Universidade Federal do Pampa – UNIPAMPA
Ciência da Computação

AL02040 – APRENDIZADO DE MÁQUINA

Trabalho 2 e 3

K-means e PAM

HENRIQUE FAN DA SILVA - 161152061

1. INTRODUÇÃO

Este trabalho tem como objetivo fazer um comparativo dos algoritmos dos algoritmos de aprendizado não supervisionados K-means e Partitioning Around Medoids (PAM), analisando os resultados em cada um deles. Algoritmos estes que são usados para particionar dados em grupos, com a tentativa de minimizar as distâncias entre os pontos do grupo e um ponto que representa o centro desse conjunto.

1.1 K-means

O K-means clusteriza os dados de acordo com o ponto médio do centro de cada agrupamento.

O algoritmo K-means funciona da seguinte forma:

1. Definir um ‘k’, ou seja, uma quantidade de clusters;
2. Depois precisa definir, aleatoriamente, um centróide para cada cluster;
3. Cada ponto deve ser associado a um centróide, de menor distância;
4. O próximo passo é reposicionar o centróide. A nova posição do centróide deve ser a média da posição de todos os pontos do cluster;
5. Os passos 3 e 4 são repetidos, iterativamente, até obter a posição ideal do centróide.

1.2 Partitioning Around Medoids (PAM)

Diferente do K-means, o PAM não utiliza a média do centro de cada grupo, e sim, um dos pontos do próprio agrupamento, chamado de medoid. Nesse algoritmo, o custo é o somatório das distâncias do medoid para os pontos do grupo.

O método PAM funciona da forma a seguir:

1. Selecionar ‘k’ pontos aleatórios entre os pontos de dados como medoids;
2. Associar cada ponto dos dados ou medoid mais próximo;
3. Enquanto o custo diminui:
    a. Para cada medoid ‘m’ e para cada ponto que não é um medoid ‘o’:

i. Trocar ‘o’ e ‘m’, associar cada ponto dos dados ao medoid mais próximo, novamente, e recalcular o custo;
ii. Se o custo for maior do que o da etapa anterior, desfaça a troca.

2. METODOLOGIA

2.1 K-means

Na implementação K-means foi usada a distância euclidiana e apenas 50 iterações, pois já era suficiente para rodar nos dados fornecidos em aula.

```
x, y = readFile(k, file)
c, cx, cy = createCentroids(x, y, k)
for i in range(50):
    cx, cy = calCentroids(x, y, c, cx, cy, k)
    c = updateCluster(x, y, c, cx, cy, k)
```

Função ***readfile()*** lê os dados e cria uma lista de pontos com as coordenadas x e y. Na ***createCentroids()*** os centróides em cx e cy, c é um vetor que criado para especificar a qual cluster é inicializado cada ponto dos dados. A função ***calCentroids()*** atualiza os centróides em cada iteração, após vem a função que atualiza o cluster de cada ponto dos dados ***updateCluster()***.

2.2 PAM

Assim como no K-means, também foi usada a distância euclidiana para gerar o custo, que é o somatório das distância dos medóides para cada ponto.

```
x, y = readFile(k, file)

c, mx, my = createMedoids(x, y, k)

c = updateCluster(x, y, c, mx, my, k)

allowableLossCost = 10

while allowableLossCost > 0:
    for i in range(0, k):
        for j in range(0, len(c)):
            if c[j] == i:
                if (mx[i] != x[j]) and (my[i] != y[j]):
                    mx_old = mx[i]
                    my_old = my[i]
                    previousCost = calCost(x, y, c, mx, my, k)
                    mx[i] = x[j]
                    my[i] = y[j]
                    c = updateCluster(x, y, c, mx, my, k)
                    newCost = calCost(x, y, c, mx, my, k)

                    if previousCost < newCost:
                        allowableLossCost -= 1
                        mx[i] = mx_old
                        my[i] = my_old
                        c = updateCluster(x, y, c, mx, my, k)
```

