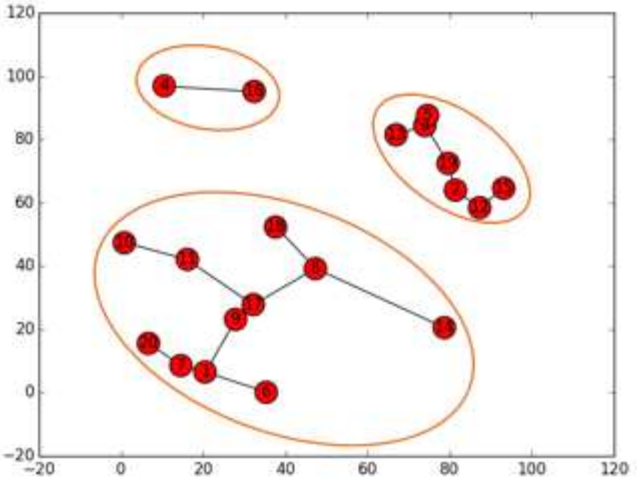


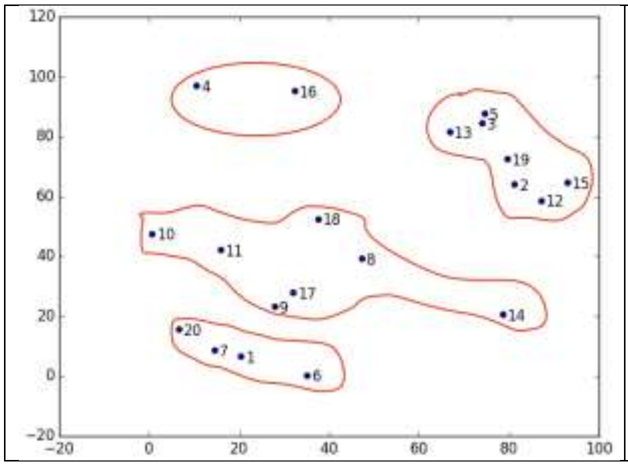
*Figure 12 – linear separation from the data for k = 2*

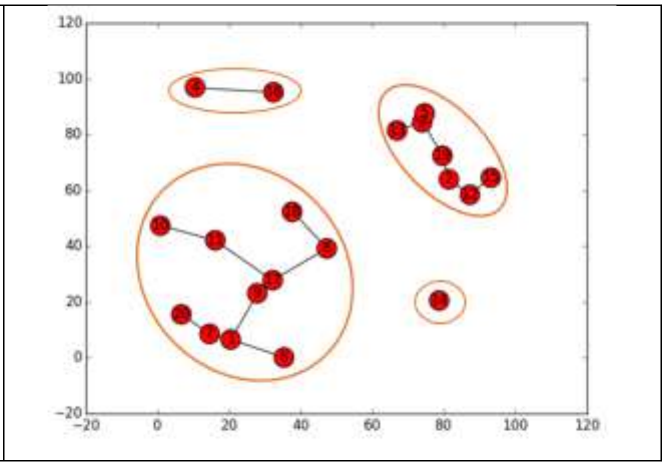
On the other hand, the cluster generated by the spanning tree are more linearly separated than the plans generated by the optimization problem. On a case where the data is linearly separated between the classes, the spanning tree heuristic should return the same values as the optimization problem, being a good approximation for the cluster-median problem.

Let's consider the following randomly data set of 20 points, for making more clear the above approach. The results obtained using both methods are shown in the below table:

| **AMPL: Integer Optimization Problem** | **Minimum Spanning Tree** |
|---|---|


4
16
5
3
13
19
2
15
12
18
10
11
8
17
9
14
20
7
1
6


4
16
5
3
13
19
2
15
12
18
10
11
8
17
9
14
20
7
1
6


4
16
5
3
13
19
2
15
12
18
10
11
8
17
9
14
20
7
1
6






When 2 clusters are built both methods got the same result, the same happens for 3 clusters. Nonetheless, it does not occur when 4 clusters are made. The reason is the same: the minimum spanning tree method is looking for widely separate clusters.

