МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Машинное обучение»

Тема: Кластеризация (DBSCAN, OPTICS)

Студент гр. 6307 _____ Гарифуллин В.Ф.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2020

1. Загрузка данных

```
[1]  import pandas as pd
     import numpy as np
     data = pd.read_csv('CC GENERAL.csv').iloc[:,1:].dropna()
```

2. DBSCAN

1. Проведем кластеризацию методов k-средних

```
[3]  from sklearn.cluster import KMeans
     k_means = KMeans(init='k-means++', n_clusters=3, n_init=15)
     k_means.fit(data)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=15, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

2. Так как разные признаки лежат в разных шкалах, то стандартизируем данные

```
[4]  from sklearn import preprocessing
     data = np.array(data, dtype='float')
     min_max_scaler = preprocessing.StandardScaler()
     scaled_data = min_max_scaler.fit_transform(data)
```

3. Проведем кластеризацию методов DBSCAN при параметрах по умолчанию. Выведем метки кластеров, количество кластеров, а также процент наблюдений, которые кластеризовать не удалось

```
[5]  from sklearn.cluster import DBSCAN
     clustering = DBSCAN().fit(scaled_data)
     print(set(clustering.labels_))
     print(len(set(clustering.labels_)) - 1)
     print(list(clustering.labels_).count(-1) / len(list(clustering.labels_)))
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
36
0.7512737378415933
```

Метки кластеров от 0 до 35.
Всего их 36
-1 – не удалось кластеризовать.
75% наблюдений класетризовать не удалось

Опишите все параметры, которые принимает DBSCAN

eps – максимальное расстояние между двумя наблюдениями, при котором одно будет считаться соседним с другим.

min_samples – число наблюдений в окрестности точки, необходимое, чтобы эта точка считалась базовой. Сюда входит и сама точка.

metric – показывает, как вычисляется расстояние между экземплярами в массиве признаков.

metric_params – дополнительные параметры метрики

algorithm – алгоритм, который будет использоваться модулем NearestNeighbors для вычисления точечных расстояний и поиска ближайших соседей.

leaf_size – размер листа, передаваемый в BallTree или cKDTree.

p – степень в расстоянии Минковского, которое используется, для вычисления расстояния между точками.

n_jobs – число потоков.

4. Постройте график количества кластеров и процента не кластеризованных наблюдений в зависимости от максимальной рассматриваемой дистанции между наблюдениями. Минимальное значение количества точек образующих, кластер оставить по умолчанию

```
clusters_counts = []
fails_percents = []
eps_range = np.arange(0.1, 5, 0.5)
for eps in eps_range:
  clustering = DBSCAN(eps=eps).fit(scaled_data)
  clusters_counts.append(len(set(clustering.labels_)) - 1)
  fails_percents.append(list(clustering.labels_).count(-1) / len(list(clustering.labels_)) * 100)
```

```
import matplotlib.pyplot as plt
plt.plot(eps_range, clusters_counts)
plt.xlabel("Максимальная рассматриваемая дистанция между наблюдениями")
plt.ylabel("Количество кластеров")
```

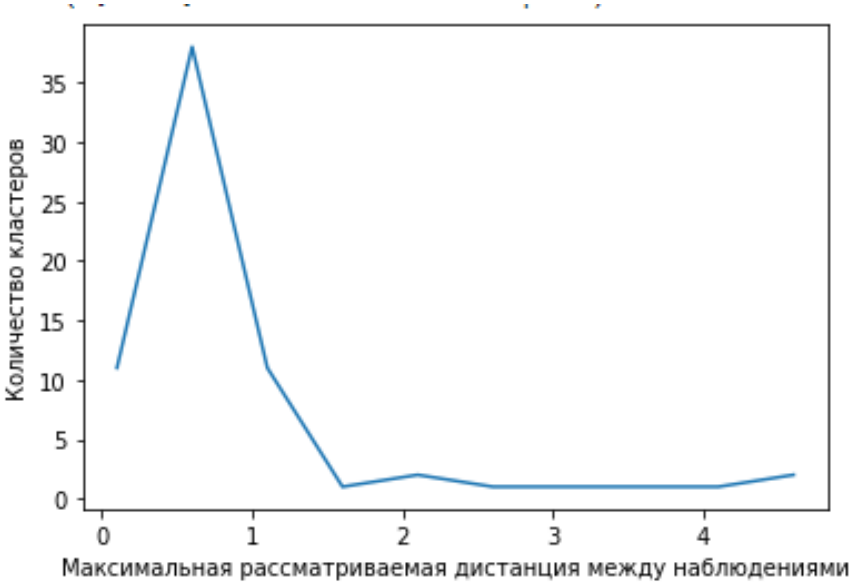

Рис. 1. График зависимости количества кластеров от максимальной рассматриваемой дистанции между наблюдениями

