Umelá inteligencia

Zadanie č. 3b — klastrovanie

Obsah

[Riešený problem 1](#_Toc152673054)

[Opis riešenia 2](#_Toc152673055)

[Reprezentácia údajov 2](#_Toc152673056)

[Použitý algoritmus 2](#_Toc152673057)

[K-means (stred je centroid) 2](#_Toc152673058)

[K-medoids (stred je medoid) 3](#_Toc152673059)

[Divízne zhlukovanie (stred je centroid) 4](#_Toc152673060)

[Zhodnotenie riešenia 4](#_Toc152673061)

[Výsledky testovania a vizualizácia 4](#_Toc152673062)

[1. príklad 5](#_Toc152673063)

[2. príklad 8](#_Toc152673064)

[3. príklad 11](#_Toc152673065)

[Záver 13](#_Toc152673066)

# Riešený problem

Máme 2D priestor, ktorý má rozmery X a Y, v intervaloch od -5000 do +5000. Tento 2D priestor treba vyplniť 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 treba vygenerovať ďalších 40000 bodov, avšak tieto body nebudú generované úplne náhodne, ale nasledovným spôsobom:

1. Náhodne vybrať jeden zo všetkých doteraz vytvorených bodov v 2D priestore (nielen z prvých 20)
2. Ak je bod príliš blízko okraju, tak zredukovať príslušný interval, uvedený v nasledujúcich dvoch krokoch
3. Vygenerovať náhodné číslo X\_offset v intervale od -100 do +100
4. Vygenerovať náhodné číslo Y\_offset v intervale od -100 do +100
5. Pridať 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

Úlohou je naprogramovať zhlukovač pre 2D priestor, ktorý zanalyzuje 2D priestor so všetkými jeho bodmi a rozdelí tento priestor na k zhlukov (klastrov).

Na riešenie problému treba použiť k-means (stred je centroid), k-medoids (stred je medoid) a divízne zhlukovanie (stred je centroid).

Za úspešný zhlukovač považujeme taký, v ktorom žiaden klaster nemá priemernú vzdialenosť bodov od stredu viac ako 500.

# Opis riešenia

Pre túto implementáciu som použil programovací jazyk Python (verzie 3.11) a vývojové prostredie IntelliJ IDEA Ultimate (2023.3 EAP). Na vizualizáciu údajov som použil modul "matplotlib".

## Reprezentácia údajov

Množina 40000+20 vygenerovaných bodov je reprezentovaná zoznamom (x, y)-dvojíc súradníc.

## Použitý algoritmus

### K-means (stred je centroid)

Hlavnou podstatou algoritmu K-means je výber K náhodných bodov, ktoré sa majú stať centroidmi zo z množiny údajov, priradenie každého údajového bodu k najbližšiemu centroidu (pomocou euklidovskej vzdialenosti) a iteračná aktualizácia centroidov, kým sa už výrazne nemenia (a ani zhluky) alebo kým sa nedosiahne určený počet iterácií. Centroidy sú stredové body zhlukov, ktoré predstavujú priemernú polohu všetkých dátových bodov priradených k príslušnému zhluku. Sú to čisto geometrické body, ktoré nemusia byť vždy skutočnými prvkami údajovej množiny.

def calculate\_centroid(cluster):  
 if not cluster:  
 return None  
  
 c\_x = sum(point[0] for point in cluster) / len(cluster)  
 c\_y = sum(point[1] for point in cluster) / len(cluster)  
  
 return c\_x, c\_y  
  
def k\_means(points, k, max\_iterations=1000):  
 start\_time = time.time()  
  
 *# Initial random centroid generation* centroids = random.sample(points, k)  
 clusters = [[] for \_ in range(k)]  
  
 for \_ in range(max\_iterations):  
 *# Generate K clusters* new\_clusters = [[] for \_ in range(k)]  
  
 *# Assign each point to the nearest centroid (based on the Euclidean distance)* for point in points:  
 closest\_centroid\_index = min(range(k), key=lambda i: euclidean\_distance(point, centroids[i]))  
 new\_clusters[closest\_centroid\_index].append(point)  
  
 *# Check for convergence by comparing old clusters with new clusters* if clusters == new\_clusters:  
 break  
  
 clusters = new\_clusters  
  
 *# Update centroids to the mean of the points within each cluster* for j in range(k):  
 if clusters[j]:  
 centroids[j] = calculate\_centroid(clusters[j])  
  
 return clusters, centroids, time.time() - start\_time

### K-medoids (stred je medoid)

Algoritmus K-medoids s použitím medoidov funguje takmer rovnako ako vyššie uvedený algoritmus K-means s výnimkou spôsobu výpočtu centrálnych bodov. Medoid je najreprezentatívnejší alebo najcentrálnejší bod v zhluku, ktorý minimalizuje rozdielnosť voči všetkým ostatným bodom v zhluku. Je to skutočný údajový bod zo z dátovej množiny na rozdiel od centroidu.

def k\_medoids(points, k, max\_iterations=1000):  
 start\_time = time.time()  
  
 *# Initial random medoid selection* medoids = random.sample(points, k)  
 clusters = [[] for \_ in range(k)]  
  
 for \_ in range(max\_iterations):  
 *# Generate K clusters* new\_clusters = [[] for \_ in range(k)]  
  
 *# Assign each point to the nearest medoid (based on the Euclidean distance)* for point in points:  
 closest\_medoid\_index = min(range(k), key=lambda i: euclidean\_distance(point, medoids[i]))  
 new\_clusters[closest\_medoid\_index].append(point)  
  