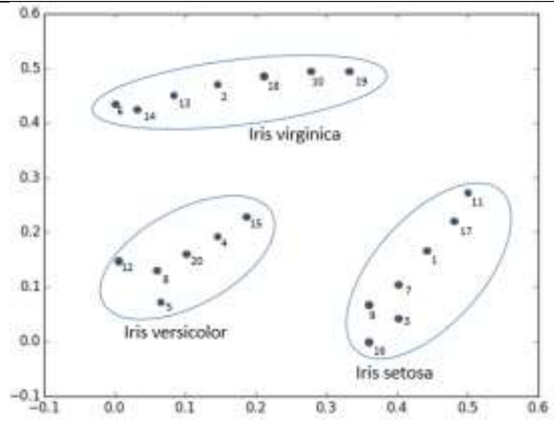
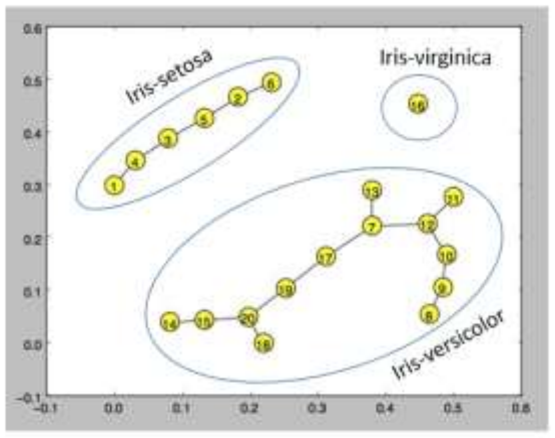
**Iris data set:**

For the Iris data set, only the classification into 3 clusters is considered. It is previously known from the data set that only one class is linearly separable from the other two. This way, it is expected to have a better result with AMPL than with the spanning tree. For this particular application, the expected classification is known before and described below:

- Points 1 – 6: Iris-setosa
- Points 7 – 13: Iris-versicolor
- Points 14 – 20: Iris-virginica

This way, it is expected to obtain 3 clusters, the first containing points 1 to 6, the second containing points 7 to 13 and the last one with the points 14 to 20.

With this previous information, the clusters are generated with AMPL and the Spanning tree and the results can be seen below:

| AMPL: Integer Optimization problem | Minimum spanning Tree |
|---|---|
|  |  |

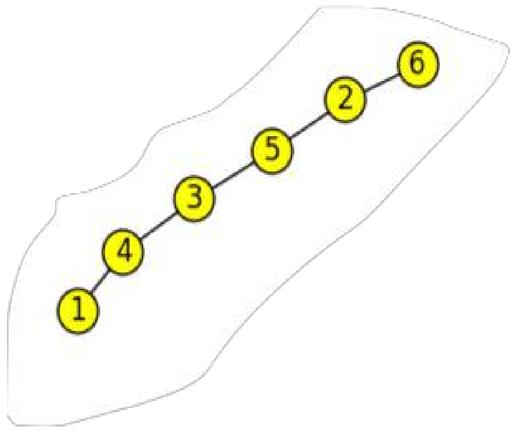
Let's analyze the k median points obtained when the cluster-median problem is formulated as an integer optimization problem and when it is solved by the heuristic solution with minimum spanning tree.

**K-median points – Heuristic solution with minimum spanning tree**

The median points for the spanning tree were obtained using the following equation:

$$\sum_{i\,\epsilon\,\mathcal{I}} d_{i,p} = \min_{j\,\epsilon\,\mathcal{I}} \sum_{i\,\epsilon\,\mathcal{I}} d_{ij}$$

where p represents the median point



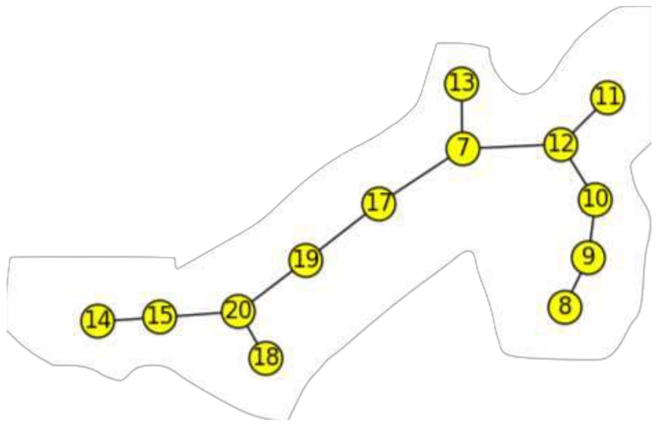
The below table represents the matrix containing all the Euclidean distances for all the points contained in this cluster, and it is clear that the median is the **vertex 3** (point 3). Mathematically, it can be proved as:

