In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
pip install requests
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from reque
sts) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests)
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests)
(2024.2.2)
```

In [ ]:
```
import pandas as pd
import numpy as np
import requests
import matplotlib.pyplot as plt
from io import StringIO
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from sklearn.neighbors import NearestNeighbors
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances
```

In [ ]:
```
url = 'https://cs.joensuu.fi/sipu/datasets/D31.txt'
response = requests.get(url)
if response.status_code == 200:
    data = StringIO(response.text)
    data = pd.read_csv(data, delim_whitespace=True, header=None, names=['x','y','cluster'])
else:
    print(f"Error fetching data: {response.status_code}")
```

In [ ]:
```
d31_data = np.array(data[['x','y']])
```

In [ ]:
```
shapeddata_df = pd.read_csv("/content/drive/MyDrive/IE529_comp2/ShapedData.csv", header=None, names=['x','y'])
shapeddata_mat = np.array(shapeddata_df)
clustering_df = pd.read_csv("/content/drive/MyDrive/IE529_comp2/clustering.csv", header=None, names=['x','y'])
clustering_mat = np.array(clustering_df)
```

In [ ]:
```
def labelscal(data_points, centers):
    diff = data_points - centers.reshape(centers.shape[0], 1, centers.shape[1])
    dist = np.sqrt((diff**2).sum(axis=2))
    closest_pt = np.argmin(dist, axis=0)
    return closest_pt
```

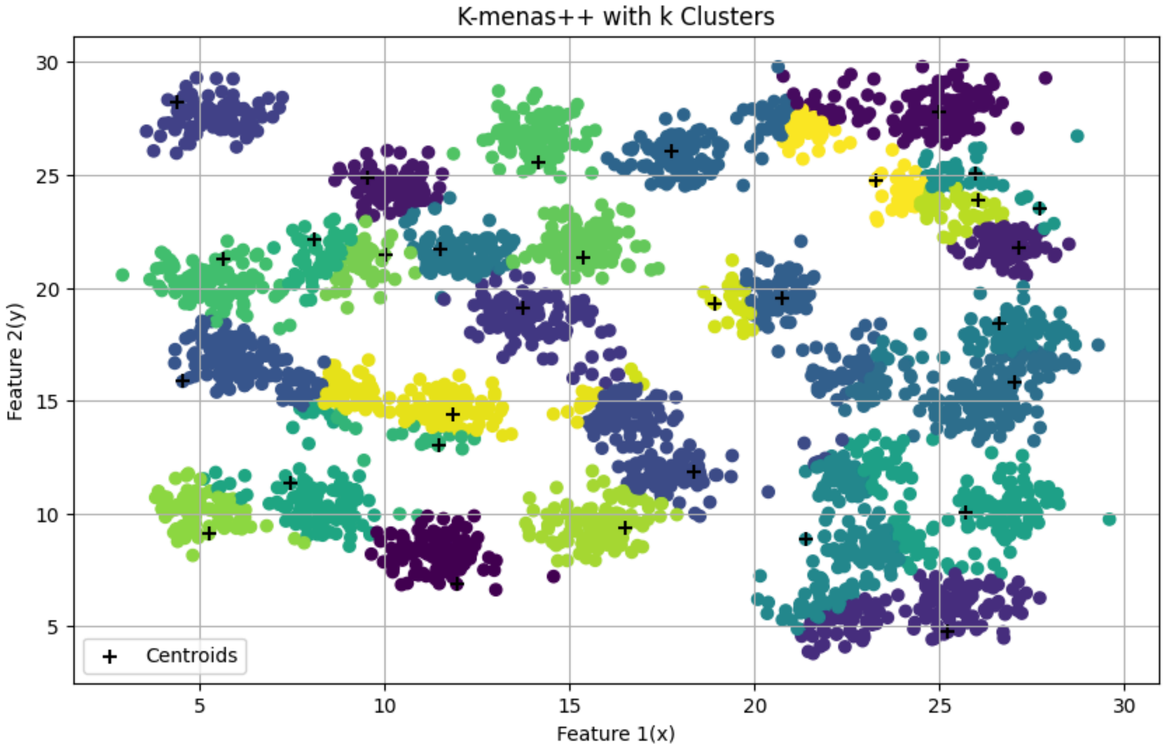
In [ ]:
```
def printgraph(data,label,center):
  plt.figure(figsize=(10, 6))
  plt.scatter(data[:, 0], data[:, 1], c=label, cmap='viridis', marker='o')
