**CS 590 – PARALLEL AND DISTRIBUTED COMPUTING**

**HOMEWORK 7**

**Name:** Ivan Sangines Escrig

**ID#:** 968606

**Instructor:** Dr. Mahmood

**Date:** December 15, 2018

**TABLE OF CONTENT**

[**INTRODUCTION**](#_Toc446970371) 3

[**SCREENSHOTS:**](#_Toc446970373) 4

[**CONCLUSION:**](#_Toc446970375) 6

**INTRODUCTION**

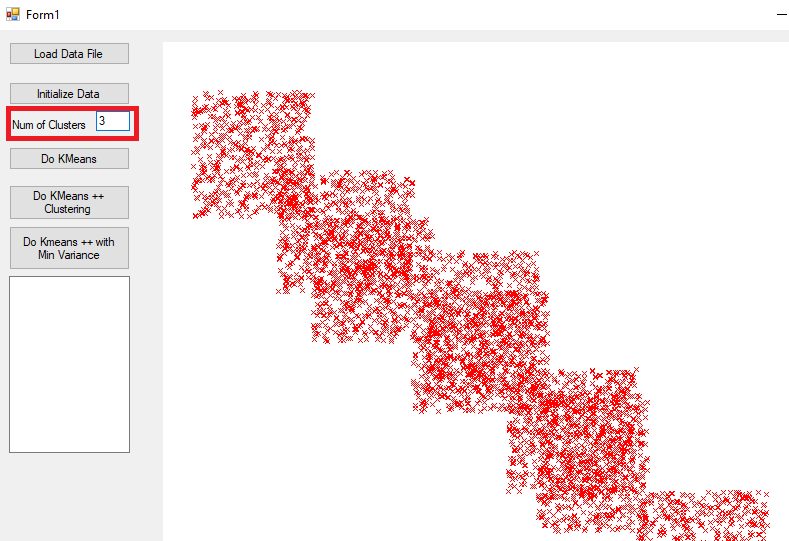
The porpuse of this assignment is to get to learn how Kmeans, Kmeans++, GMM and Kmedoids work. The secon part of this assignment cosists on parallelizing all these algorithm in order to improve their response time. Also, we are asked to apply GMM algorithm in order to divide an image in different clusters

**SCREEN SHOTS:**

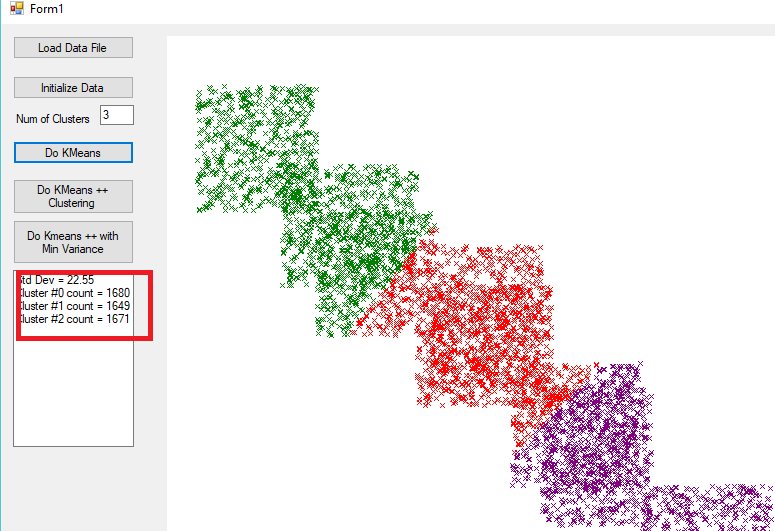
The first part of this assignment was to understand and cleate one project for each algorithm:

Kmeans, this algorithm consist in randomly initializing the centers of the clusters for a further clustering of the different points. Then, we will loop through it and repeat the same process until it converges or we complete all the iterations.