 *# Check for convergence by comparing old clusters with new clusters* if clusters == new\_clusters:  
 break  
  
 clusters = new\_clusters  
  
 *# Update medoids by choosing the point that minimizes total distance within clusters* for j in range(k):  
 if clusters[j]:  
 *# Calculate distances between points within the cluster* distances = [sum(euclidean\_distance(point, p) for p in clusters[j]) for point in clusters[j]]  
 *# Find the index of the point that minimizes the total distance* min\_distance\_index = distances.index(min(distances))  
 medoids[j] = clusters[j][min\_distance\_index]  
  
 return clusters, medoids, time.time() - start\_time

### Divízne zhlukovanie (stred je centroid)

Algoritmus divízneho zhlukovania je prístup hierarchického zhlukovania "zhora nadol", ktorý začína priradením všetkých dátových bodov k jednému zhluku a považuje ho za koreňový zhluk, ktorý zahŕňa celý dataset. Potom sa nájde najväčší zhluk (ten, ktorý obsahuje najviac bodov) a rozdelí sa na polovicu. Na túto časť som sa rozhodol použiť K-means (K=2). Rozdelenie sa uskutoční, kým je priemerná vzdialenosť každého zhluku od jeho stredu väčšia ako 500 a počet zhlukov je menší ako veľkosť datasetu.

def calculate\_avg\_distance\_from\_center(cluster, center):  
 return sum(euclidean\_distance(center, p) for p in cluster) / len(cluster)

def divisive\_clustering(points):  
 start\_time = time.time()  
  
 *# A single cluster in the beginning* clusters = [points]  
  
 successful = False  
  
 while len(clusters) < len(points) and not successful:  
 max\_cluster = max(clusters, key=len) *# Find the cluster with the maximum number of points* if max\_cluster:  
 *# Using K-means to halve the largest cluster* split, \_, \_ = k\_means(max\_cluster, 2)  
  
 *# Replace the largest cluster with the two subclusters* clusters.remove(max\_cluster)  
 clusters.extend(split)  
  
 successful = not any(calculate\_avg\_distance\_from\_center(cl, calculate\_centroid(cl)) > 500 for cl in clusters)  
 else:  
 break  
  
