Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Zadanie 3 – Clustering

Olivér Izsák

Študijný program:Aplikovaná Informatika
Ročník: 2
Krúžok: Utorok 16:00
Predmet: Umelá inteligencia
Cvičiaci: Ing. Boris Slíž
Ak. rok: 2021/2022

Assignment description

Original:

Zadanie 4b – klastrovanie

Máme 2D priestor, ktorý má rozmery X a Y, v intervaloch od -5000 do +5000. Tento 2D priestor vyplňte 20 bodmi, pričom každý bod má náhodne zvolenú polohu pomocou súradníc X a Y. Každý bod má unikátne súradnice (t.j. nemalo by byť viacej bodov na presne tom istom mieste).

Po vygenerovaní 20 náhodných bodov vygenerujte ďalších 20000 bodov, avšak tieto body nebudú generované úplne náhodne, ale nasledovným spôsobom:

Náhodne vyberte jeden zo všetkých doteraz vytvorených bodov v 2D priestore.

Ak je bod príliš blízko okraju, tak zredukujete príslušný interval v nasledujúcich dvoch krokoch.

Vygenerujte náhodné číslo X_offset v intervale od -100 do +100

Vygenerujte náhodné číslo Y_offset v intervale od -100 do +100

Pridajte nový bod do 2D priestoru, ktorý bude mať súradnice ako náhodne vybraný bod v kroku 1, pričom tieto súradnice budú posunuté o X_offset a Y_offset

Vašou úlohou je naprogramovať zhlukovač pre 2D priestor, ktorý zanalyzuje 2D priestor so všetkými jeho bodmi a rozdelí tento priestor na k zhlukov (klastrov). Implementujte rôzne verzie zhlukovača, konkrétne týmito algoritmami:

k-means, kde stred je centroid

k-means, kde stred je medoid

aglomeratívne zhlukovanie, kde stred je centroid

divízne zhlukovanie, kde stred je centroid

Vyhodnocujte úspešnosť/chybovosť vášho zhlukovača. Za úspešný zhlukovač považujeme taký, v ktorom žiaden klaster nemá priemernú vzdialenosť bodov od stredu viac ako 500.

Vizualizácia: pre každý z týchto experimentov vykreslite výslednú 2D plochu tak, že označkujete (napr. vyfarbíte, očíslujete, zakrúžkujete) výsledné klastre.

Dokumentácia musí obsahovať opis konkrétne použitých algoritmov a reprezentácie údajov. V závere zhodnoťte dosiahnuté výsledky ich porovnaním.

Poznámka: Je vhodné použiť rôzne optimalizácie pre dostatočne efektívnu prácu Vášho zhlukovača. Napríklad pre aglomeratívne zhlukovanie je možné použiť 2-rozmernú maticu vzdialenosti dvojíc bodov. Naplnenie takejto matice má kvadratickú zložitosť. Potom sa hľadá najbližšia dvojica (najmenšie číslo v matici), to má opäť kvadratickú zložitosť, ale nenásobia sa tie časy, ale sčítavajú. Po výbere najbližšej dvojice túto dvojicu treba zlúčiť a tým sa zníži veľkosť matice o 1 (lebo sa zníži počet zhlukov o 1) Pri tom sa aktualizujú len vzdialenosti pre tento nový zhluk (len jeden stĺpec/riadok, zvyšok matice ostáva nezmenený).

The realization of the problem

Clustering is the act of grouping a set of objects together on a specific shared pattern. It is mainly used for statistical data analysis, pattern recognition and machine learning.

This implementation of the assignment contains 4 different clustering algorithms.
Which are the following: - **K-means with centroid**
- **K m-means with medoid**
- **Agglomerative clustering**
- **Divisive clustering**

Each of these algorithms have their advantages and disadvantages.
Such advantages for K-means algorithms are: easy to implement, scales to large data sets and guarantees convergence. Meanwhile the disadvantages are that the K variable has to be chosen manually, it is dependant on initial values, it is sensitive to outliers.(Singular points of data away from the big cluster).
For agglomerative and divisive clustering the advantages that it is efficient for small data sets, but computationally demanding.(Both have O(n3) time complexity)

For my implementation I used 5 classes for my solution:

Point

These are the coordinates for the points. Contains x and y coordinates.