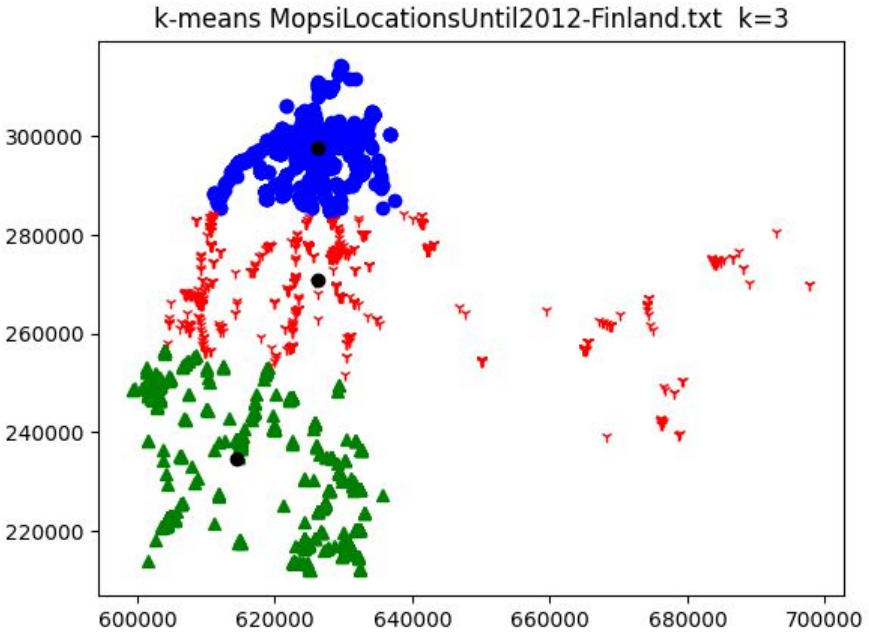
Função ***readfile()*** lê os dados e cria uma lista de pontos com as coordenadas x e y. Na ***createMedoids()*** os centróides em mx e my. Função que atualiza o cluster de cada ponto dos dados ***updateCluster()***. E a função ***calCost()*** é usada para calcular o custo antes e após a troca de medoid.

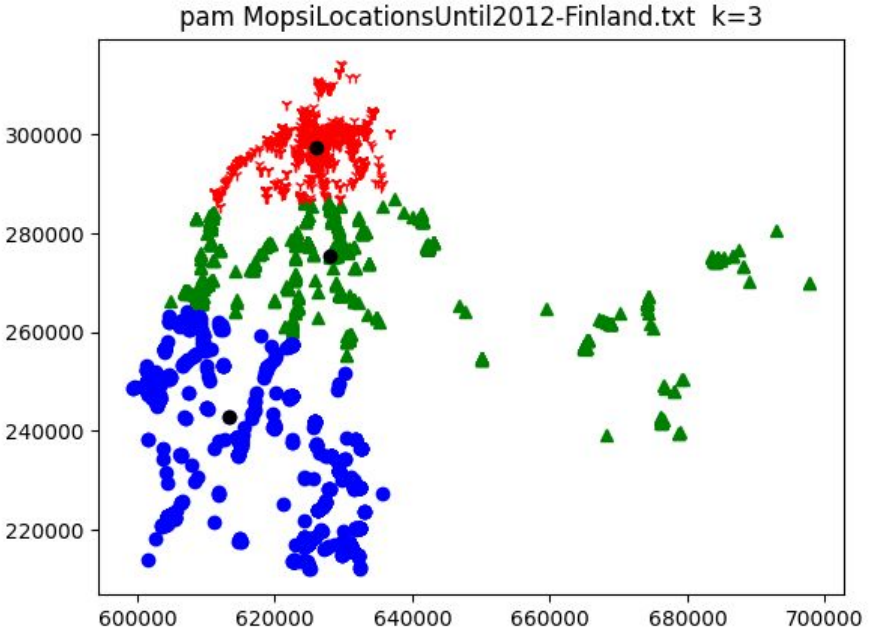
A variável ***allowableLossCost*** é usada para definir a condição de parada, permitindo apenas dez destrocas de medoids. Caso essas dez destrocas foram atingidas é porque os clusters estão relativamente reparados.