  plt.scatter(center[:, 0], center[:, 1], s=50, c='black', marker='+', label='Centroids')
  plt.title('K-means++ Parallel_pois with k Clusters')
  plt.xlabel('Feature 1(x)')
  plt.ylabel('Feature 2(y)')
  plt.legend()
  plt.grid(True)
  plt.show()
```

K-means + + seeding:

In [ ]:
```
def kmeans_plus_plus(data_points, k):
  K_centers = [data_points[np.random.randint(0, data_points.shape[0])]]
  for i in range(1, k):
      distances = np.min([np.sum((data_points - center)**2, axis=1) for center in K_centers], axis=0)
      #print(distances)
      probabilities = distances/np.sum(distances)
      #print(probabilities)
      new_center = data_points[np.random.choice(data_points.shape[0], p=probabilities)]
      K_centers.append(new_center)
  return np.array(K_centers)
```

In [ ]:

```
K_centroids_plus = kmeans_plus_plus(d31_data,31)
labels_plus = labelscal(d31_data,K_centroids_plus)
printgraph(d31_data,labels_plus,K_centroids_plus)
compute_clustering_cost(d31_data,K_centroids_plus)
```



Out[42]: 11264.509225410002

K-means ||:

In [ ]:

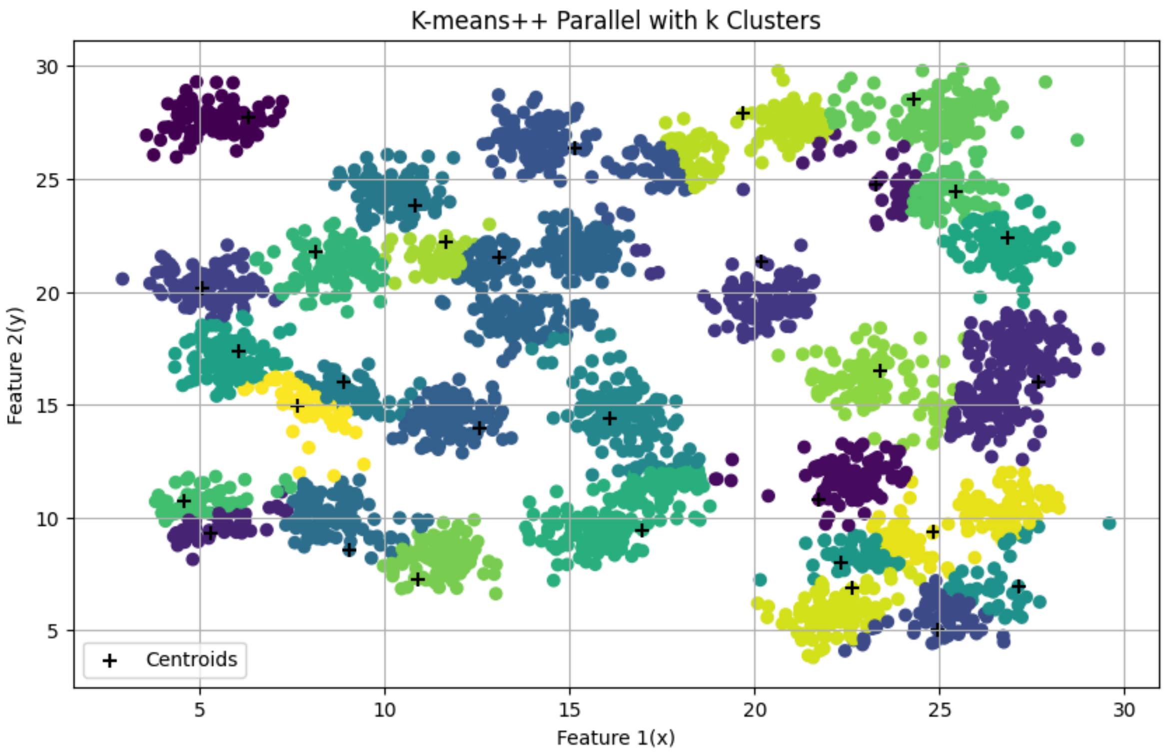
```
def kmeans_parallel_seeding(data_points, k, T):
  l = k*5
  K_centers = [data_points[np.random.randint(0, data_points.shape[0])]]
  for i in range(T):
      C_prime = []
      distances = np.min([np.linalg.norm(data_points - center, axis=1)**2 for center in K_centers], axis=0)
      total_cost = np.sum(distances)
      probabilities = l*distances/total_cost
      probabilities = probabilities/np.sum(probabilities)
      sampled_points = np.random.choice(data_points.shape[0], l, replace=False, p=probabilities)
      C_prime = data_points[sampled_points]
      K_centers.extend(C_prime)
  return np.array(K_centers)