Clusters

This object represents the clusters. It has a center which is represented by a point.
Also has an arraylist of points, these are the points that are part of the cluster.
Has a centerX and centerY variables which are for adding together the coordinates of all the cluster points . It contains a method addCenterSum, which stores the coordinates in the above mentioned variables. Then getCenterSum function calculates the centroid coordinates and returns them in an array.

Generator

This one is responsible for creating the 2D environment for the points, this is why it extends the Jframe. Its job is assigning a specific colour for each cluster when visualizing the points/clusters, this is why it also contains another class inside which is the color class(Generator2).

Algorithms

The class that contains all the clustering algorithms. The algorithms will be described later in the documentation.

Start

This class is responsible for starting the program, it contains a basic UI which asks the user to choose from number of generated points, then asks the user to choose one of the 5 options. 1.K-means-Centroid,2.K-means-Medoid,3.Agglomerative,4.Divisive,5.All 4 so they can be compared.
Then depending on the chosen option it will ask the user to input the K value for the K-means algorithms.

Algorithm implementations

Generating the startin points:

This part of the program is the same for all algorithms, since here only points are generated between -5000 to 5000 coordinates.
First the 20 starting points around the whole coordinate system, but with the rule that 2 points cannot be 400 distance close to each other, this way the clumping of clusters can be avoided. Then the following N amount of points are generated the with the rule that they have to be in 100 pixel radius from one of the already generated points.

K-means with Centroid

The K-mean algorithm with Centroid implementation works the following way:

1.Out of the generated N+20 amount of points K amount is choosen to become a cluster center. These custer centers cannot be closer than 500 pixel to each other. This way they are spread out more evenly.
2.The algorithm traverses the arraylist that contains all the generated points, and at each point compares them to all the cluster centers and the one cluster center that is closest to the selected point will become the cluster that will contain that point.This goes on in a loop, until all points have a corresponding cluster they belong to.
3.After all the points have been assigned, the program calculates the new centroids for each cluster by adding together all the coordinates and dividing that with the number of points in that cluster. That coordinate will become the new centroid.
4.Point 3. and 4. are repeated until the last centroid will be equal to the currently calculated one. This means that we have the final solution.

K-means with Medoid

The K-mean algorithm with Medoid implementation works the following way:

1.Out of the generated N+20 amount of points K amount is choosen to become a cluster center. These custer centers cannot be closer than 500 pixel to each other. This way they are spread out more evenly.
2.The algorithm traverses the arraylist that contains all the generated points, and at each point

compares them to all the cluster centers and the one cluster center that is closest to the selected point will become the cluster that will contain that point.This goes on in a loop, until all points have a corresponding cluster they belong to.
3.After all the points have been assigned, the program calculates the new Medoids for each cluster by adding together all the coordinates and dividing that with the number of points in that cluster. Then goes through all the points in that cluster and chooses the one that is closest to the calculated centroid and that closest one will become the medoid. Since it is possible to have more points that can be medoids An arraylist was also added that contains the candidate medoids, so when the algorithm gets stuck in a loop it will use different medoids for a potential better solution, or just to get out of it, and another thing was added for avoiding being stuck is shuffle for the arraylist of points, so more medoid candidates are tried out.
4.Point 3. and 4. are repeated until the loop variable that is being incremented at the end of the while loop, reaches 1001.

Agglomerative

The agglomerative algorithm implementation works the following way:

1.Make all the points into an individual cluster and add them to an arraylist of clusters.
2.While loop,Call the getClosest method. Which calculates the 2 closest clusters, and returns them. Then it merges them into one cluster, by calculating a new center from their average, and adding all the points to the new merged cluster. Then calculate the new merged cluster's average distance from its points. If its above 500 then it is discarded, and removed from the list. If its below then then the merged one gets added to the results, while their parts are removed from the arraylist of clusters.If there is only one cluster left,add it to the result.

