Cluster-Median Problem

Optimization in Data Science

Jordi Bosch Bosch
Marc Esquerrà Corominas

November 1, 2020

Content

1 Introduction

Clustering is one of the most famous methods in unsupervised learning. It consists on analysing data and dividing it in such way that the objects included in each group share similar properties.

This method has proved to be very useful in lots of circumstances, in which one wants to cluster a huge amount of data, which classification is not obvious. Some examples are the classification of websites, novelty or, like in our case, the classification of football players by their position on the field given their FIFA stats.

For clustering we need to define the notion of distance between different objects. If our observations are characterized by $n \in \mathbb{N}$ numerical attributes, our data will live in $\mathbb{R}^n$. Hence, the distance between two objects of our data set will be any distance defined over $\mathbb{R}^n$ (usually the Euclidean distance).

The main goal of this assignment is to formulate and solve the cluster-median problem to classify a real set of data. It is well known that the k-medoid is the optimal algorithm of solving the cluster-median problem, however it is computationally very expensive. Therefore, another objective of the assignment is to compare the k-medoid with two heuristic methods that approximate it, the k-means and the minimum spanning tree (MST) heuristic. Finally, we will check how the computation time of each method varies when increasing the size of the data set.

2 Our Data

To apply the clustering algorithms to a real data set, we want to group football players by their position on the field of football players given their FIFA 20 stats. We will consider the top 500 players according their overall rating on the EA's game. Each player will have associated 34 attributes, that include pacing, shooting, passing, dribbling, defending, physical and goalkeeping skills and go from 1 to 99. To scale the data we will divide each attribute of the players by their corresponding overall score. This way, the value of an attributes will indicate if a player stands out or not in that attribute. Our intention is to divide the players in 4 groups: goalkeepers, defensive area, midfield and attacking area.

We also count with the players' position in the game. After obtaining the clusters for each method we will be able to check if the players in a group have positions that are in a similar zone on the field. The players of our data set have the following 15 different positions, which theoretical position on the field is shown in Figure 1:

- Goalkeeper, **GK** (70).
- Center back, **CB** (83): defender that plays in the middle of the defense.
- Right back, **RB** (16): defender that plays in the right of the defense.
- Left back, **LB** (19): defender that plays in the left of the defense.
- Right wing back, **RWB** (2): similar to a RB with a higher offensive projection.
- Left wing back, **LWB** (1): similar to a LB with a higher offensive projection.
- Center defensive midfielder, **CDM** (39): midfielder that usually helps on defensive tasks and associates with the defenders to begin attacking chance.
- Center midfielder, **CM** (66): plays in the middle of the field.
- Right midfielder, **RM** (23): plays in the right of the middle of the field. Its offensive projection depends on the team tactics.
- Left midfielder, **LM** (13): plays in the left of the middle of the field. Its offensive projection depends on the team tactics.

- Center attacking midfielder, **CAM** (45): most advanced midfielder. Usually associates with the attackers to generate goal chances. Its the bridge between the midfield and the attacking area.
- Right wing, **RW** (19): plays in the right of the attacking area.
- Left wing, **LW** (17): plays in the left of the attacking area.
- Center forward, **CF** (19): the attacking reference the team or one of them. Plays in the middle of the attacking area and has an associative playing style.
- Striker, **ST** (68): the attacking reference of the team. Similar to a CF, but more focused on scoring.