```
plt.plot(eps_range, fails_percents)
plt.xlabel("Максимальная рассматриваемая дистанция между наблюдениями")
plt.ylabel("Процент не кластеризованных наблюдений")
```

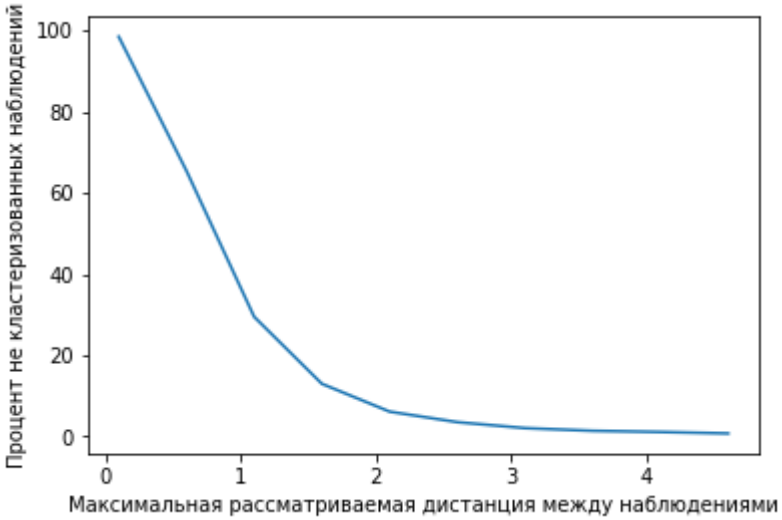

Рис. 2. График зависимости процента некластеризованных наблюдений от максимальной рассматриваемой дистанции между наблюдениями

5. Постройте график количества кластеров и процента не кластеризованных наблюдений в зависимости от минимального значения количества точек, образующих кластер. Максимальную рассматриваемую дистанцию между наблюдениями оставьте по умолчанию

```
clusters_counts = []
fails_percents = []
min_samples_range = np.arange(2, 10)
for min_samples in min_samples_range:
  clustering = DBSCAN(min_samples=min_samples).fit(scaled_data)
  clusters_counts.append(len(set(clustering.labels_)) - 1)
  fails_percents.append(list(clustering.labels_).count(-1) / len(list(clustering.labels_)) * 100)
```

```
plt.plot(min_samples_range, clusters_counts)
plt.xlabel("Минимальное количество точек, образующих кластер")
plt.ylabel("Количество кластеров")
```

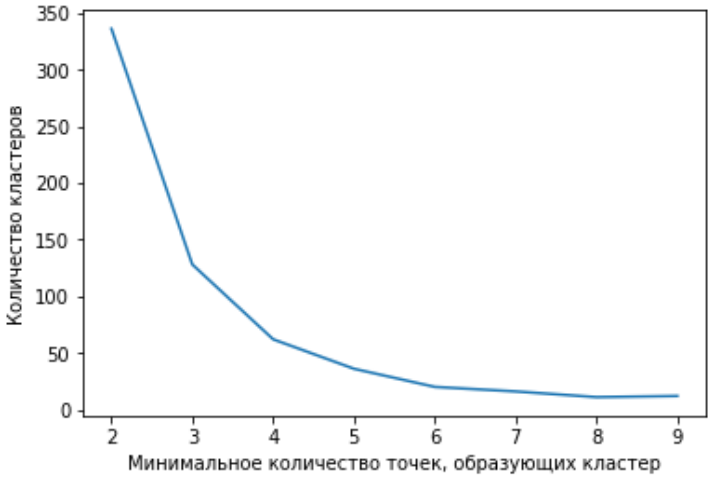

Рис. 3. График зависимости количества кластеров от минимального количества точек, образующих кластер