$$\sum_{i\,\epsilon\,\mathcal{I}} d_{i,p} = \min_{j\,\epsilon\,\mathcal{I}} \sum_{i\,\epsilon\,\mathcal{I}} d_{ij}$$
$$\sum_{i\,\epsilon\,\mathcal{I}} d_{i,3} = \min_{j\,\epsilon\,\mathcal{I}} \sum_{i\,\epsilon\,\mathcal{I}} d_{ij}$$
$$2.7806 = \min[2.99, 3.1387, 2.7806, 3.1209, 3.151, 4.1903]$$

For comparison reasons, this cluster will be named cluster-3 with a sum of distances of 2.7806.

| Vertices | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0.7874 | 0.3436 | 0.1732 | 0.6557 | 1.04 |
| 2 | 0.7874 | 0 | 0.6782 | 0.8544 | 0.3605 | 0.4582 |
| 3 | 0.3436 | 0.6782 | 0 | 0.3 | 0.6403 | 0.8185 |
| 4 | 0.1732 | 0.8544 | 0.3 | 0 | 0.7071 | 1.0862 |
| 5 | 0.6557 | 0.3605 | 0.6403 | 0.7071 | 0 | 0.7874 |
| 6 | 1.04 | 0.4582 | 0.8185 | 1.0862 | 0.7874 | 0 |
| **Sum of distances** | **2.9999** | **3.1387** | 2.7806 | **3.1209** | **3.151** | **4.1903** |

It is obviously that the distance between a vertex and itself is zero. Then, this cluster will be named cluster-16 with a sum of distances equal to 0.



Finally for this cluster, the below table represents the matrix containing all the Euclidean distances for all the points contained in this cluster, and it is clear that the median is the **vertex 7** (point 7). Mathematically, it can be proved as:

$$\sum_{i\,\epsilon\,\mathcal{I}} d_{i,p} = \min_{j\,\epsilon\,\mathcal{I}} \sum_{i\,\epsilon\,\mathcal{I}} d_{ij}$$

$$\sum_{i\,\epsilon\,\mathcal{I}} d_{i,7} = \min_{j\,\epsilon\,\mathcal{I}} \sum_{i\,\epsilon\,\mathcal{I}} d_{ij}$$

$$9.9015 = \min[9.9015, 16.5395, 15.1532, 11.2563, 11.1974, 10.6902, 10.9291, 19.8994, 16.3257, 11.6093, 16.2225, 11.8628, 15.0248]$$

For comparison reasons, this cluster will be named cluster-7 with a sum of distances of 9.9015