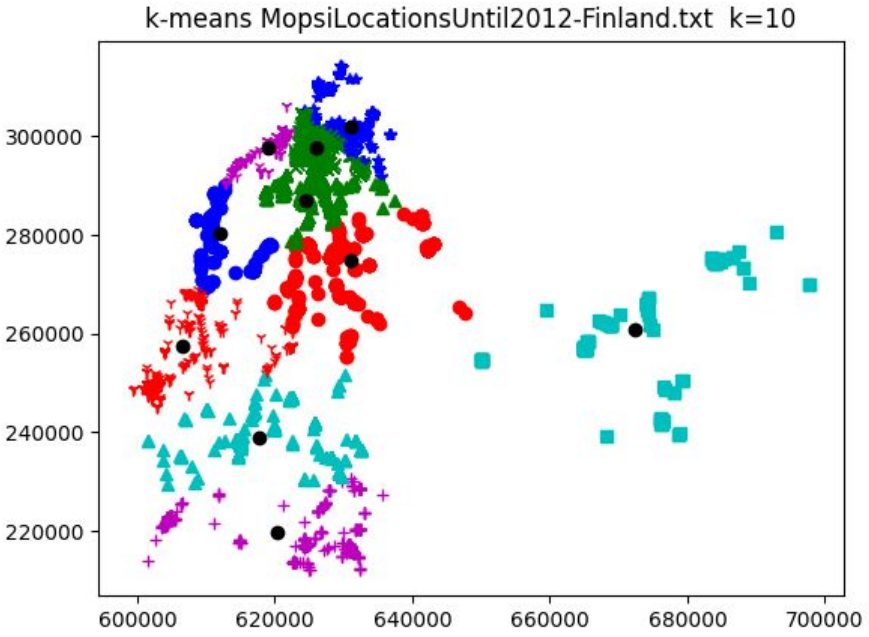
3. RESULTADOS

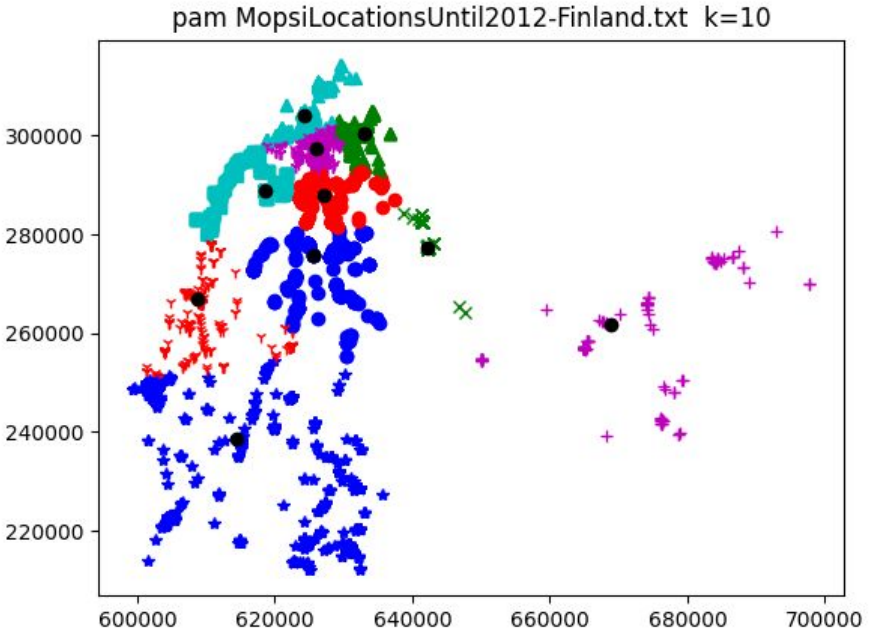
Em ambos os algoritmos foi testado com valores de k 3, 10, 20 com os dois dataset disponibilizado em aula:

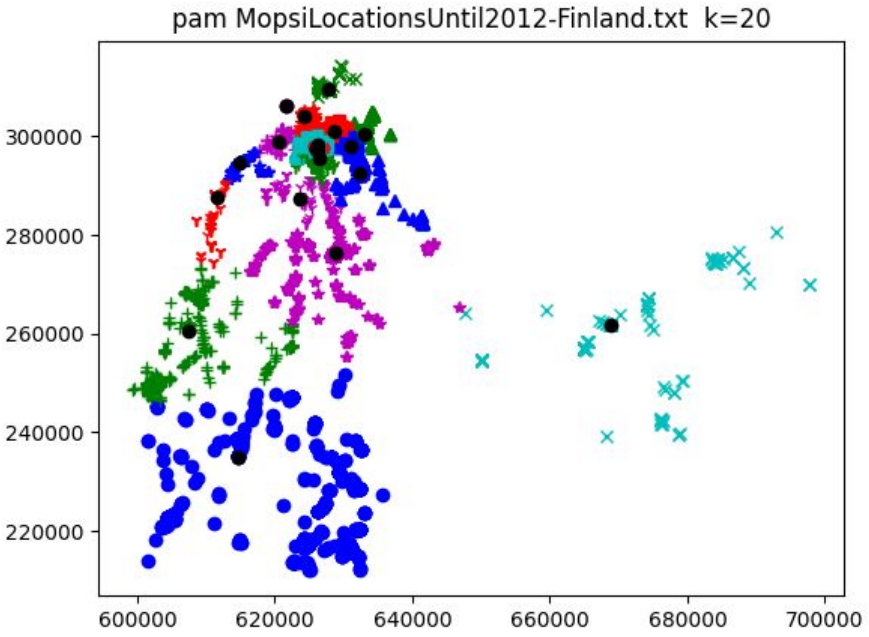
- MopsiLocationsUntil2012-Finland.txt
- MopsiLocations2012-Joensuu.txt

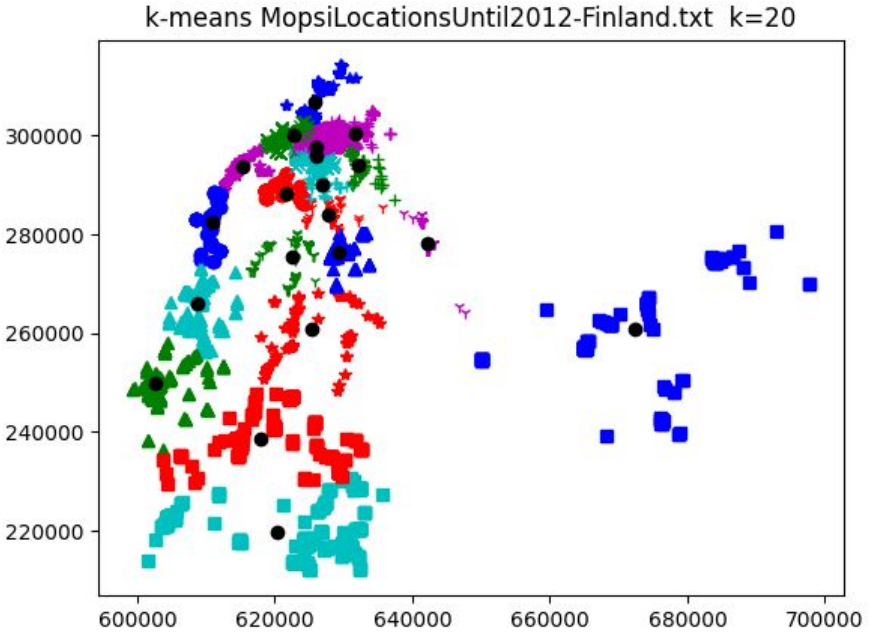

k-means MopsiLocationsUntil2012-Finland.txt k=3
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