Divisive

The divisive algorithm implementation works the following way:

1.Create the starting cluster , take all the points and assign them into the one starting cluster. Calculate the center of the cluster.Then call the div() function, with the cluster and all the points as the parameters.
2.The div() is a recursive function. It takes the cluster given as parameter and an arraylist of points, then calculates the clusters average distance from the center and if its below or equal to 500 then it adds to the final result. If it is not below 500 then it calls the K-mean centroid algorithm with K variable being 2, and then the 2 resulting clusters are put into another 2 div() functions. This goes on recursively until all of them go below 500 average distance from center.

Testing,results and evaluation

I did the following tests for my implementation:

1.Testing for K-means Centroid/Medoids from K=1 to K=20, 20 000 points generated.

2.Testing for K-means Centroid/Medoid,Agglomerative,Divisive k=15, k=20, 20 000 points generated.

The test results explained:

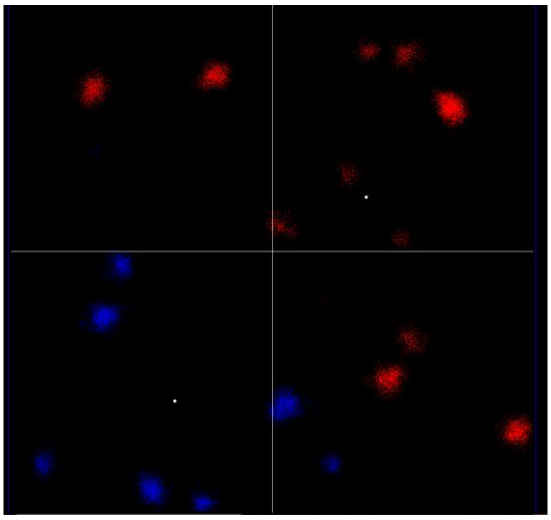
I will not show all the test results from k=1 to k=20, just some specific ones.

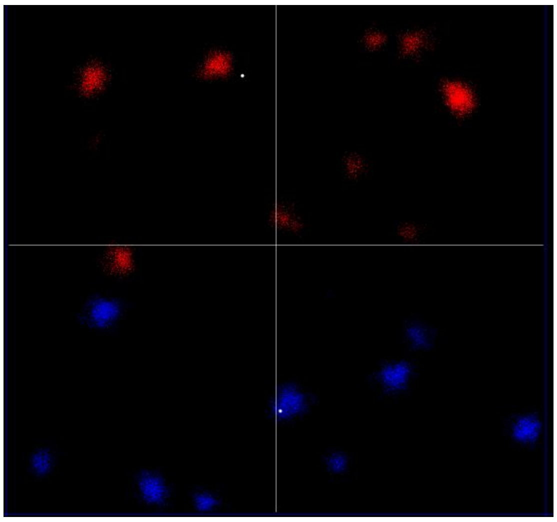
The overall average means: the sum of all clusters average distance from points divided by the number of clusters.

Amount Cluster avg>500 means: the eucledian sum of all points from the middle of the cluster divided by the amount of points in the cluster

K=2

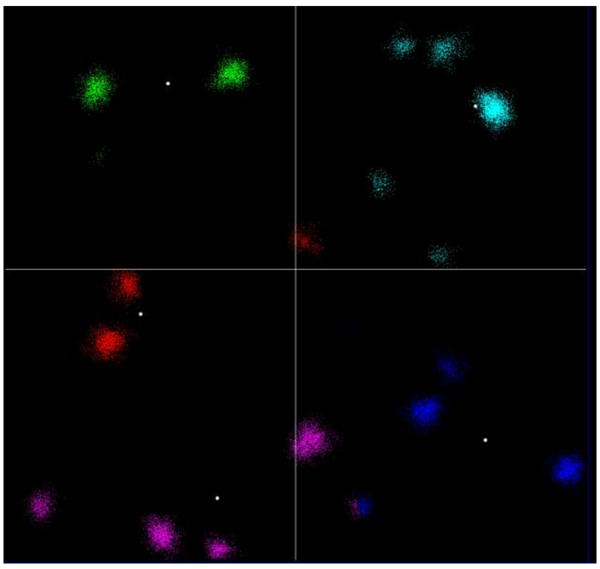
Type:	Centroid	Medoid
Overall average:	2869	2928
Amount cluster avg >500:	2	2