 return clusters, successful, time.time() - start\_time

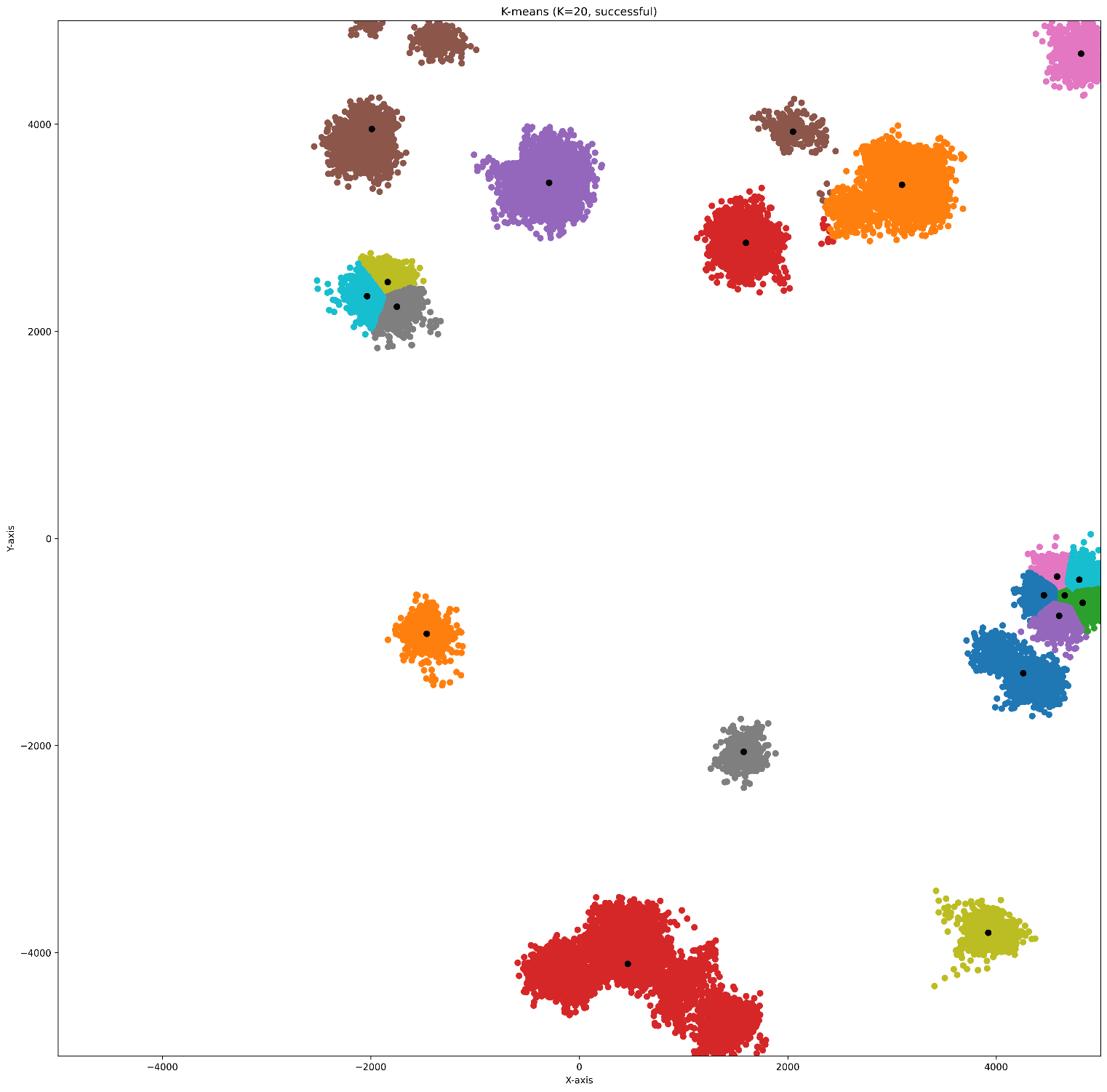
# Zhodnotenie riešenia

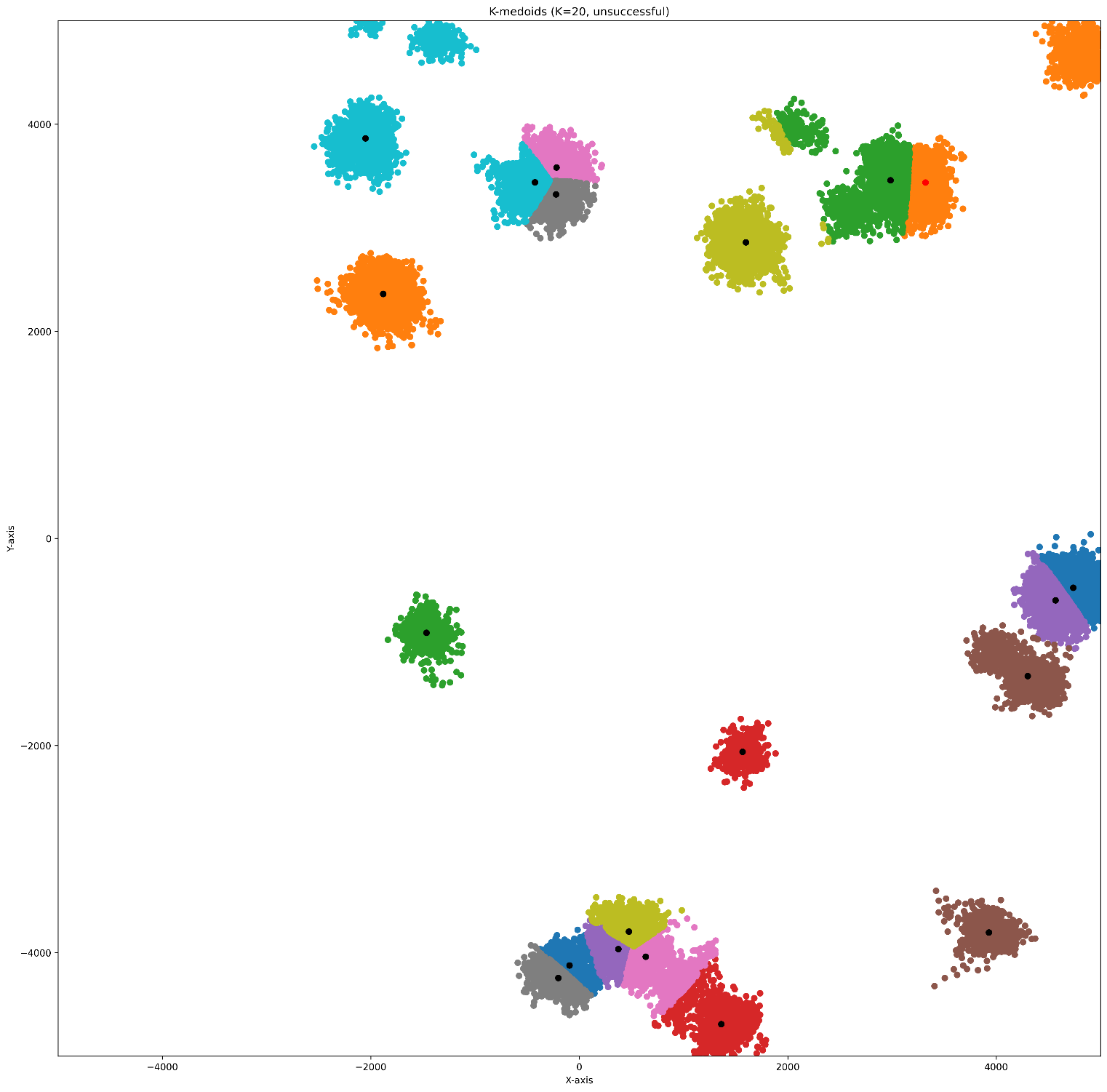
## Výsledky testovania a vizualizácia

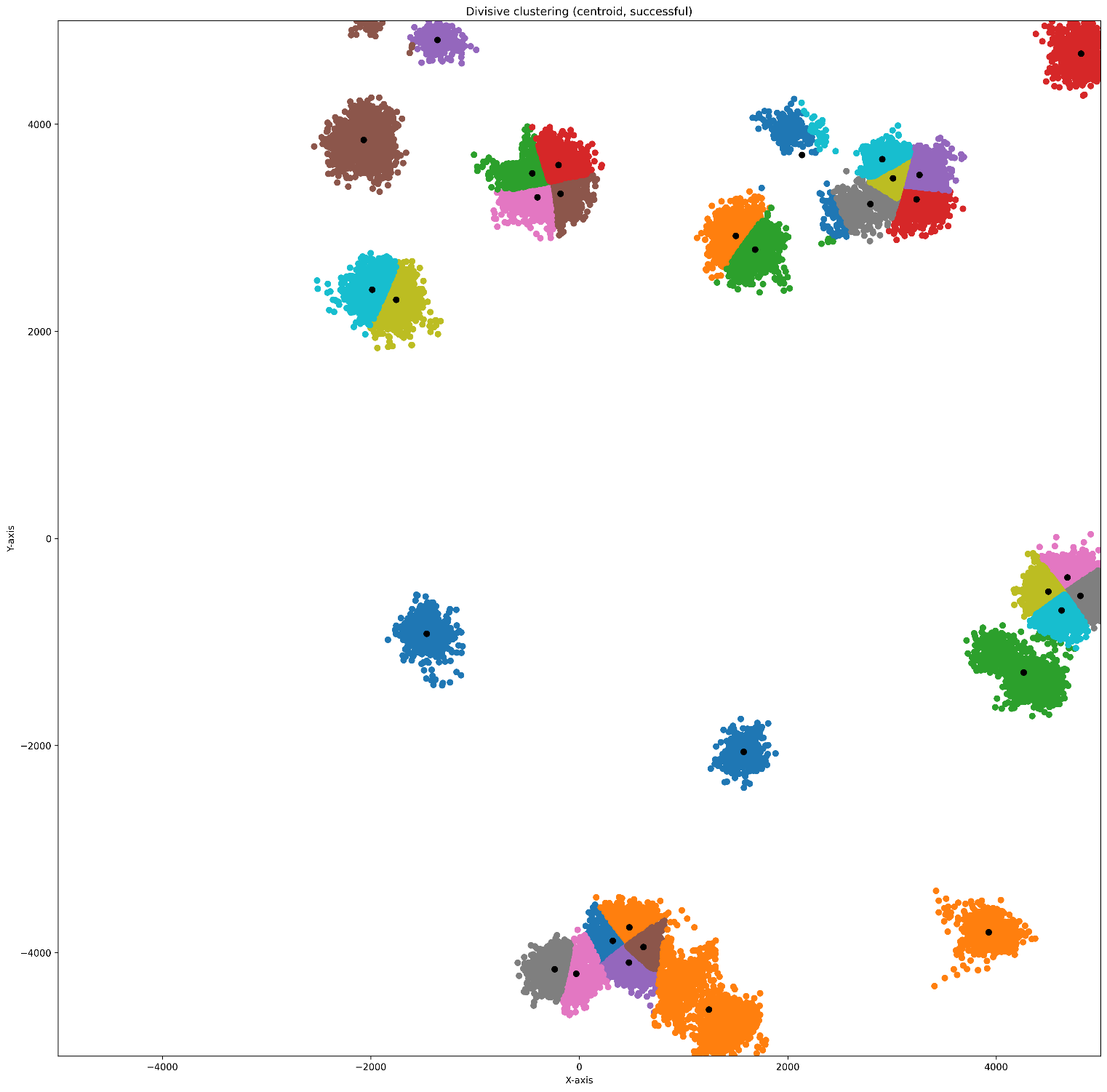
|  |  |  |  |
| --- | --- | --- | --- |
|  | **K-means (K=20)** | **K-medoids (K=20)** | **Divízne zhlukovanie** |
| Čas 1, s | 35.86085939 | 910.617033 | 5.324202061 |
| Čas 2, s | 17.24153161 | 749.2716079 | 3.245486021 |
| Čas 3, s | 37.66490602 | 907.4700212 | 4.215854645 |
| *Priemer* | 30.25576568 | 855.7862207 | 4.261847576 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **K-means (K=20)** | **K-medoids (K=20)** | **Divízne zhlukovanie** |
| Úspešnosť 1 | Áno | Nie | Áno |
| Úspešnosť 2 | Áno | Áno | Áno |
| Úspešnosť 3 | Nie | Nie | Áno |