```

In [ ]:

```
l_centers.shape
```

Out[31]: (201, 2)

In [ ]:
```
l_centers = kmeans_parallel_seeding(d31_data,31,10)
K_centroids_ll = kmeans_plus_plus(l_centers,31)
labels_ll = labelscal(d31_data,K_centroids_ll)
printgraph(d31_data,labels_ll,K_centroids_ll)
compute_clustering_cost(d31_data,K_centroids_ll)
```

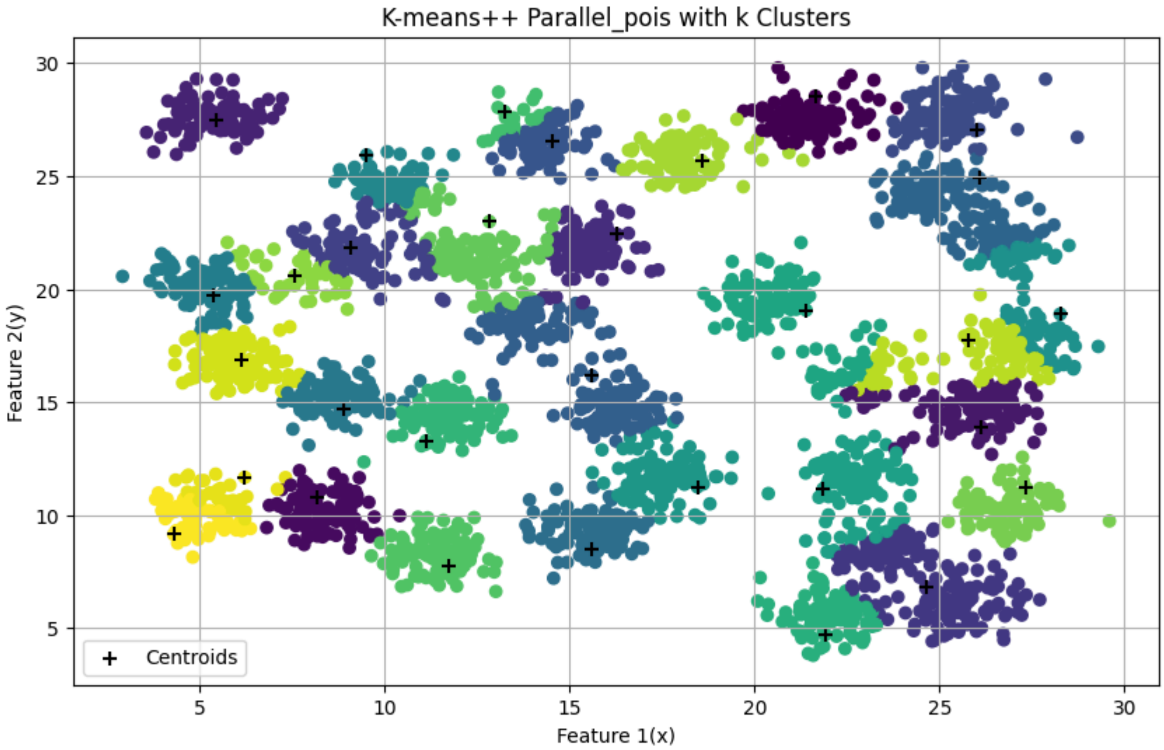


Out[46]: 8758.948287800002

K-means ||_pois Seeding:

In [ ]:
```
def kmeans_pois_seeding(data_points, k, T):
  l= 2*k
  K_centers = [data_points[np.random.randint(0, data_points.shape[0])]]
  for i in range(T):
      C_prime = []
      distances = np.min([np.linalg.norm(data_points - center, axis=1)**2 for center in K_centers], axis=0)
      total_cost = np.sum(distances)
      lambdas = l*distances/total_cost
      probabilities = 1-np.exp(-lambdas)
      probabilities = probabilities/np.sum(probabilities)
      sampled_points = np.random.choice(data_points.shape[0], l, replace=False, p=probabilities)
      C_prime = data_points[sampled_points]
      K_centers.extend(C_prime)
  return np.array(K_centers)
```

In [ ]:
```
l2_centers = kmeans_pois_seeding(d31_data,31,5)
K_centroids_ll2 = kmeans_plus_plus(l2_centers,31)
labels_ll2 = labelscal(d31_data,K_centroids_ll2)
printgraph(d31_data,labels_ll2,K_centroids_ll2)
compute_clustering_cost(d31_data,K_centroids_ll2)
```



Out[63]: 8331.66166229

In [ ]:
```
def compute_clustering_cost(data_points, centers):
    # Calculate distances from each center and store them in a list
    all_distances = [np.linalg.norm(data_points - center, axis=1)**2 for center in centers]
    # Find the minimum distance to any center for each data point
    min_distances = np.min(all_distances, axis=0)