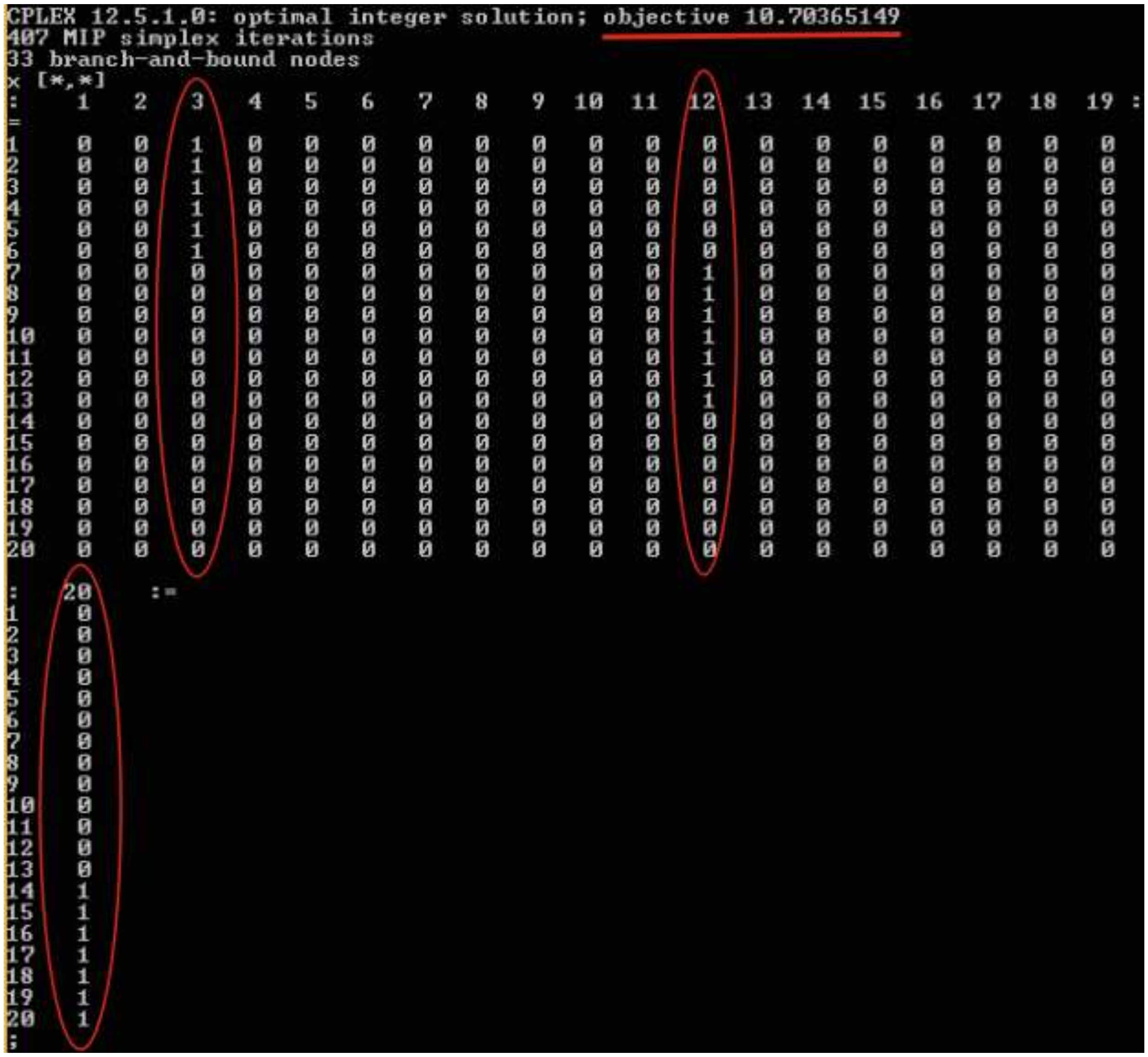
| Vertices | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1.1958 | 1.0723 | 0.5744 | 0.5292 | 0.469 | 0.3464 | 1.7263 | 1.3675 | 0.616 | 1.319 | 0.6856 | 1.1358 |
| 2 | 1.195 | 0 | 0.4242 | 0.7746 | 0.8307 | 0.8185 | 1.0344 | 2.8862 | 2.5259 | 1.7234 | 2.488 | 1.844 | 2.3195 |
| 3 | 1.072 | 0.4242 | 0 | 0.5477 | 0.755 | 0.7 | 1.0149 | 2.7659 | 2.3281 | 1.5588 | 2.3431 | 1.6432 | 2.1633 |
| 4 | 0.574 | 0.7746 | 0.5477 | 0 | 0.3317 | 0.2236 | 0.6403 | 2.2782 | 1.8493 | 1.1091 | 1.8138 | 1.1136 | 1.6553 |
| 5 | 0.529 | 0.8307 | 0.755 | 0.3317 | 0 | 0.1414 | 0.5292 | 2.2226 | 1.8682 | 1.1401 | 1.7493 | 1.1 | 1.5968 |
| 6 | 0.469 | 0.8185 | 0.7 | 0.2236 | 0.1414 | 0 | 0.5099 | 2.1863 | 1.8083 | 1.0677 | 1.7146 | 1.0536 | 1.5716 |
| 7 | 0.346 | 1.0344 | 1.0149 | 0.6403 | 0.5292 | 0.5099 | 0 | 1.8815 | 1.5967 | 0.8124 | 1.5684 | 0.995 | 1.3304 |
| 8 | 1.726 | 2.88 | 2.7659 | 2.2782 | 2.2226 | 2.1836 | 1.8815 | 0 | 0.6557 | 1.2248 | 0.7874 | 1.2845 | 0.7416 |
| 9 | 1.367 | 2.5259 | 2.3281 | 1.8493 | 1.8682 | 1.8083 | 1.5967 | 0.6557 | 0 | 0.8185 | 0.6708 | 0.8367 | 0.5099 |
| 10 | 0.616 | 1.7234 | 1.5588 | 1.1091 | 1.1401 | 1.0677 | 0.8124 | 1.2248 | 0.8185 | 0 | 1 | 0.5385 | 0.728 |
| 11 | 1.319 | 2.488 | 2.3431 | 1.8138 | 1.7493 | 1.7146 | 1.5684 | 0.7874 | 0.6708 | 1 | 0 | 0.7681 | 0.6245 |
| 12 | 0.685 | 1.844 | 1.6432 | 1.1136 | 1.1 | 1.0536 | 0.995 | 1.2845 | 0.8367 | 0.5385 | 0.7681 | 0 | 0.6481 |
| 13 | 1.135 | 2.3195 | 2.1633 | 1.6553 | 1.5968 | 1.5716 | 1.3304 | 0.7416 | 0.5099 | 0.728 | 0.6245 | 0.6481 | 0 |
| **Sum of distances** | 9.9015 | **16.5395** | **15.1532** | **11.2563** | **11.1974** | **10.6902** | **10.9291** | **19.8994** | **16.3257** | **11.6093** | **16.2225** | **11.8628** | **15.0248** |