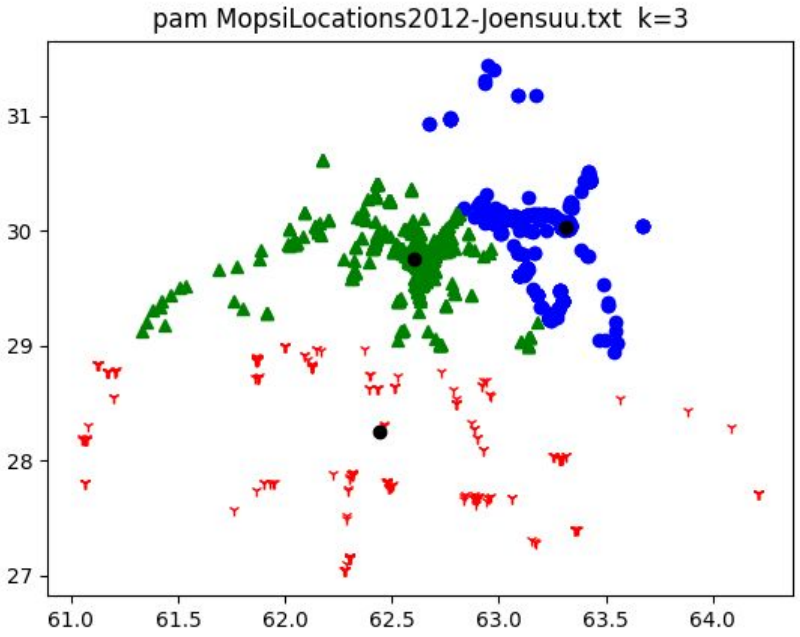

pam MopsiLocationsUntil2012-Finland.txt k=3
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

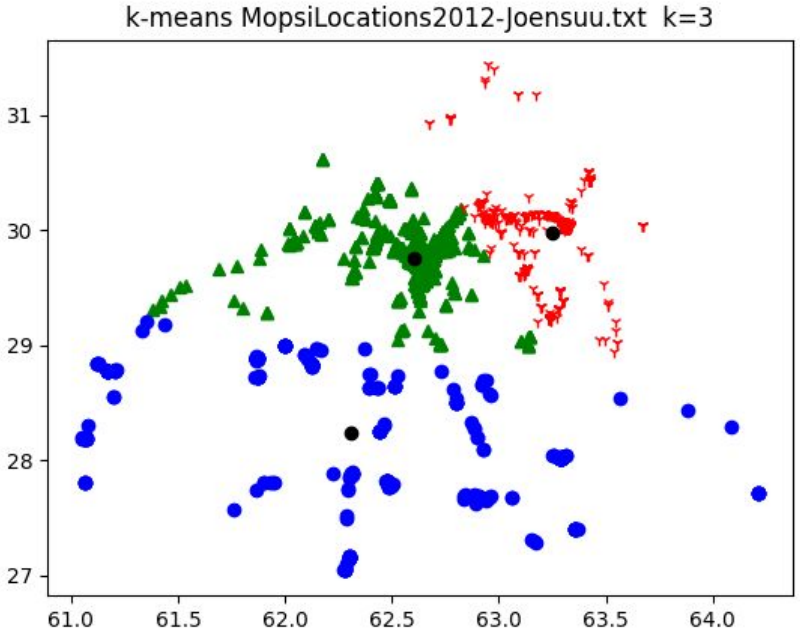

k-means MopsiLocationsUntil2012-Finland.txt k=10
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