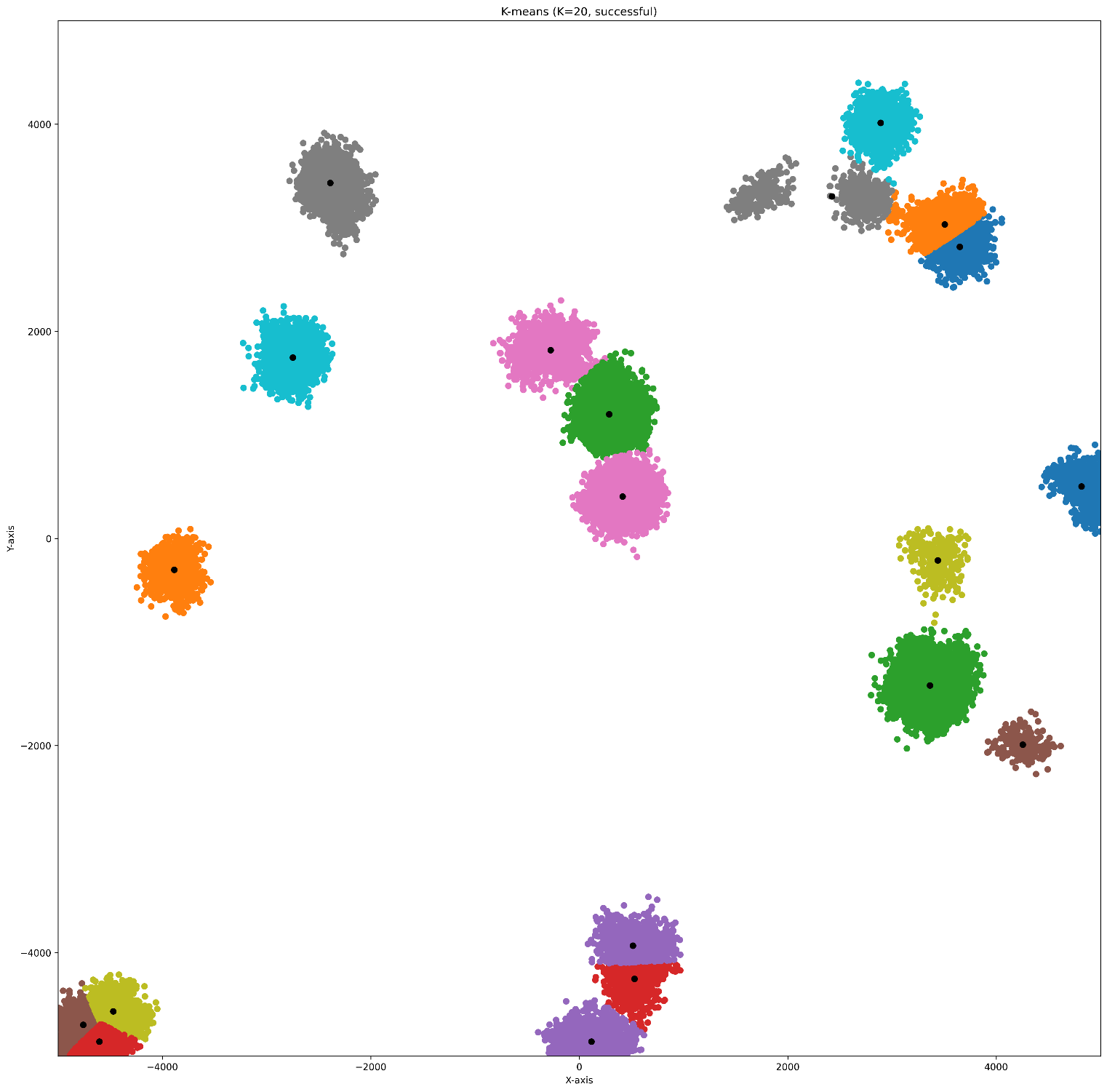
### 1. príklad

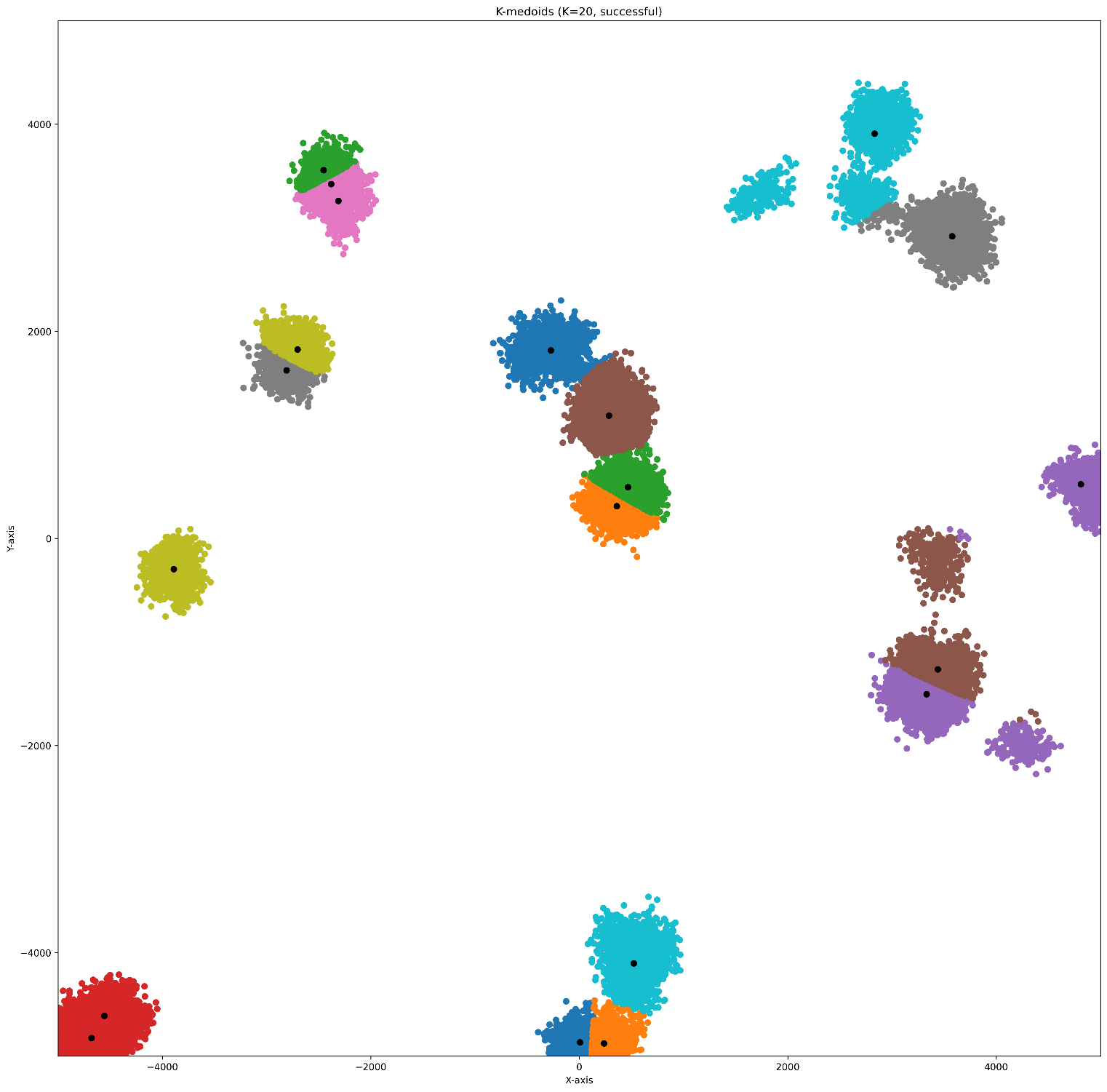


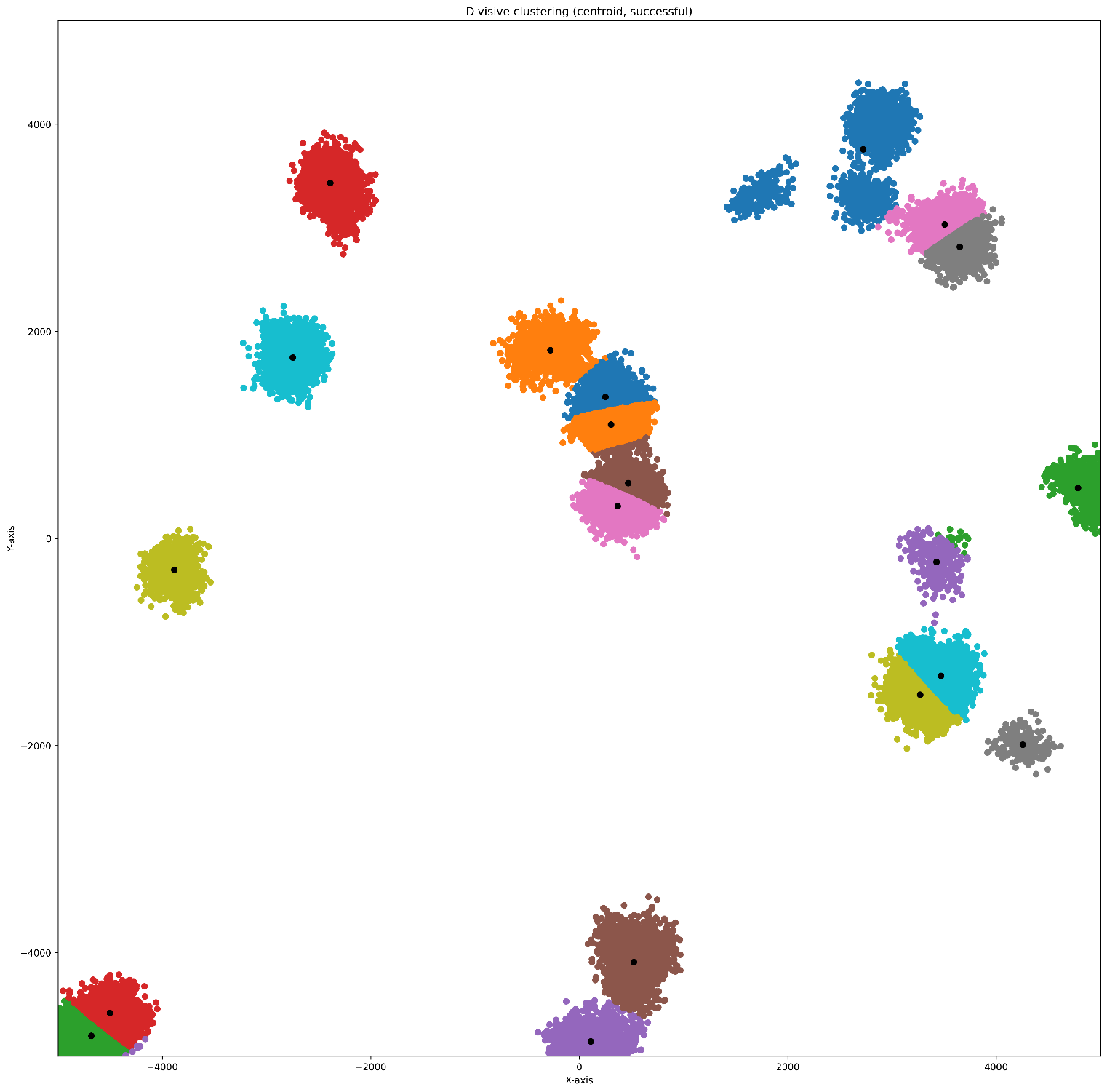




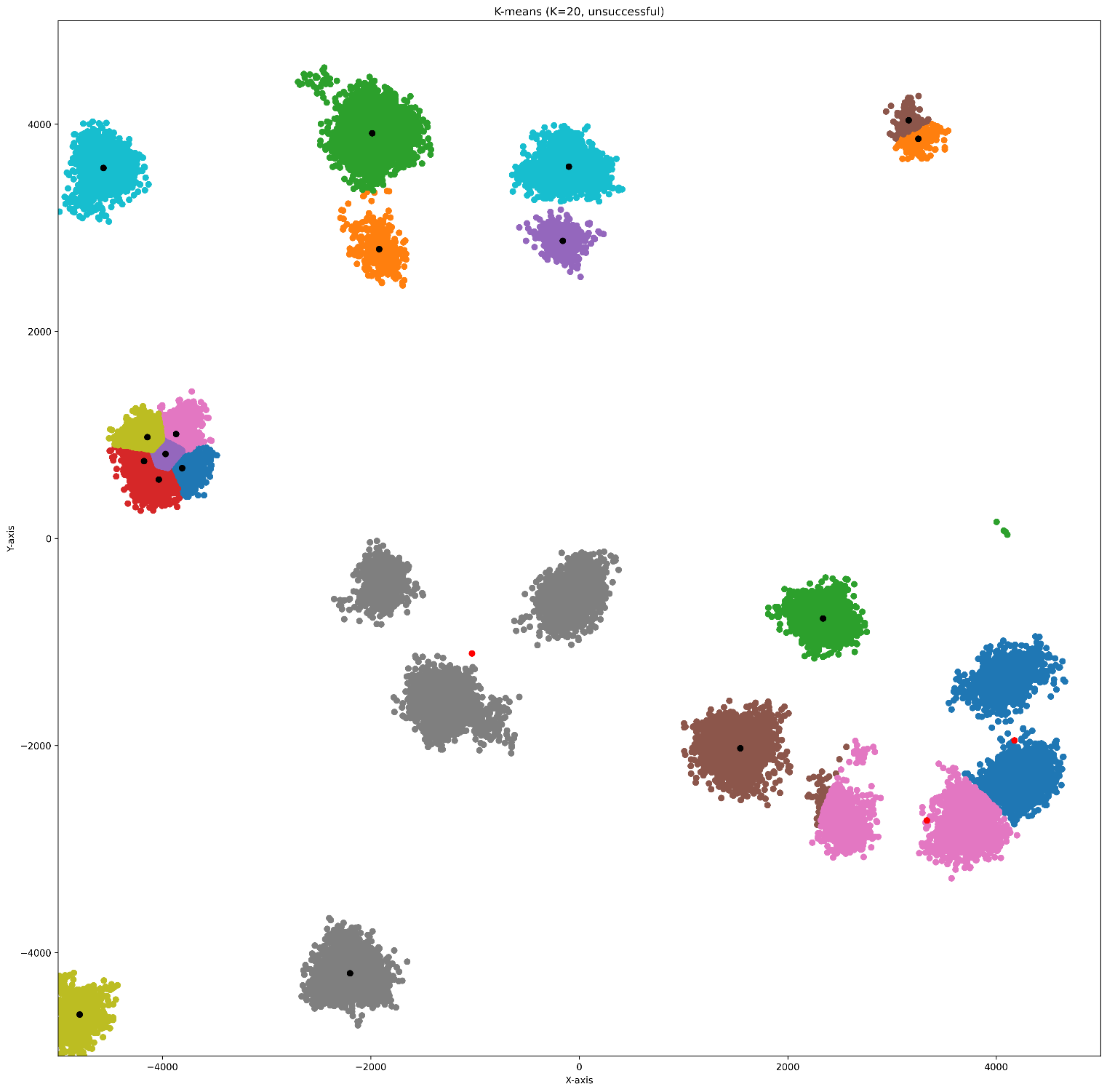
### 2. príklad

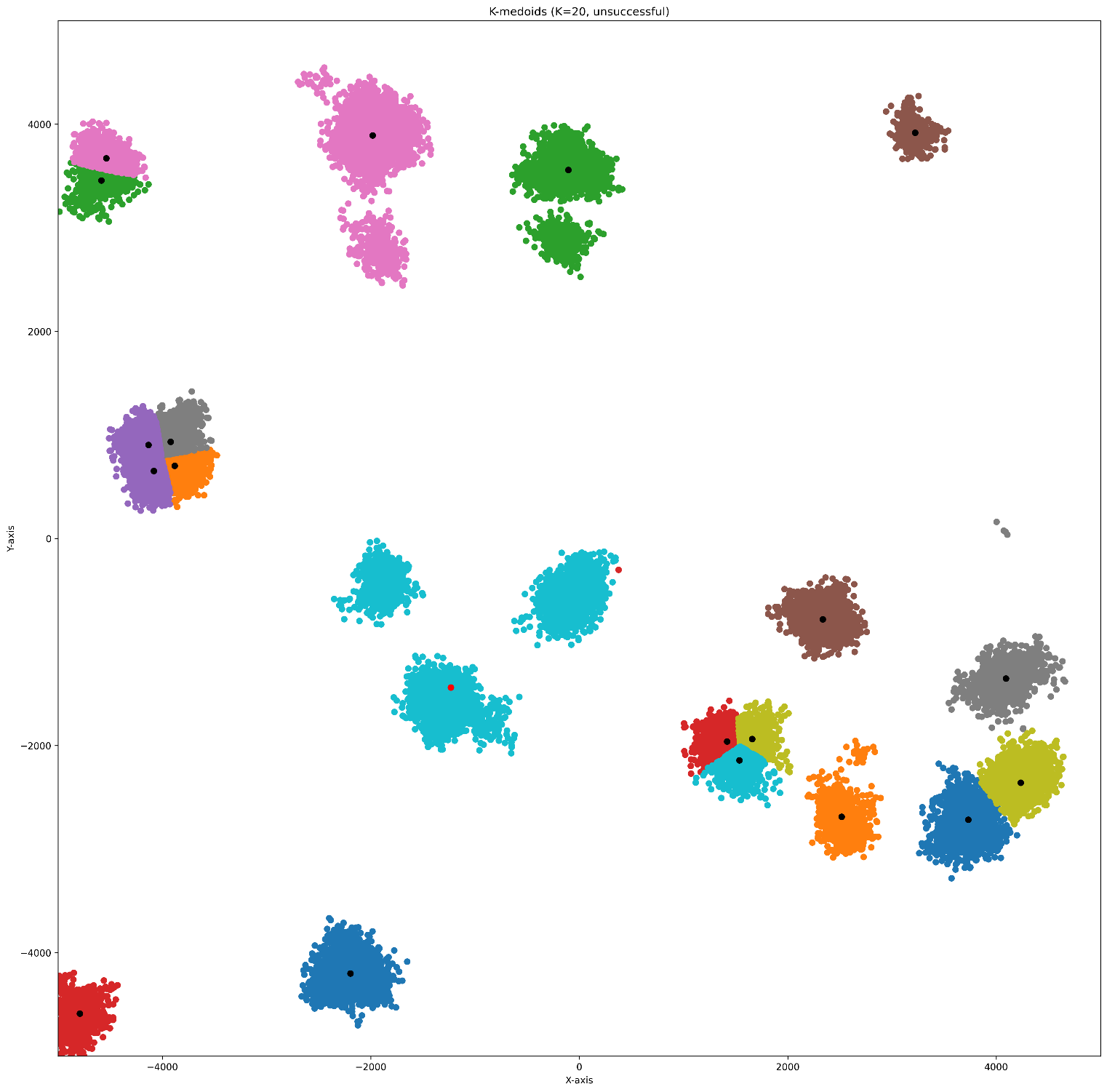


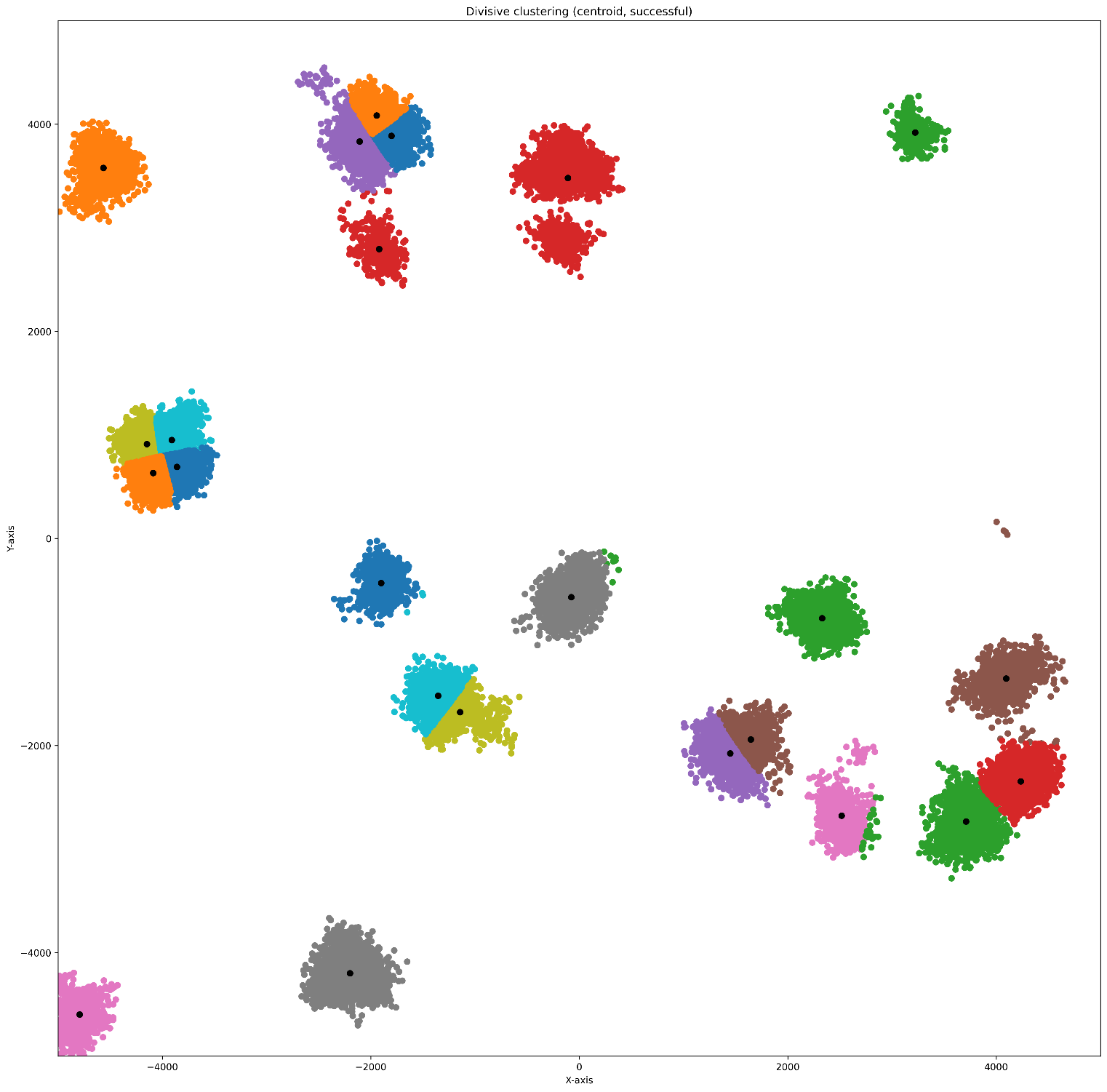




### 3. príklad







## Záver

Na záver možno povedať, že