Finally, adding up the sum of distances of all the clusters, the objective function obtained using the heuristic solution with minimum spanning tree is:

| Cluster | Sum of distances |
|---|---|
| Cluster-3 | 2.7806 |
| Cluster-16 | 0 |
| Cluster-7 | 9.9015 |
| Objective Function | 12.6821 |

**K-median points – Integer Optimization problem.**

Formulating the cluster-median problem as an Integer Optimization Problem, and solving it using AMPL, the following clusters were found:



```
CPLEX 12.5.1.0: optimal integer solution; objective 10.70365149
407 MIP simplex iterations
33 branch-and-bound nodes
x [*,*]
:    1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  :
=
1    0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2    0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
3    0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
4    0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
5    0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
6    0   0   1   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   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   1   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

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

Using AMPL, the medians obtained were the points 3, 12 and 20, whose form the clusters: cluster-3, cluster-12 and cluster-20, respectively.

| **Cluster** | **Sum of distances** |
|---|---|
| Cluster-3 | 2.7806 |
| Cluster-12 | 2.8625 |
| Cluster-20 | 5.0577 |
| Objective Function | 10.7036 |

Taking into account that the main goal of the cluster-median problem is minimize the sum of all the distances from each point to its nearest k-median point, and considering only the objective functions obtained by both methods, it is simple to notice that the solution obtained using the integer optimization problem is smaller than the one obtained using the heuristic solution with MST, thus better.

Since, the expected results are known in advance, then this solution becomes more intuitive, and it can be easily confirmed by comparing the clusters obtained with both solutions and the expected.

It is important to notice, that for both solutions the cluster-3 contains the same points and has exactly the same sum of distances, nevertheless the other two clusters are not the same. This is easy to explain, because the fact that one class was linearly separable from the other two was known in advance.

## 5. Final observations and comments:

Summing up, in this assignment two different data sets and two different methods were used to solve the cluster-median problem, yielding the following observations and comments:

According to the results obtained, we deduced that the heuristic solution using minimum spanning tree provides an acceptable solution if the classes are linearly separable, this conclusion can be better seen when real and meaningful data was used instead of randomly generated data.

Analyzing the real data-set case, it is possible to conclude that, for no linearly separated data, the results obtained with the cluster-median problem implemented as an integer optimization problem (solved by AMPL) are usually better, however, the computation of an integer optimization problem is more difficult (NP-hard) than the computation of the spanning tree ($O(k * log(m))$). It is also known that the integer optimization problem has a limitation of data, only working well for classifying a small amount of data, having $O(m^2)$ interactions for classifying m points.

This way, there is no absolutely best solution for all problems. The best solution will depend on the data analyzed, on the linearity separation of this data and on the computational resources available. However, nowadays, there exist a wide range of methods affording better solutions for the cluster-median problem.