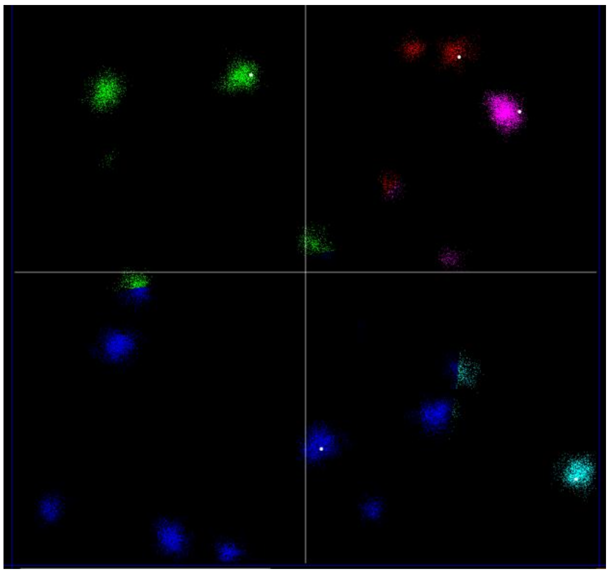




K=5

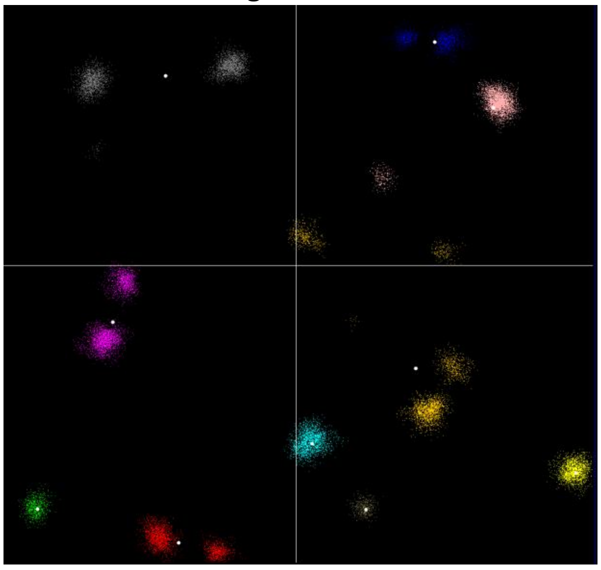
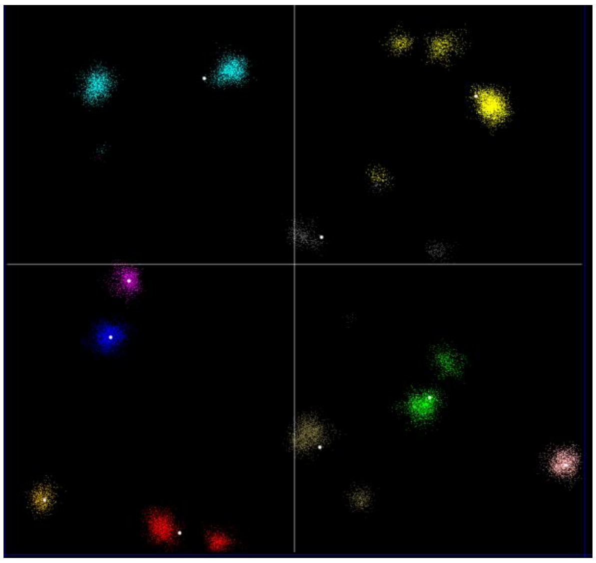
Type:	Centroid	Medoid
Overall average:	1192	1191
Amount cluster avg >500:	5	4





