**GROUP NUMBER:** 36

**GROUP MEMBERS:** Francesco Visonà, Alessandro Dario, Luca Pellegrini

**AVAILABLE INPUTS:** Input files are available in the hdfs file system: /data/BDC2425/artificial1M7D100K.txt and /data/BDC2425/artificial4M7D100K.txt

**PART 1:** The goal of this test is to assess the scalability of the standard and fair implementations. The test must be performed on file artificial4M7D100K.txt. However, if your implementation is slow (i.e., taking more than 10 minutes for the slowest run), you can use the smaller file artificial1M7D100K.txt. You must use the following parameters: L=16, K=100, M=10.

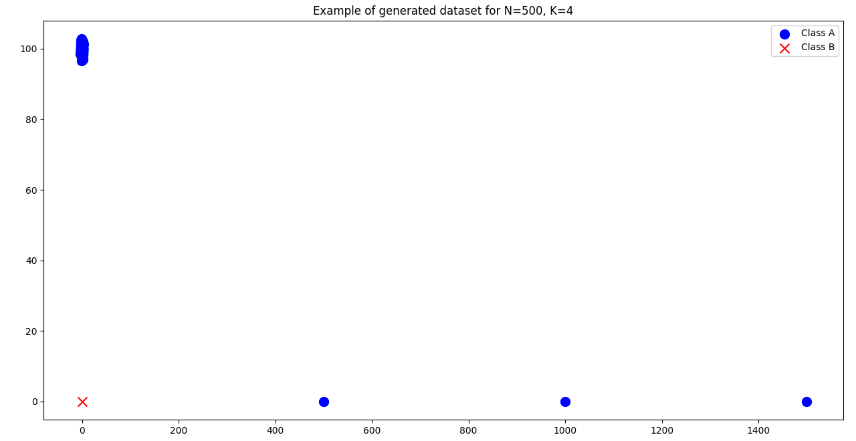
Fill in the following table.

**Name of used file:** /data/BDC2425/artificial4M7D100K.txt

|  |  |  |  |
| --- | --- | --- | --- |
| **SCALABILITY WITH RESPECT TO NUMBER OF EXECUTORS** | | | |
| **Number of executors** | **Spark Lloyd’s implementation** | **MRFairLloyd** | **MRComputeFairObjective** |
| 2 | 11802 | 41330 | 1905 |
| 4 | 10010 | 28395 | 1263 |
| 8 | 6822 | 16986 | 795 |
| 16 | 6821 | 7802 | 346 |

**General hints:**

* Remember that Spark uses the lazy evaluation for constructing an RDD. Therefore, be sure to include an action on the final RDD when you take running times.
* Any used RDD in your program should be cached.
* Do not include the reading of the input in your running times.

****

**PART 2:** Describe the program GxxGEN.java or GxxGEN.py that you have implemented for point 5 of the specifications. Include a brief high-level description of your program and the constraints, if any, on the input parameters (e.g., the minimum number of points *N*).

Our generator follows closely what is represented in *Figure 1*.

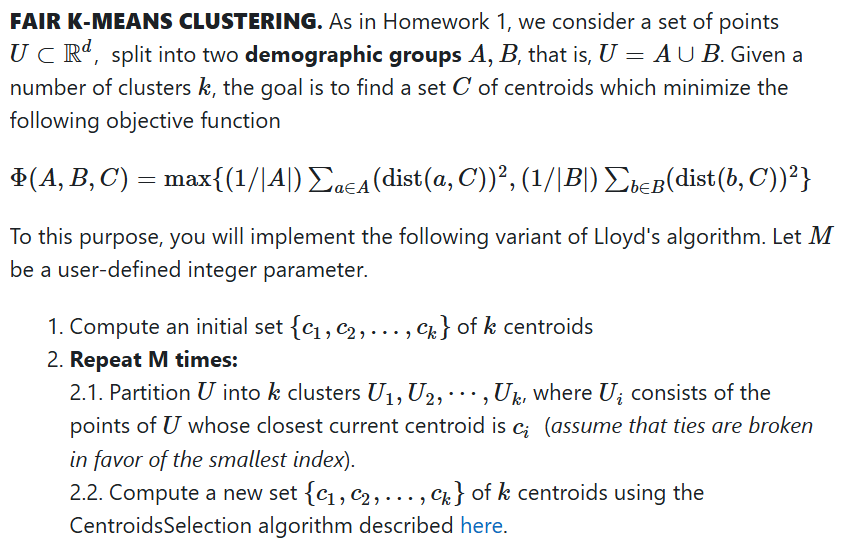
It simply creates K-1 points of group A at distance 500 one from the other on the x axis, starting from [500,0] (the blue points that are in [500,0], [1000,0] and [1500,0] in *Figure 1*). Then it puts one single point of group B in [0,0] and the remaining N-K points are given to group A and put near [100,0], in a ball of radius approximately 1 generated randomly.

Figure 1. Example of dataset generated by our procedure with N=500 and K=4.

This procedure is done to make it possible to obtain a dataset where the fair objective function of the solution returned by the standard Lloyd’s algorithm is bigger than the fair objective function of the solution returned by our fair version for all the N > K+1.

Both the algorithms will with high probability put K-1 centers in the points of group A that are on the x axis, but the real difference is in how they treat the other N-(K-1) points.  
While the standard K-Means algorithm will put a center closer to [100,0] (the larger N-K is, the closer the center will be to [100,0]), our fair implementation will put a center near [50,0].

This means that the objective function:



will be:

1. In the case of the standard Lloyd’s algorithm, considering the distance of every point of group A to its closest center to be really small, while the distance of the only point of group B to its closest center is 100:
2. In the case of our fair algorithm, if one of the centers is put in [50,0]:

This means that the objective function of the standard Lloyd’s algorithm will tend to be 4 times the objective function of our fair algorithm.

We’d like to notice that the real clustering done by our algorithm and the standard Lloyd’s algorithm depends heavily on the initialization of the centers, but with high probability the analysis will be right.