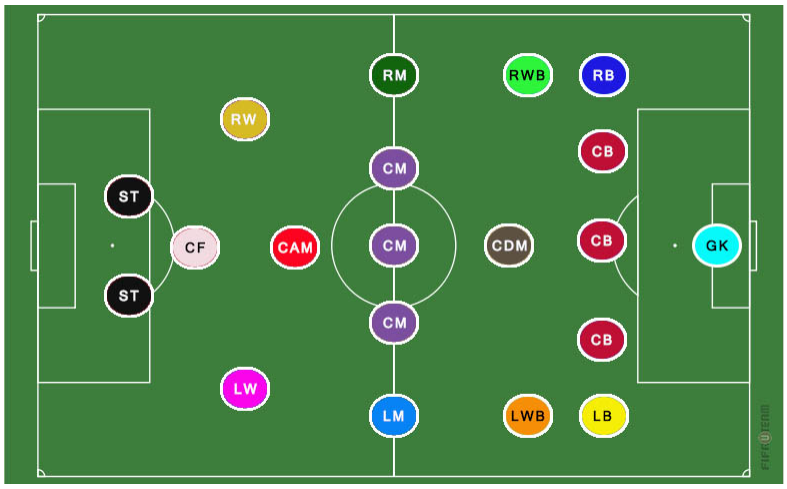

Figure 1: Theoretical positions on the field for the different football players. The team scores in the left goal. Source: https://www.fifauteam.com/fifa-ultimate-team-positions-and-tactics/.

3 k-medoid

We want to solve the cluster-median problem. This problem consists on grouping a set of data in k different clusters while minimizing the distance from the median of each cluster to the rest of points in that same cluster.

Let us consider that we have a data set A with $m \in \mathbb{N}$ observations and each one of them has $n \in \mathbb{N}$ numerical attributes associated. Therefore, $A \subset \mathbb{R}^n$. We want to divide the data set into k clusters, $A = \bigsqcup_{i=1}^{k} S_i$. Each cluster has a median $r_i \in S_i$, $i \in \{1, \dots, k\}$.

Mathematically, the point r is the median of the set $\mathcal{J} \subset \mathbb{R}^n$ if

$$\sum_{i \in \mathcal{J}} d_{ir} = \min_{j \in \mathcal{J}} \sum_{i \in \mathcal{J}} d_{ij},$$

where d_{ij} denotes the Euclidean distance between the elements i and j.

The goal of the cluster-median problem is to minimize the sum of all the sums for each cluster of the distances between its median and the rest of the points that belong to that cluster. We can formulate the problem as a linear programming optimization. To do it, we will make the following two consideration:

- There are m clusters, $(m - k)$ of which are empty.
- If the j-th cluster is not the void, then the element j belongs to it and it is its median.

After making these considerations, the formulation of the problem as a linear programming problem is the following:

$$\textbf{min} \quad \sum_{i=1}^{m}\sum_{j=1}^{m} d_{ij}x_{ij}$$

subject to

$$\sum_{j=1}^{m} x_{ij} = 1, \qquad \forall i \in \{1,\dots,m\} \tag{1}$$

$$\sum_{j=1}^{m} x_{jj} = k \tag{2}$$

$$x_{jj} - x_{ij} \geq 0 \qquad \forall i,j \in \{1,\dots,m\} \tag{3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in \{1,\dots,m\}$$

Where the problem variables are defined as

$$x_{ij} = \begin{cases} 1 & \text{if element } i \text{ belongs to cluster } j, \\ 0 & \text{otherwise} \end{cases}$$

and equation (1) imposes that each element belongs to one cluster, equation (2) that we have exactly k clusters and equation (3) that an element can belong to the j-cluster only if this cluster exists (cluster j exists if and only if element j belongs to it, i.e. $x_{jj} = 1$).

3.1 Results

Since we count with a considerable amount of players, to make more understandable the obtained results, we will present them with a table per cluster with the number of players for each position that that cluster contains, instead of a vector of size m where the i-th position has the value of the cluster that the i-th player belongs to.