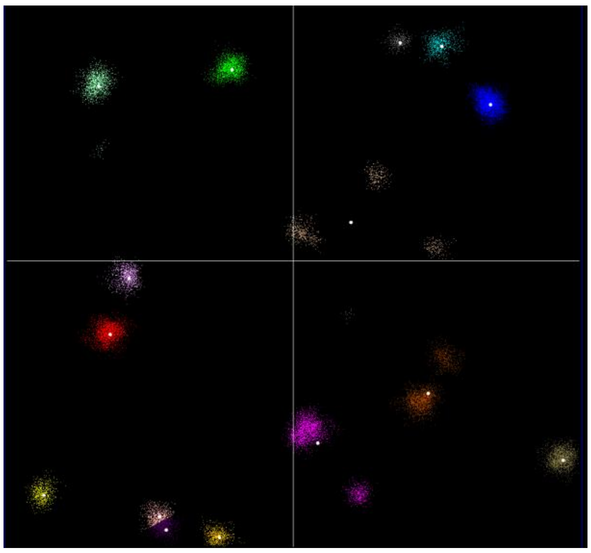
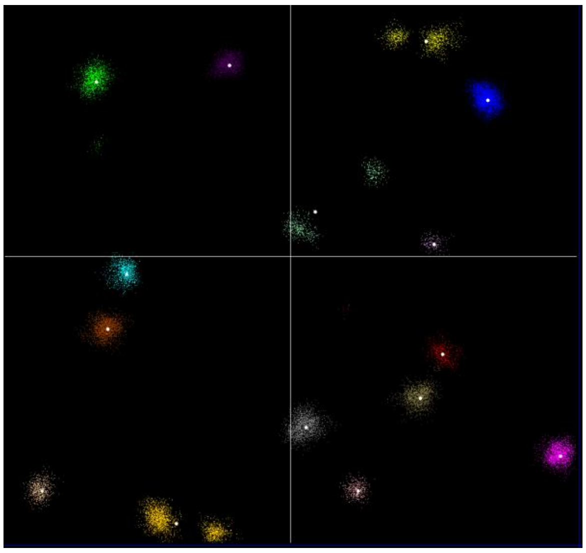
K=10