    # Sum up all minimum distances to get the total cost
    total_cost = np.sum(min_distances)
    return total_cost

def calculate_cost(data, centers):
    distances = pairwise_distances(data, centers)
    min_distances = np.min(distances, axis=1)
    return min_distances.sum()
```

In [ ]:

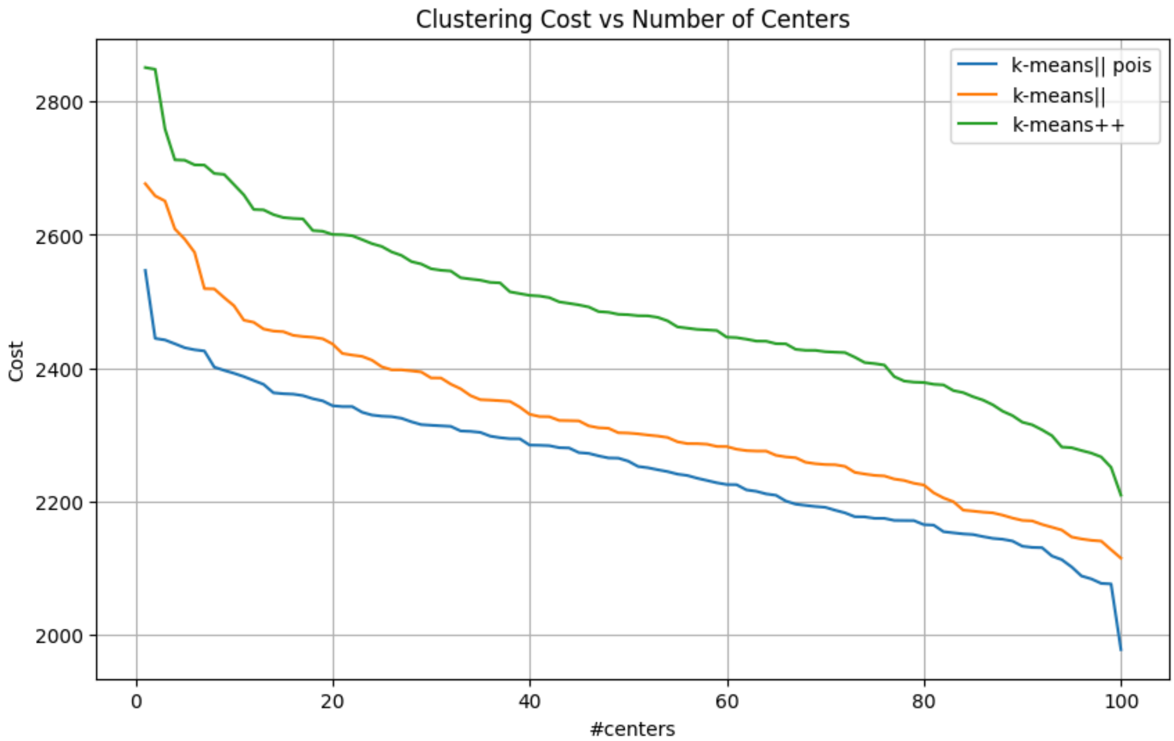
```
def run_experiment(data_points, K):
  costs_pl = []
  costs_ll = []
  costs_l2 = []
  for k in range(1, K+1):
    centers_plus_plus = kmeans_plus_plus(data_points, K)
    cost_plus_plus = compute_clustering_cost(data_points, centers_plus_plus)
    costs_pl.append(cost_plus_plus)
    #print(cost_plus_plus)

    centers_parallel = kmeans_parallel_seeding(data_points, K, 5)
    centers_parallel1 = kmeans_plus_plus(centers_parallel,K)
    cost_parallel = compute_clustering_cost(data_points, centers_parallel1)
    costs_ll.append(cost_parallel)
    #print(costs_ll)

    centers_pois = kmeans_pois_seeding(data_points, K, 5)
    centers_pois1 = kmeans_plus_plus(centers_pois,K)
    cost_pois = compute_clustering_cost(data_points, centers_pois1)
    costs_l2.append(cost_pois)
    #print(costs_l2)
  costs_pl =  sorted(costs_pl, reverse=True)
  costs_ll = sorted(costs_ll, reverse=True)
  costs_l2 = sorted(costs_l2, reverse=True)
  plt.figure(figsize=(10, 6))
  plt.plot(range(1,K+1), costs_pl, label='k-means|| pois')
  plt.plot(range(1,K+1), costs_ll, label='k-means||')
  plt.plot(range(1,K+1), costs_l2, label='k-means++')
  plt.xlabel('#centers')
  plt.ylabel('Cost')
  plt.legend()
  plt.grid(True)
  plt.title('Clustering Cost vs Number of Centers')
  plt.show()
```

In [ ]:

```
run_experiment(d31_data,100)
```