We solved the previous formulated problem with AMPL. The value of the objective is 311.83 and the resulting clusters are the following:

- **Cluster 1**:

GK
70

- **Cluster 2**

CB	CDM	LB	LWB	RB	RWB
81	14	3	1	6	2

- **Cluster 3**:

CAM	CB	CDM	CF	CM	LB	LM	LW	RB	RM	RW	ST
9	2	25	1	66	16	2	1	10	5	1	2

- **Cluster 4**:

CAM	CF	LM	LW	RM	RW	ST
36	18	11	16	18	18	66

3.2 Execution time

We want to see how the k-medoid algorithm perform in terms of computation time when the size of the data set increases. We have selected nine different data set sizes from 100 up to 10000. The amount of execution time taken by the AMPL solver for each data set size is represented in the following table:

m	100	200	300	400	500	750	1000	5000	10000
CPU time (s)	0.484	3.782	13.172	33.688	92.25	-	-	-	-

Table 1: Execution time of the k-medoid algorithm by size of the data set.

There are missing values in Table 1 due to the fact that our computer has not enough computation power to solve the problem for those sizes of the data set and the AMPL solver breaks.

We can see in Figure 2 that the execution time of the k-medoid algorithm as a function of the data set's size has a parabolic behaviour, which is coherent with the theory, since the complexity of this algorithm is $\mathcal{O}(m^2k^2)$.

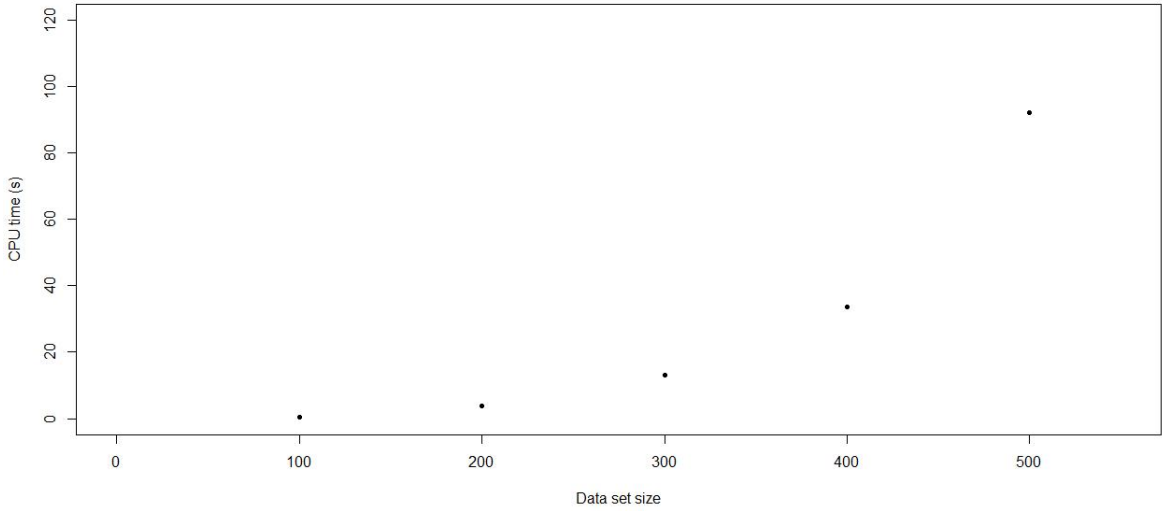

Figure 2: Execution time of the k-medoid algorithm as a function of the data set's size.

4 Heuristics

Looking for the exact solution can be very time-consuming, specially when dealing with large values of points. For this reason, sometimes it is very useful to consider heuristic algorithms to approximate the solution of our original problem. Since finding the linear programming solution is very time consuming for large values of points, we decided to search for some heuristics to find faster solutions with the trade-off of loosing some accuracy. Those heuristics are necessary when we scale our problem to large values of points. We will compare them to our optimal solution and see how was their performance in accuracy and in computational time.

4.1 k-means

k-means is a famous heuristic when dealing with clustering. The main difference with k-medoid is that we will use the approximation of the median of a subset of points using the mean.