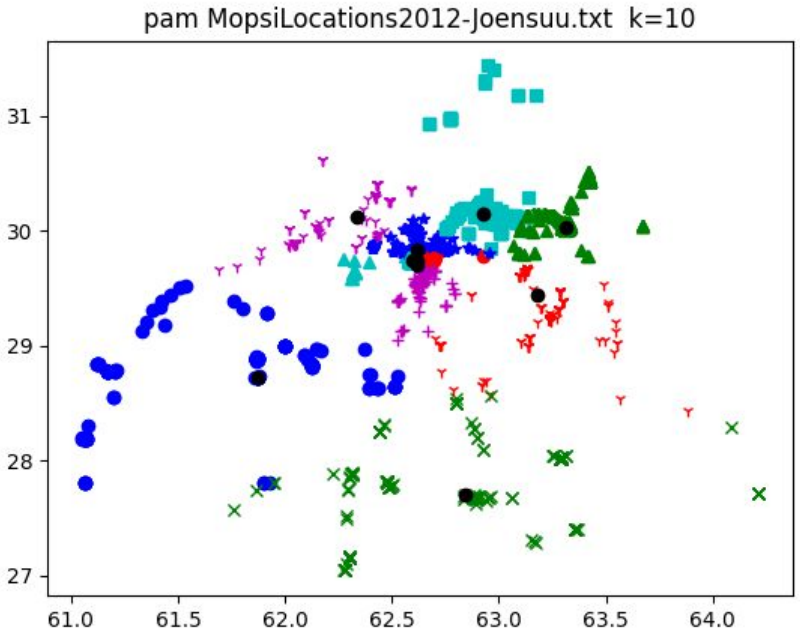

pam MopsiLocationsUntil2012-Finland.txt k=10
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

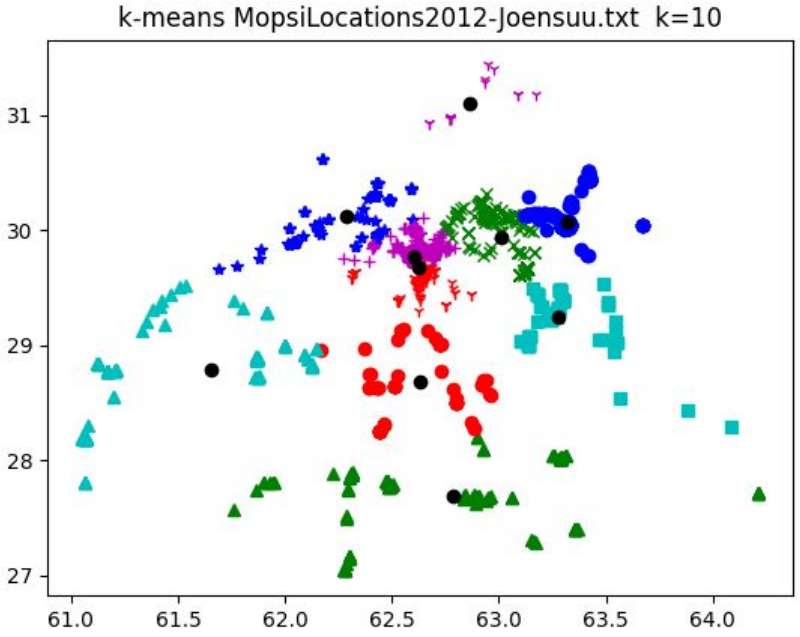

pam MopsiLocationsUntil2012-Finland.txt k=20
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