Type:	Centroid	Medoid
Overall average:	463	456
Amount cluster avg >500:	2	3

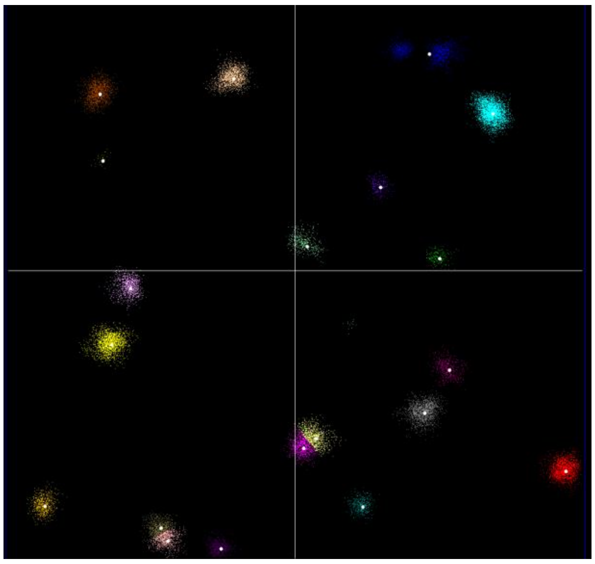
K=15

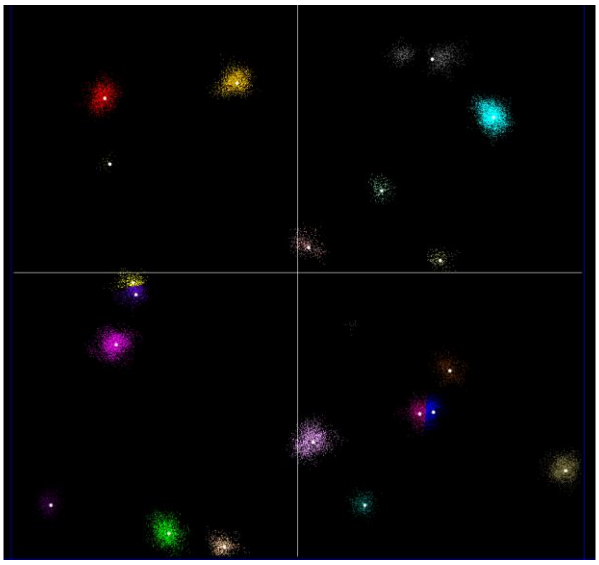
Type:	Centroid	Medoid
Overall average:	246	235
Amount cluster avg >500:	1	1

K=20

Type:	Centroid	Medoid
Overall average:	162	161
Amount cluster avg >500:	0	0





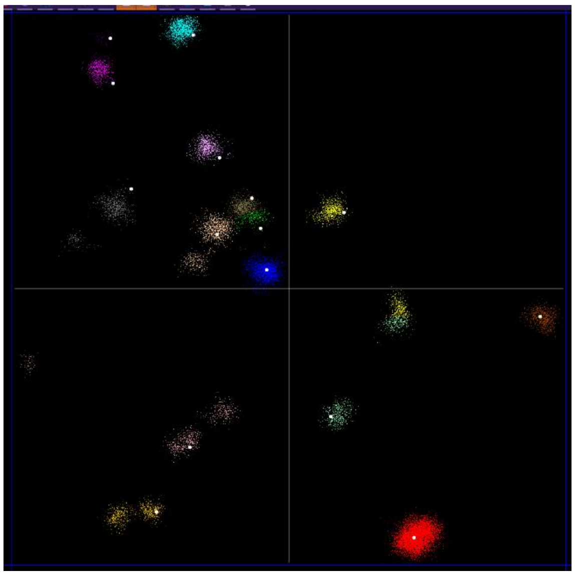
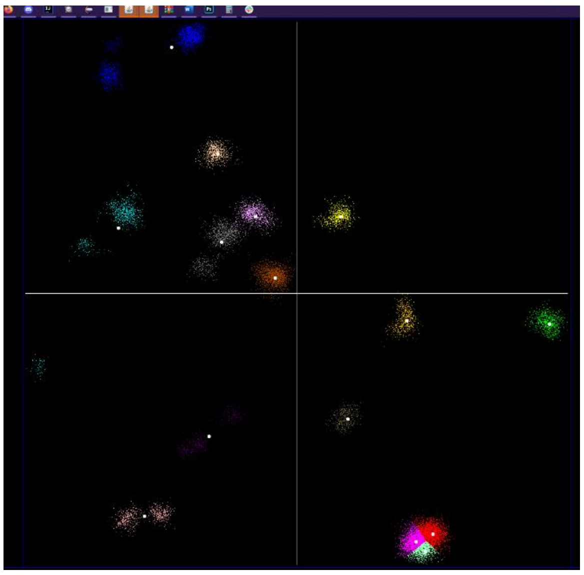
As we can see from these tests, the higher the K value, the smaller the overal average and the amount of cluster averages that are below 500. Also we can see a constant better overall average for medoid, but only slightly better.

Above K=14 , I shouldn’t be getting any clusters that have an average above 500. Since around 14-15 is the K value, where it is actually possible to get results below 500. One way to improve the amount of cluster average so it is less frequently below 500 is to run the first part of the algorithm (where it chooses the starting location/center of the cluster randomly) multiple times, and choose the one that gives the best results.