This is computationally expensive, since finding the median at each iteration for each cluster is $\mathcal{O}(n^2)$. (We have to loop over the m points and compute distances with the $(m-1)$ other points, for each cluster). In the case of the k-means we can eliminate this tedious computation, since we only need to loop over the m points of each cluster to compute the mean

$$\frac{1}{n}\sum_{i=1}^{n} X_i = \mathcal{O}(n).$$

4.1.1 The algorithm

1. Start with k random points (which we will call centers of the clusters)
2. Assign each point of our set into the cluster center that is more close to it.
3. Right now, we have that each element is assigned into one cluster. We update the cluster center using the approximation of the median of a subset by the mean of all the points.
4. Repeat Steps 2 and Step 3 until the process stabilizes and there's no changes in the clusters.

4.1.2 Advantages

- Gives us a very fast way to cluster data.
- Can be applied to any kind of clusters with different shapes
- Scales nicely to large amount of points

4.1.3 Disadvantages

- Highly dependent on the initial chosen values of centers. This can be mitigated performing multiple k-means initialization for small values of points. When the number is large, it would be computationally expensive.
- The mean of the clusters ("our clusters center representation") probably won't be any of our points.

4.1.4 PCA representation

Our data lives in $\mathbb{R}^{34}$. This makes it impossible for us to represent it as a 2D graph. What we've done is use PCA (Principal Component Analysis) for data dimensions reductions to represent our data in terms of two variables, that are the ones that mostly explain the variability of the data.

In Figure 3, we can see the 4 clusters generated by the k-means algorithm. One of them is really far from the others. We can hypothesize that this cluster will majorly contain goalkeepers. Their distances with the outfield players is quite big since they have really specific and differentiated statistics. The other 3 clusters have some shared region when represented with PCA. We can get some insight that, in some cases, it will be hard to distinguish a defensive midfielder from a defender and same applies to an attacking midfielder from an attacker.

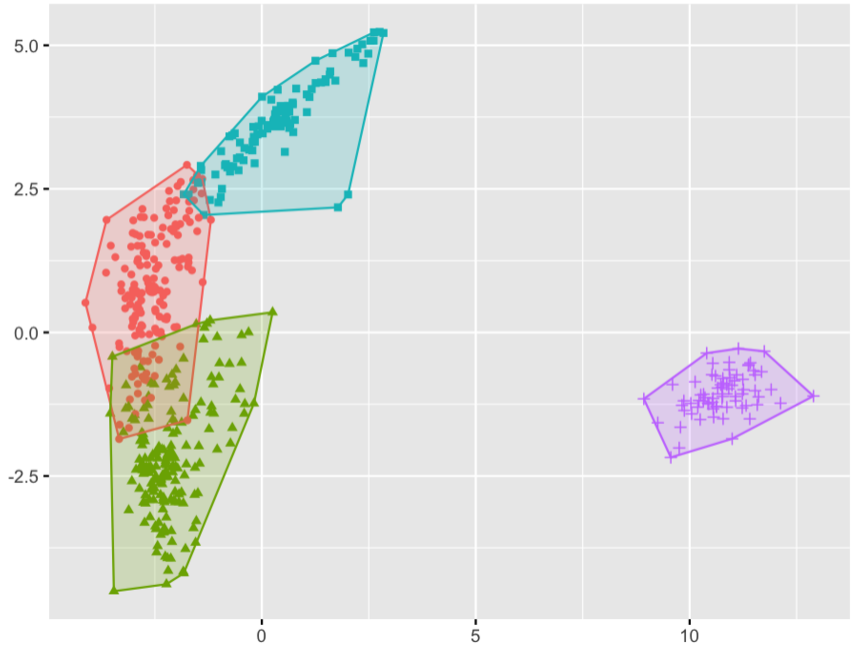

Figure 3: Representation of the clusters obtained by the k-means algorithm in the space generated by the two principal componetns.