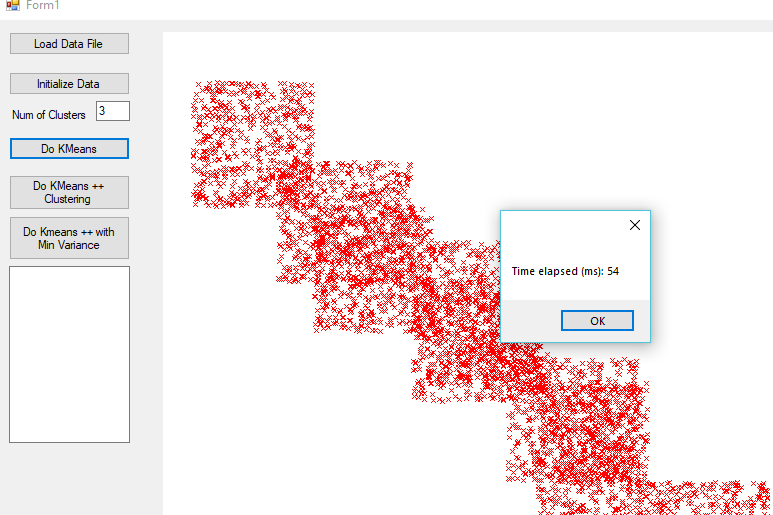
Data Initialized and we will cluster it in three different clusters as shown in the screenshot:

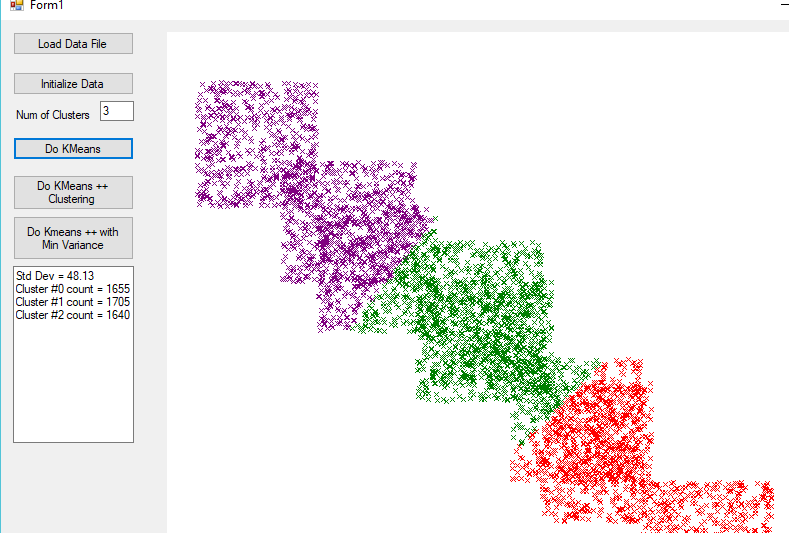


After applying Kmeans, data is divided in three different clusters and the means of each cluster is shown in the read box:



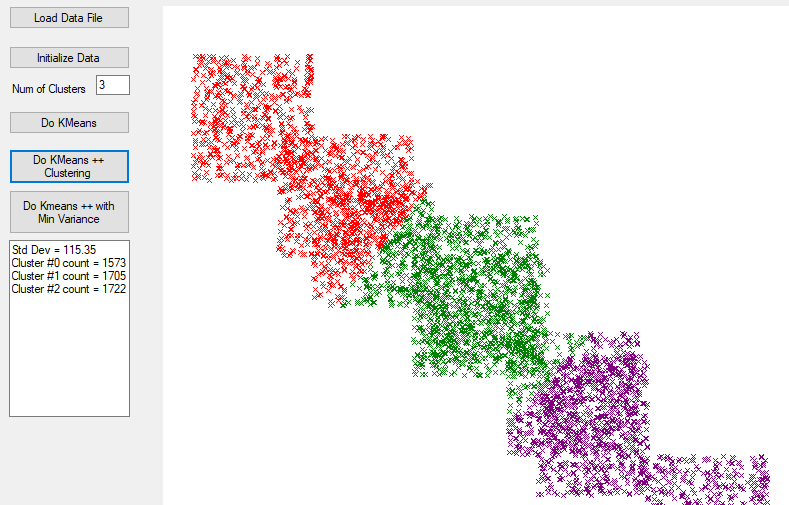
Parallelized Kmeans, same output but faster time of response:



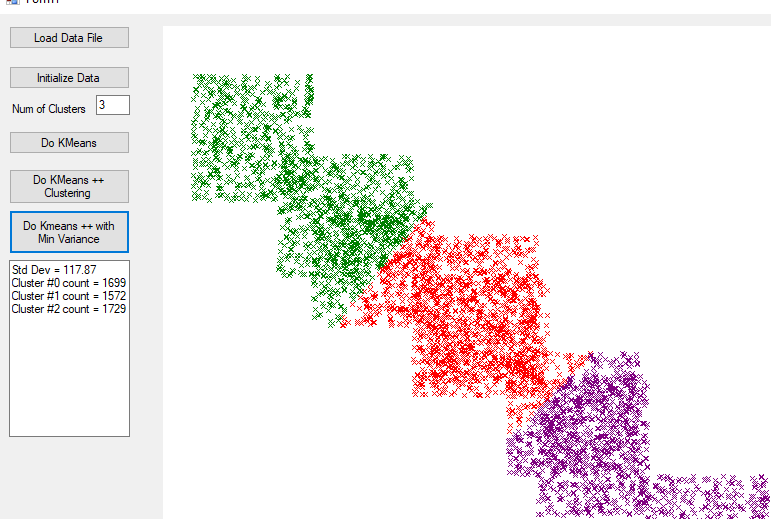


Kmeans++, this algorithm if pretty similar to the Kmeans however, the centers of the clusters are not initialized randomly. The first center is randomly picked and the rest of them are initialized based on their distance from the first cluster. After initialization the algorithm is the same as the Kmeans.

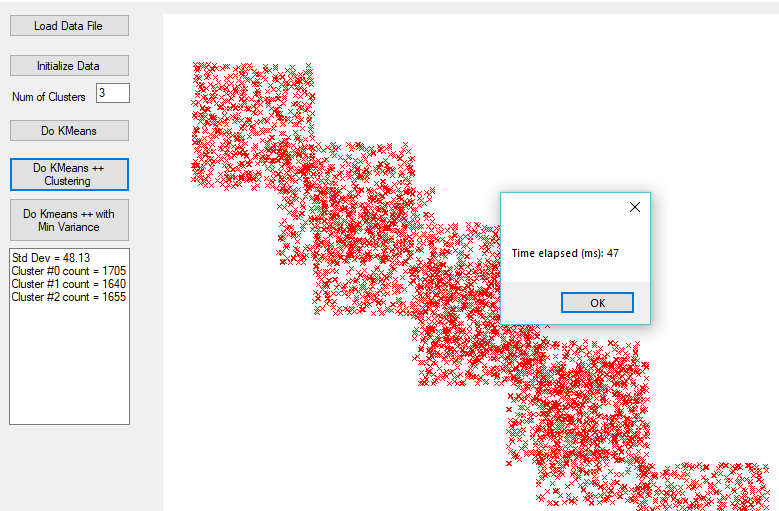
Kmeans++ using same data and clusters as for Kmeans, the clustering now is more acurate since the center of the clusters are not initialized randomly:



Kmeans++ but this time the best result will be the one with lower variance:

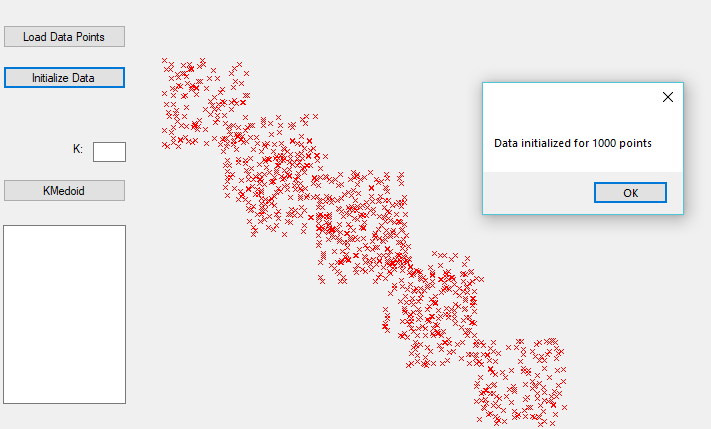


Kmeans++ parallelized:

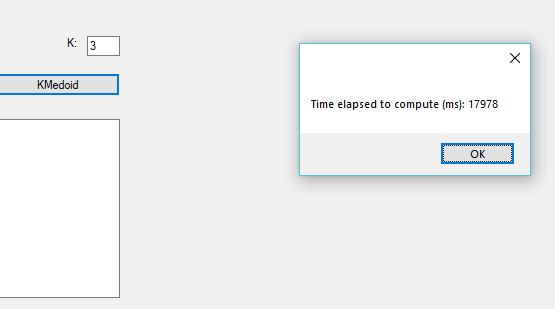


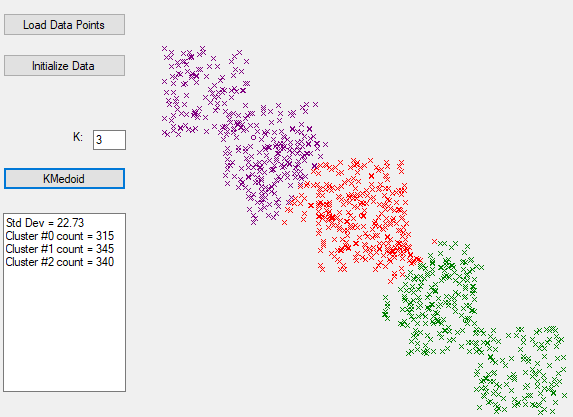
Kmedoids, this algorithm start with X number of centers of data and does the clustering for the different points. Then we will rotate one of the centers and based on metrics like siluette or inertia we will decide if the new clustering is better than the previous one. This step will be repeated untill it converges.

Appliying serialized Kmedoids on a 1000 data points:

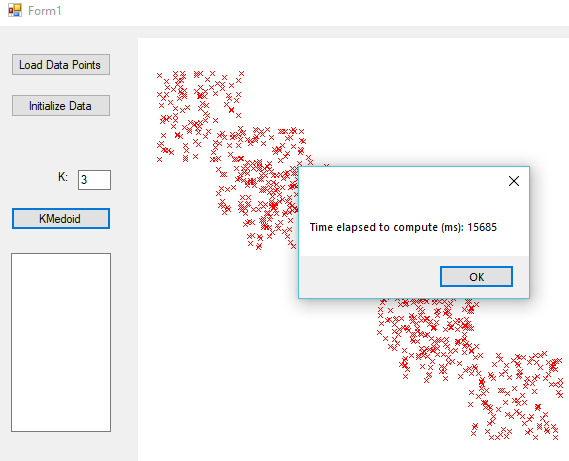


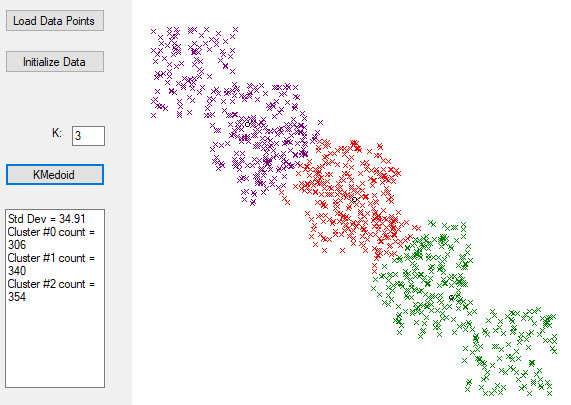
Result of the Kmedoids using 3 clusters:





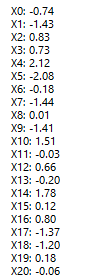
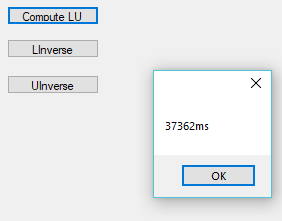
Parallelized Kmedoids in a 1000 data points, using 3 clusters, we can appreciate an improvement on the time response 15000 parallelized and 18000 serialized:



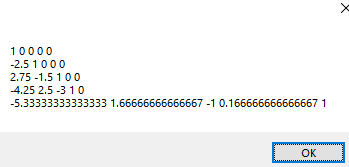


GMM, this clustering algorithm is very different from the previous ones. We first need to say how many clusters we want and then we will compute the data and create a mu matrix which will have the average (can be average position x and y, can be average color pixels RGB etc.) of each center. Then we will calculate the probability of each point to belong to each cluster. Finally, we will group the points to the cluster center which has more probability.

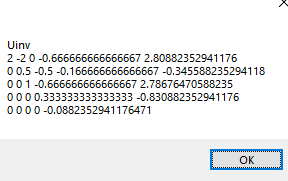
GMM serialized, time taken to compute and showing just part of the results:



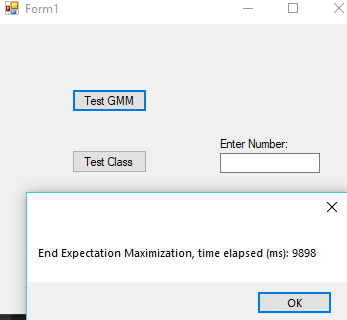
Linverse result:



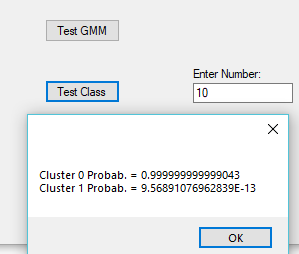
Uinverse result:



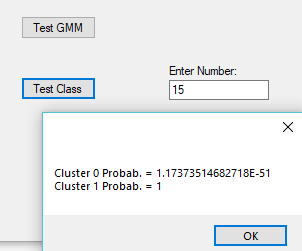
Using GMM paralelized in order to determine to what cluster one number belongs (the initial data has points closer to 10 and 15, the idea is that any point close to 10 or below 10 will belong to the cluster where the average is aprox 10, the rest will go to the cluster with average around 15):



Number 10 will have almost a probability of 1 to belong to cluster 0, which has avergae close to 10:



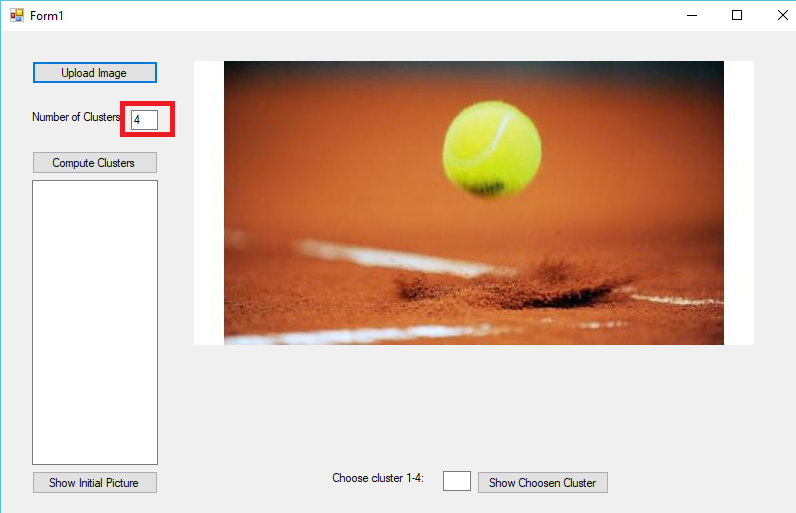
Number 15 has a probability of 1 to belong to cluster 1, which has averaage close to 15:



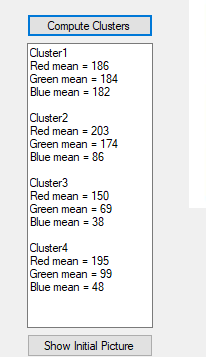
The last par of this assignment was to implement the GMM algorithm in a 3D problem applied to images. The three dimenssions will the the color of pixels RGB. The first step was to store in a matrix all the colors of each pixel. Once the matrix is created, the user will specify how many clusters we want and apply the GMM algorithm in a 3D problem. The GMM will end up creating a matrix with the average of RGB for each cluster. Then, we will calculate the probability of each pixel to belong to a cluster based on its RGB color. Once the clustering is computed the user will have two options:

* Specify the cluster number the user wants to see. Then, the program will white out all the other clusters
* Click on the image and all the clusters but the one where that point belongs will be white out.

Uploaded image, we will compute GMM with 4 clusters

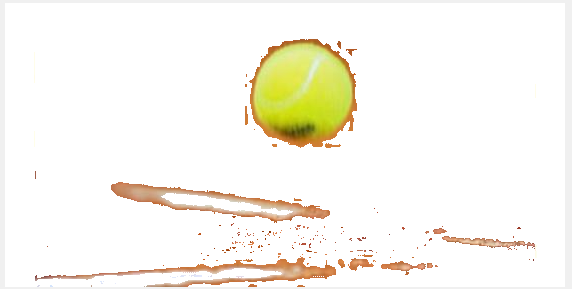


Once the computation is done, the average for each variable (RGB) of each cluster will appear on the left side:

****

Now we are able to click anywhere in the image or specify the cluster we want to see and just the determined cluster will show up:

Clicked on the ball:

****

Showing cluster 3:

****

**Conclusion:**

After completing this assignment I was able to understand better how Kmean, Kmeans++, Kmedoids and GMM work. Also, I was able to practice my skills in parallel computing by improving those algorithms.

The second part of this assignment helped me to understand better how GMM works and how to apply it to different dimensions problems