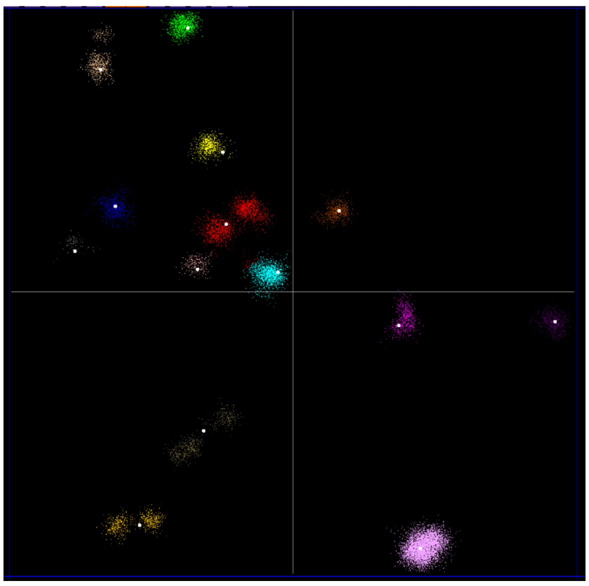
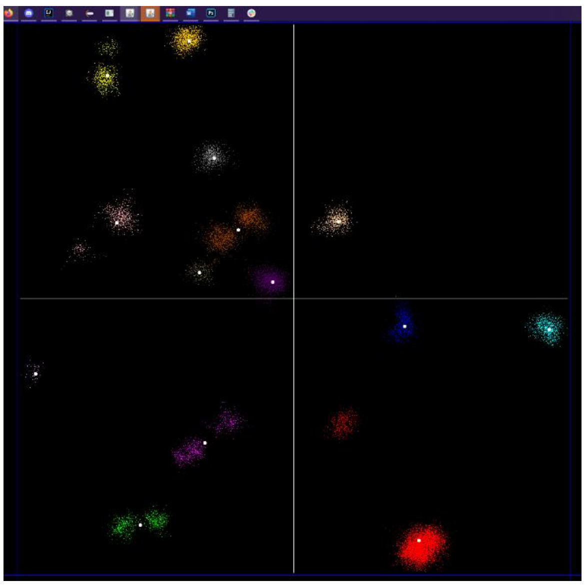
Test 2: Comparing all algorithms at k=15 and k=20 + divisive,agglomerative(no K value for these)

K=15

Type:	Centroid	Medoid
Overall average:	277	261
Amount cluster avg >500:	1	1

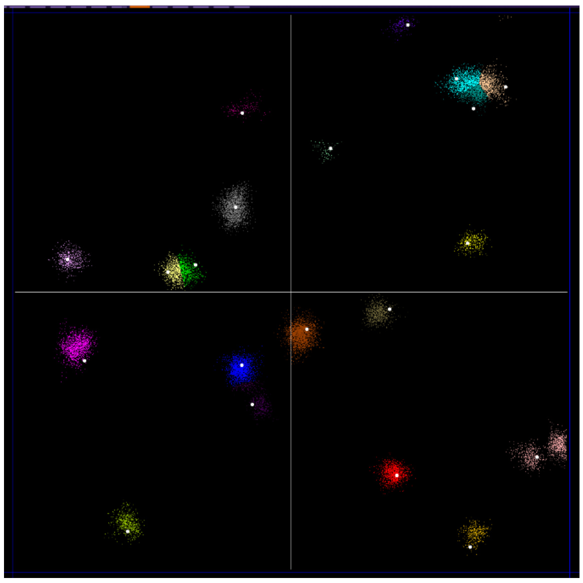
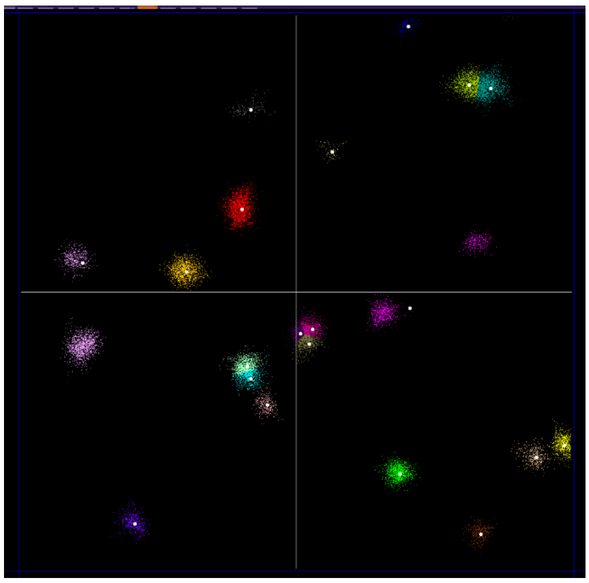