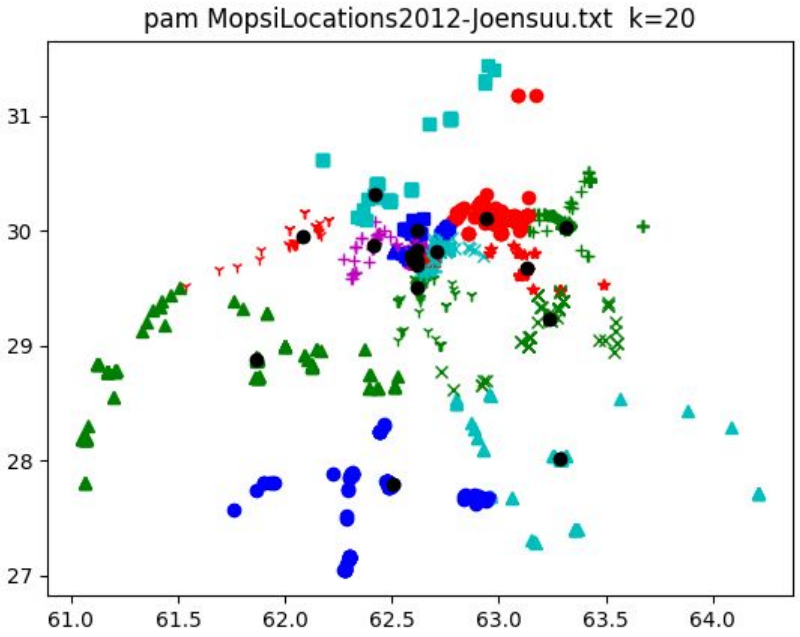

k-means MopsiLocationsUntil2012-Finland.txt k=20
300000
280000
260000
240000
220000
600000
620000
640000
660000
680000
700000

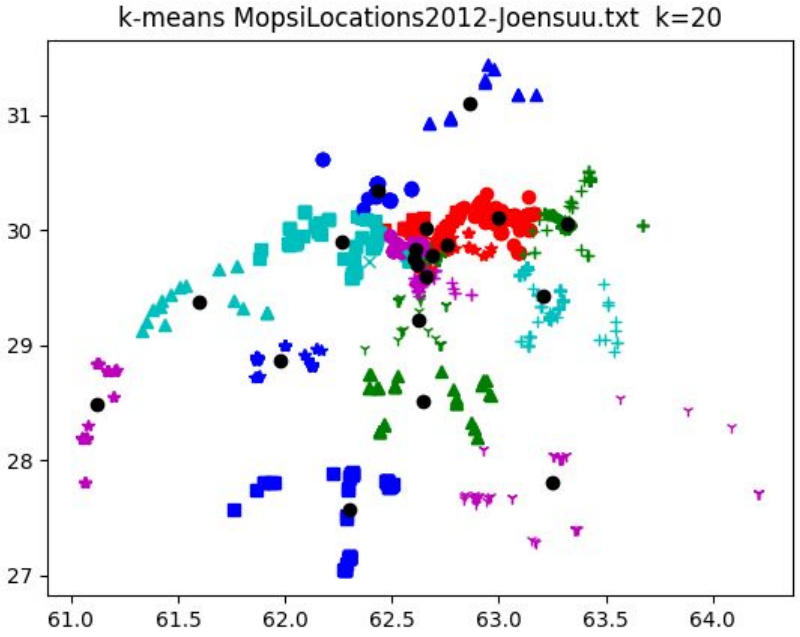

pam MopsiLocations2012-Joensuu.txt k=3
31
30
29
28
27
61.0
61.5
62.0
62.5
63.0
63.5
64.0


k-means MopsiLocations2012-Joensuu.txt k=3
31
30
29
28
27
61.0
61.5
62.0
62.5
63.0
63.5
64.0


pam MopsiLocations2012-Joensuu.txt k=10
31
30
29
28
27
61.0
61.5
62.0
62.5
63.0
63.5
64.0


k-means MopsiLocations2012-Joensuu.txt k=10
31
30
29
28
27
61.0
61.5
62.0
62.5
63.0
63.5
64.0





4. CONCLUSÃO

Tanto quanto o K-means e o PAM tiveram resultados muito parecidos, a diferença entre as posições dos pontos representativos dos clusters, em cada comparação dos resultados dos arquivos e da quantidade de clusters ‘k’, se deu pela inicialização aleatória. Outra diferença é o tempo de execução de cada algoritmo, o PAM por exemplo, demorou em torno de minutos nas execuções, enquanto o K-means ficou na casa dos segundos.