**K-Mean Clustering**

**Vuk Mićić**

**89221026**

**In this paper, I present a unified framework for implementing the K-means clustering algorithm. I will present how the UI works and how the GUI was implemented. I also explain how the K-Means clustering algorithm was implemented sequentially, in parallel and distributed.**

**1 Introduction**

K-means clustering is a method for partitioning n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. Given a set of data points, the algorithm aims to minimize the within-cluster variance.

Given a set of observations x1,x2,…,xn, where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k sets S={S1,S2,…,Sk}. So as to minimize the within-cluster sum of squares (WCSS):



where **μi** is the mean of points in **Si**.

The graphical representation of the clusters can be obtained by assigning a color to each data point according to its cluster membership. For each data point **x**, we determine its cluster by evaluating the distance to each cluster centroid and assigning it to the nearest one. This is repeated until the centroids do not change significantly or a predefined number of iterations is reached.

**Algorithm**

1. **function** k-means(k,X
2. Initialize μ1,μ2,…,μk (randomly or using a heuristic)
3. repeat
4. for each data point **xi** in **X** do
5. Assign **xi** to the cluster with the nearest centroid
6. end for
7. for each cluster **S**j do
8. Update the centroid **μj** to be the mean of all points in **Sj**
9. end for
10. until centroids do not change or max iterations are reached
11. return **S,μ**
12. **end function**

Here, k is the number of clusters, and the main algorithm assigns cluster labels to each data point based on the nearest centroid.

Since every data point is processed independently from the others, this algorithm can easily be parallelized and adapted to use multiple threads. Thus, the rest of the paper will describe the methods used for parallelization and the measured performance results for different degrees of parallelism.

**2. Design and Problem description**

I must make an user friendly UI. I must make a GUI that satisfies the project guidelines. toggle on/off GUI, Drawing of GUI should be done independently of computational threads. Default size 800x600. Use a library of a map. Graphical interface can scale, zoom and move. The program can be ran in different modes (sequential, parallel, distributed) by specifying a parameter. User can specify the number of accumulation sites and the number of clusters. The program measures run-time needed to complete. Every version (sequential, parallel, and distributed) measures cycles passed. Every update of positions of all particles is considered a cycle. All three implementations run the simulation until they reach a specified number of cycles. In case the number of sites is greater then the provided dataset, their positions and capacities must be chosen at random. Capacities should have an upper bound equal to the sample dataset. The positions must be in Europe and even though the equidistant distance can be used. The implementation must adapt automatically to the hardware it is being ran on (Physical CPU’s, Cores, Memory, etc..);. I want a lot of the sequential code to be reusable in distributed and parallel modes(such as data manipulation).This is achieved thanks to the clustering service interface. The GUI needed unique implementations to represent the design given in the project guidelines such as custom waypoints and waypoint renderer. Also the guidelines specify a unique data set for the initial 11k sites that will be used in the K-means clustering algorithm. Testing is identical in all three modes. Sequential, being the simplest was straightforward to develop, testing and calculating with be explained below. The problem of implementing parallel mode was solved using a thread pool, where each thread would do the cluster computation for a sub set of 1000 sites. Distributed is done using a server, where we open a socked and use in and out socket. Using the in socket we send it data to do the K-means clustering algorithm. After it computes it returns it.

**3.Implementation**

**MAIN**->contains prints for an aesthetic UI (specifications go to interface-> (clustering service):do we want sequential, parallel or distributed:, do we want to do testing or visuals the results, do we want lock clusters or locked sites, desired run time, desired number of clusters/sites(unique depending on what we lock)). If we pick visual we go to clusteringService.calculateKMeans() and invoke(later) a window frame, if testing we run clusteringService.runTesting().

**Clustering Service** exists so I don’t have to make 3 objects for each (sequential, parallel and distributed mode), more universal.

**GUI** -**Window Frame**: receives test results (sets window as in documentation), makes and adds panel and function to draw the dots addDots().

**MapViewPanel** – uses jxmapviwer and swing all (used for drawing dots). InitMap() chatgpt/github function- takes the map from the url as specified in documentation, and adds listeners as specified in documentation. AddDots()- Uses **CustomWaypoint**, because the default ones are like in google maps (red pin), this one satisfies the project aesthetic requirements(color). Also uses **DotwaypointRenderer** Class to draw dots instead of pins and makes centroids bigger and black. Method to generate n unique random colors.

**Data set**: contains information of all sites. ReadSitesFromFile -internet’s function that reads the Germany.json file. Rest methods are for data manipulation, they are here because they are used in all three modes. GetNSites, generateNsites, getFirstcentroid, RecomputeCentroids, CalculateCentroid, calculateTotalSSE, calculateClusterSSE, euclidianDistance, generateRandomDouble.

**Centroid**: has fancy name (is Boolean).

**Site**: Has same information from Germany.json file and new (Cluster number).

**Test result**- average test result method and and print data method.

**Sequential**: Most basic mode that uses a single thread for everything. When the object is created in the constructor it creates a data set

**Calculations(seq)**-> return cluster for visualization in main. SSE used to calculate centroid of cluster precision. It works by getting sites and centroids and going through each site and calculating what centroid it should be clustered in, it recomputes the centroids until SSE satisfies the precision. Calculation groups sites into cluster, recalculates centroids in the cluster so that it is in the center. Then it calculates how accurate the cluster is, if it satisfies the SSE it will end the calculations and return.

**Testing(seq)**->uses the same logic with minimal modifications. In testing we have two types of testing -locking clusters(20) and each iteration adds 500 sites, lock 30k sites and each iteration add 5 clusters (both of these are ran 3 times)((by project guidelines)). It receives the specification from the Main scanner. Gets n sites, runs while until it breaks for reasons (time limit, cluster limit). Runs 3 tests, averages the result. clusters 3 times less than sites(specification), -print test results method- prints test results

**Testing Type** in enum (locked sites, locked clusters)

**Interface**-> run testing method, calculate method, set number of clusters and sites method, set testing type method, set run time block method

**Parallel: Testing** is identical to sequential

**Calculating** is different, it uses threading. Sites are done with calculateClustersThreaded and centroids are done with recomputeCentroidsThreaded.

Calculate clustersThreaded-> is done by dividing a list into 1k sub lists that each are done by a thread and they work in parallel. After everything is done we return the entire list. Threads are done with thread pool (computer chooses how many threads it will use, dynamically chosen)

Clustering worker has a identical method in sequential. I use implements callable so I can return the result.

Similar logic for centroids. Centroid recompute worker has identical code as in sequential

**Distributed**-> **Testing** is sequential. Do all tests and send it back in one package.

**Distributed server**->opens the socket and accepts sockets(works kind of like a scanner) , and calulate thread gets a socket.

**Calculate Thread**->we divide the socket into in and out socket so we can receive and send info(object output and input stream).

**DataPackage**-> sent to the server.

**Distributed service**->sends to a server a data package.

**Calculate**-> send package using outsocket, get result and return it

**4.Testing**

Testing was done using 2 methods:

Locking cluster at 20 and increasing the number of sites by 500, it was run for 60s.

Locking the number of accumulation sites, I locked them at 30000 and set the number of clusters to 5, for each new configuration the number of clusters increase by 5. It was run for 60s.

Testing results are uploaded in a text file.

**5.Results and conclusions**

In both tests I notice a similar pattern where the parallel mode runs best. Sequential and Distributed do the same amount of tests and have an almost identical time in the second method. In the first method sequential outperforms distributed a little bit better. In conclusion the best mode for Running a K-mean Clustering algorithm is in parallel threading.