implementované zhlukovacie algoritmy, konkrétne K-means (K=20), K-medoids (K=20) a divízne zhlukovanie (pomocou K-means s K=2), vykazovali v danom kontexte rôznu mieru úspešnosti a výpočtovej efektívnosti.

K-means dosiahol úspech pri zhlukovaní 2 z 3 prípadov. Pri rozdelení údajov do zhlukov preukázal primeranú úroveň účinnosti. S priemerným časom spracovania 30.26 sekundy bol v porovnaní s K-medoids pomerne efektívny.

K-medoids bol úspešný v 1 z 3 prípadov, čo znamená nižšiu úspešnosť v porovnaní s K-means pre tieto tri datasety. Výpočtový čas pre K-medoids bol výrazne vysoký, v priemere 855.79 sekundy, kvôli spôsobu aktualizácie medoidov zhluku, čo prispelo k jeho dlhšiemu času behu.

Divízne zhlukovanie dôsledne dosiahlo úspešné zhlukovanie, pretože je štruktúrované tak, že úspešnosť je hlavná ukončovacia podmienka v tejto implementácii. Z hľadiska rýchlosti výrazne prekonal K-means aj K-medoids, pričom priemerný čas spracovania bol 4.26 sekundy, čo z neho robí najrýchlejší spomedzi testovaných algoritmov.