Step 1: Improved Approximation Guarantees for k-means++

- Enhancement of Theoretical Bounds: The paper demonstrates that the expected cost of the solution produced by k-means++ can now be bounded by 5(ln k + 2) times the optimal solution's cost. This improves the previous bound of 8(ln k + 2) provided by Arthur and Vassilvitskii (2007).
- Method of Improvement: This enhancement is achieved through a refined analysis of the expected cost of covered clusters, offering a tighter bound on these costs.

In [ ]:
```
cluster_means = data.groupby('cluster').mean()
opt_cluster_means=np.array(cluster_means)
```

In [ ]:
```
opt_distance = []
for i in range(1,32):
  dist = 0
  center = opt_cluster_means[i-1]
  temp=data[data['cluster']==i]
  temp = np.array(temp[['x','y']])
  for j in range(100):
    dist = dist + np.sqrt(np.sum((temp[j] - center)**2))
  opt_distance.append(dist)
```

In [ ]:
```
def simulate_kmeans_plus_plus(data_points, k, targ_clust):
    centers_sim = kmeans_plus_plus(data_points, k-1)
    new_center = kmeans_plus_plus(targ_clust, 1)[0]
    all_centers = np.vstack([centers_sim, new_center])
    return calculate_cost(data_points, all_centers), new_center
```

In [ ]:
```
targ_clust = np.array(data[data['cluster']==10][['x','y']])
costs = []
for i in range(1000):
    cost, new_center = simulate_kmeans_plus_plus(d31_data, 31, targ_clust)
    costs.append(cost)

expected_cost = np.mean(costs)
print("Expected Cost:", expected_cost/31)
print("Optimal Cost:",optimal_cost)
optimal_cost = calculate_cost(targ_clust, [np.mean(targ_clust, axis=0)])
print("5 * OPT1(Pi):", 5 * optimal_cost)
```

```
Expected Cost: 146.03756758286096
Optimal Cost: 93.17193581863616
5 * OPT1(Pi): 465.8596790931808
```