4.1.5 Results

Let's see the exact clusters distribution k-means has given us:

- **Cluster 1**:

GK
70

- **Cluster 2**

CB	CDM	RB
80	11	2

- **Cluster 3**:

CAM	CB	CDM	CF	CM	LB	LM	LW	LWB	RB	RM	RW	RWB	ST
9	3	28	1	66	19	4	2	1	14	7	1	2	1

- **Cluster 4**:

CAM	CF	LM	LW	RM	RW	ST
36	18	9	15	16	18	67

To compare this heuristic with the k-medoid, we have calculated the median of each one of the resulting clusters after applying the k-means. Then, we have added for each cluster the sums of the distances from its median to the rest of the points of the cluster to obtain the value of the objective function if this was the result obtained with the k-medoid algorithm. 313.76 is the obtained value. Thus, the relative error of the k-means algorithm with respect to the k-medoid algorithm is:

$$\varepsilon_{k\text{-means}} = \frac{|311.83 - 313.76|}{311.83} 100\% = 0.619\%,$$

which is a very small value.

We will also compare both algorithms by checking the percentage of matching that they present. To calculate the matching proportion, we simply have to count how many players are classified in the same cluster in both cases and divide by m. Hence, the percentage of matching between the k-means and the k-medoids is:

$$\text{MP}_{k\text{-means}} = \frac{474}{500}\,100\% = 94.8\%,$$

which is a very high matching percentage.

4.1.6 Execution time

We want to see how the k-means performs when we increase the number of points.

m	100	200	300	400	500	750	1000	5000	10000
CPU time (s)	0.194	0.706	0.197	0.209	0.209	0.319	0.289	0.768	1.449

Table 2: Execution time of the k-means algorithm by size of the data set.

We can clearly see in Table 2 how the CPU time depends linearly in m, which means it gives us good time bounds for scaling our problem.

One may wonder why the CPU times are not strictly increasing when we increase the amount of data that we are working with. This may be due to "cache" improvements and reusing already compiled code, which for small values of m can spend more time than executing the algorithm itself. We can see that for larger values of m, this pattern breaks and the execution time seems to increase linearly with m, which is expected as the complexity of the k-means algorithm is $\mathcal{O}(m)$.

4.2 Minimum Spanning Tree

This second heuristic we used is called the minimum spanning tree heuristic. Given the complete graph, where each edge is weighted with the distance between two objects, we are going to compute the minimum spanning tree of this graph. The MST of a connected graph, is obtained by picking a subset of its edges, such that the sum of their weights is minimum, all points are connected and there are no cycles.

Once we have the minimum spanning tree, we will cut the k-1 more weighted edges, disconnecting the tree in k components. These components will be clusters given by this method. The intuition behind this clustering techniques is that, cutting the more weighted edges we are disconnecting the regions that are more "distant", getting connected components where the objects are "close" to each other.

4.2.1 Results

If we compute the MST and then we delete the 3 heaviest edges we obtain a cluster with 428 players, another one with 70 players and two clusters that only contain one player. If we explore the cluster containing 70 players, We will find out that they are the 70 goalkeepers. This makes sense since the heaviest edge of the MST was the one joining the goalkeepers with the rest of the graph. The three remaining clusters have been obtained from deleting the two heaviest edges of the outfield players' component.

If we take another look to the distribution of our data represented in Figure 3, we can observe that we can clearly distinguish two clouds of points: the goalkeepers and outfield players. The second one cannot be trivially subdivided in three clusters. For this reason, we have concluded that the MST algorithm can be very useful for clustering, when these are easily identifiable. However, in the case that they aren't, MST doesn't perform well due to the fact that the more weighted edges will not be in the middle of the cloud but in its hull, and, therefore, cluster with a single point will be created.

Again, we will compare this heuristic with the k-medoid algorithm with the relative error and the percentage of matching:

$$\varepsilon_{\text{MST}} = \frac{|311.83 - 439.90|}{311.83}\,100\% = 40.78\%, \qquad \text{MP}_{\text{MST}} = \frac{254}{500}\,100\% = 49.20\%.$$

Neither the relative error nor the matching percentage are compelling, as one could expect after seeing the clusters that the heuristic method has formed.