Type:	Agglomerative	Divisive
Overall average:	241	237
Amount cluster avg >500:	0	0
Amount of clusters:	14	14

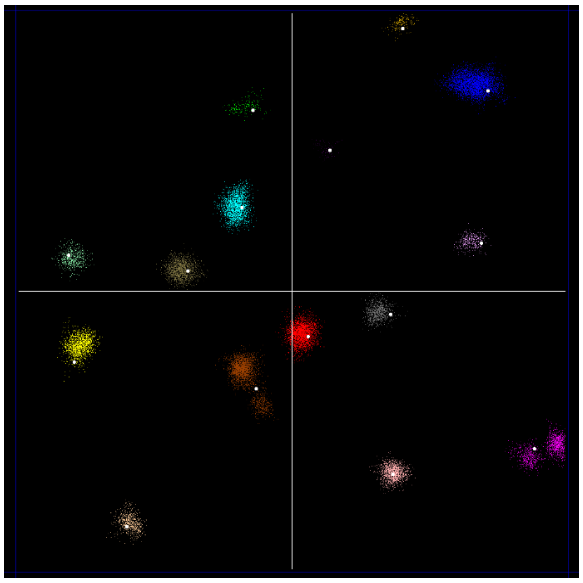
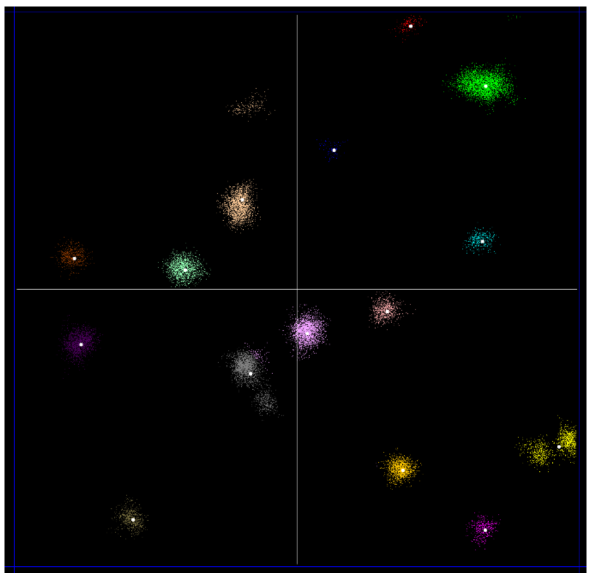



K=20

Type:	Centroid	Medoid
Overall average:	162	235
Amount cluster avg >500:	0	0

Type:	Agglomerative	Divisive
Overall average:	232	190
Amount cluster avg >500:	0	0
Amount of clusters:	15	15

The average execution time for: K-means centroid was 0.25 second,
K-means medoid was 6.5 second,
Agglomerative was 37800 second(10.5 hour),
Divisive was 0.5 second.

I would like to point out a mistake in my agglomerative algorithm where the last cluster is not being displayed in the 2d environment for some reason. I didn't manage to find the bug, because when I noticed it I was already out of time, since testing was really time consuming(10 hours).

Conclusion

In conclusion, I think my program's performance it statisfactory.
For the K-means medoid/centroid algorithms I wasn't able to make it so it always creates clusters that have an average distance below 500. However depending on the amount of K points it is sometimes unavoidable to go above 500. Also as the K number increases so does the the average distance decrease below 500. One way I could have made sure to more reliably get an average distance below 500 is to put the algorithm in a loop, and continue creating new ones, until I get a suitable position for the starting clusters at the beginning of the algorithm. However this is just basically playing with random generation.

The agglomerative algorithm is probably the most accurate but also takes the most amount of time to execute so it was hard to test and there was also that bug with the display I mentioned in the testing.

Meanwhile the divisive algorithm is the second most accurate one and it also takes the shortest amount of time to execute.