4.2.2 Execution time

We can see in Table 3 it grows like $\mathcal{O}(m^2)$. That's why we can easily handle small cases (lower than 1000), but from that threshold it becomes really time consuming. This is due to the computation of the distance matrix $\mathcal{O}(m^2)$.

m	100	200	300	400	500	750	1000	5000	10000
CPU time (s)	0.039	0.19	0.36	0.776	0.837	1.95	3.36	159	-

Table 3: Execution time of the MST heuristic algorithm by size of the data set.

5 Analysis of the results

We have solved the cluster-median problem with the k-medoid algorithm and two heuristic algorithms and, then, compared their solutions and execution times. Now, we want to see what is the distribution of the clusters obtained by the k-medoid and analyze with the real data why the clusters are the way they are.

The following tables show the proportion of the total number of player that share the same position that are contained in each cluster:

- **Cluster 1 (Goalkeepers)**:

GK
1

- **Cluster 2 (Defenders)**

CB	CDM	RB
0.9638	0.282	0.125

- **Cluster 3 (Midfielders)**:

CAM	CB	CDM	CF	CM	LB	LM	LW	LWB	RB	RM	RW	RWB	ST
0.20	0.036	0.717	0.052	1.00	1.00	0.307	0.117	1.00	0.875	0.304	0.05	1.00	0.014

- **Cluster 4 (Attackers)**:

CAM	CF	LM	LW	RM	RW	ST
0.80	0.947	0.69	0.88	0.695	0.94	0.98

We can observe that our clusters separate the data nicely in the 4 lines of the football soccer field (Goalkeepers, Defensive area, Midfield and Attacking area) as we wanted.

As expected, the goalkeepers were easy to cluster, achieving a 100% of accuracy in our models. Their

statistics are really differentiated from the other and makes it easy to find them. The hardest players to cluster were the outfield ones. We can see that at least a 90% of them were classified in the expected group, for all the team positions, except for 4 subgroups. These happened because these players are the transition between field zones. We proceed to analyze each subgroup:

- **RB/RWB/LB/LWB** :
  These players are considered defenders, nevertheless they have been classified as midfielders. This probably happened because this players are faster and have more offensive projection and associative skills than the rest of the defenders.
- **CDM** : A considerable proportion of this players has been classified in the defensive area. This is not surprising, since in the tactics where there is only one CDM, this player is usually positioned between the CBs with the goal of bringing the ball to the offensive area.
- **RM/LM** : These players have been classified between the midfield and the attacking area. RMs and LMs can have a different function depending on the team tactics. Some times they have a similar role as the CMs, but they tend to play in a side of the field. In the other case, they could be considered as wings.
- **CAM** : The majority of these players have been classified in the attacking area instead of the midfield. This happened due to the fact that CAMs have similar attributes to strikers (scoring skills and faster then the rest of midfielders) and play right behind the attacking reference of the team.

To have a better insight of how the players in each position have been distributed, we can take a look at the barplots in Figure 4.

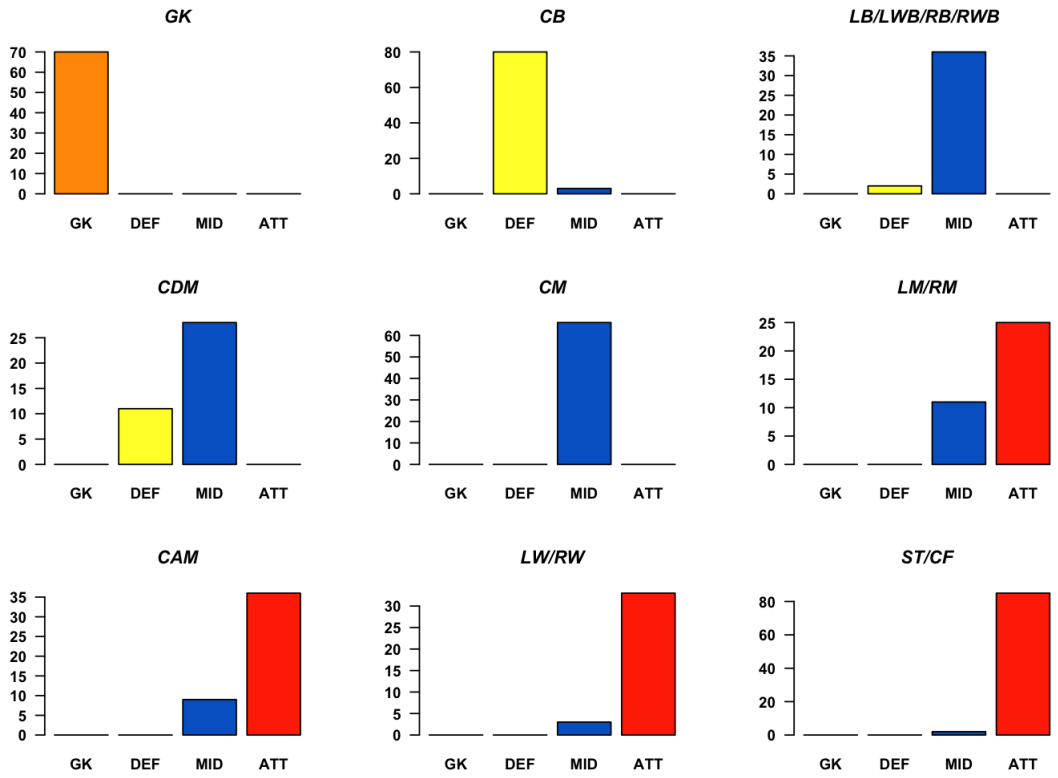

Figure 4: Distribution of the players in the clusters by their position.

Finally, we can see in Figure 5, how would be distributed the players on the field according to our clusters.

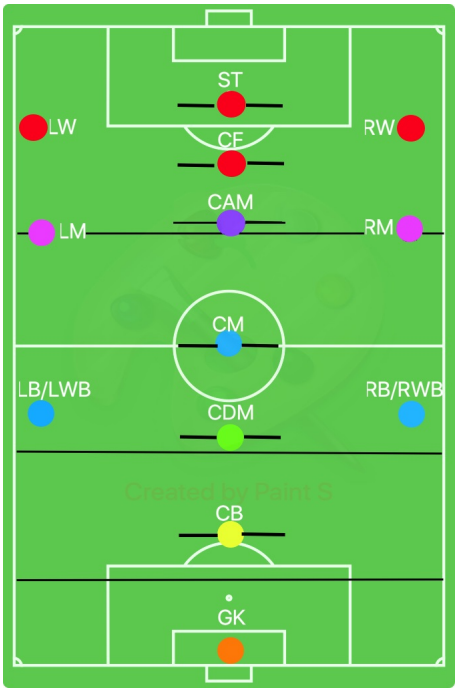

Figure 5: Players' distribution on the field by position according to the obtained clusters.

6 Conclusions

Hereby, we present a summary of our results:

	Objective function	ε of Objective Function	Matching %	CPU time ($f(m)$)
k-medoid	311.83	-	-	Parabolic
k-means	313.76	0.619	94.8	Linear
MST	439.90	40.78	49.20	Parabolic

Table 4: Comparison of heuristics to k-medoid.

We tested different algorithms and heuristics for clustering. We can see that k-means approximates really good the results of k-medoids giving us very similar clusters and close objective functions.

This way, k-means guarantees us a way to scale our problem for large values of points not loosing us much accuracy and being really fast. It takes way less computational time than $k-$medoids. That's why we can consider it a really efficient and trusty alternative.

On the other hand, the MST approach wasn't as good as we thought before. It is clear, the algorithm faces troubles when cutting the heavier edges, misunderstanding random separated points with edges separating clusters.