```
plt.plot(min_samples_range, fails_percents)
plt.xlabel("Минимальное количество точек, образующих кластер")
plt.ylabel("Процент не кластеризованных наблюдений")
```

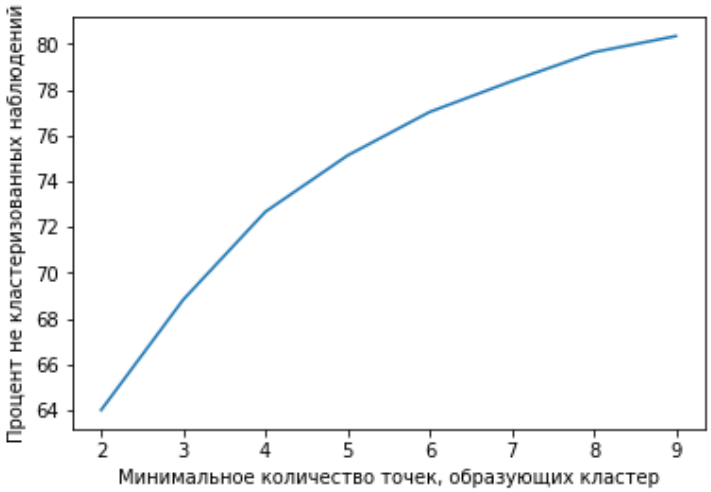

Рис. 4. График зависимости процента некластеризованных наблюдений от минимального количества точек, образующих кластер

6. Определите значения параметров, при котором количество кластеров получается от 5 до 7, и процент не кластеризованных наблюдений не превышает 12%.

```
def find_best():
  for eps in np.arange(0.5, 5, 0.5):
    for min_samples in np.arange(2, 10):
      clustering = DBSCAN(eps=eps, min_samples=min_samples).fit(scaled_data)
      clusters_count = len(set(clustering.labels_)) - 1
      fails_percent = list(clustering.labels_).count(-1) / len(list(clustering.labels_)) * 100
      if (clusters_count >= 5 and clusters_count <= 7 and fails_percent <= 12):
        print(eps, min_samples, clusters_count, fails_percent)
        return
```

```
find_best()
```

```
2.0 3 6 6.287633163501622
```

При eps = 2 и min_samples=3, количество кластеров = 6, процент не кластеризованных наблюдений = 6

7. Понизьте размерность данных до 2 при используя метод главных компонент. Визуализируйте результаты кластеризации, полученные в пункте 6 (метки должны быть получены на данных до уменьшения размерности)

```
[71] clustering = DBSCAN(eps=2, min_samples=3).fit(scaled_data)
```

```
[43] from sklearn.decomposition import PCA
     pca_data = PCA(n_components=2).fit_transform(data)
```

```
core_samples_mask = np.zeros_like(clustering.labels_, dtype=bool)
core_samples_mask[clustering.core_sample_indices_] = True
labels = clustering.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
                col = [1, 1, 1, 0.1]
    class_member_mask = (labels == k)

    xy = pca_data[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)

    xy = pca_data[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)


plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

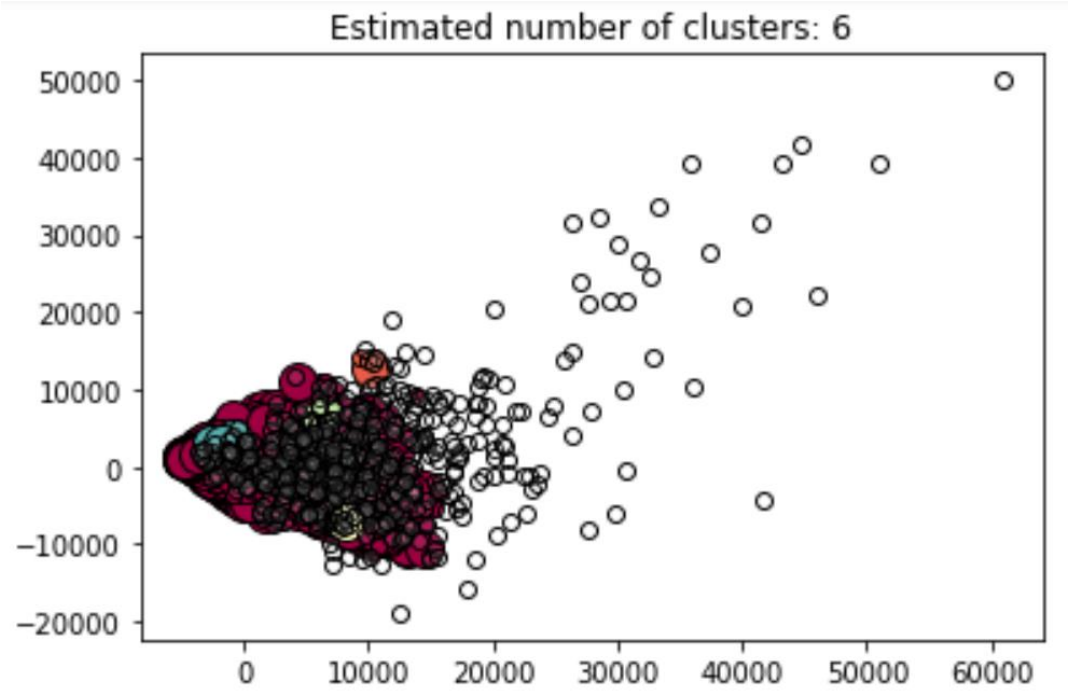

Рис. 5. Визуализация результатов кластеризации

3. OPTICS

1. Опишите параметры метода OPTICS, а также какими атрибутами он обладает

min_samples – число наблюдений рядом с точкой, чтобы она считалась ключевой точкой.

max_eps – максимальное расстояние между двумя наблюдениями, чтобы одно считалось соседним с другим

metric – показывает, как вычисляется расстояние между экземплярами в массиве признаков.

p – степень в расстоянии Минковского, которое используется, для вычисления расстояния между точками.

metric_params – дополнительные параметры метрики

cluster_method – метод используемый для извлечения кластеров

eps = max_eps

xi = определяет минимальную крутизну на графике достижимости, которая является границей кластера.

predecessor_correction – коррекция кластеров. Обычно имеет минимальный эффект.

min_cluster_size – минимальное число наблюдений в кластере, выраженное в виде абсолютного числа или доли от количества наблюдений.

algorithm – алгоритм, который будет использоваться для вычисления ближайших соседей.

leaf_size – размер листа, передаваемый в BallTree или cKDTree.

n_jobs – число потоков.

labels_ – метки кластеров.

reachability_ – достижимые расстояния.

ordering_ – список индексов в порядке кластеров.

core_distances_ – расстояние, на котором каждое наблюдение становится ключевой точкой.

predecessor_ – точка из которой данное наблюдение было достигнуто.

cluster_hierarchy_ – список кластеров, со всеми включенными индексами.

2. Найдите такие параметры метода OPTICS (*max_eps *и min_samples) при которых, чтобы получить результаты близкие к результатам DBSCAN из пункта 6.

```
[40] from sklearn.cluster import OPTICS
```

```
[106] def find_best_optics():
        for max_eps in np.arange(0.5, 3, 0.5):
          for min_samples in np.arange(2, 5):
            clustering = OPTICS(max_eps=max_eps, min_samples=min_samples, cluster_method='dbscan').fit(scaled_data)
            clusters_count = len(set(clustering.labels_)) - 1
            fails_percent = list(clustering.labels_).count(-1) / len(list(clustering.labels_)) * 100
            if (clusters_count >= 5 and clusters_count <= 7 and fails_percent <= 12):
              print(max_eps, min_samples, clusters_count, fails_percent)
              return
```

```
[107] find_best_optics()

      2.0 3 6 6.310792033348772
```

При eps = 2 и min_samples=3, количество кластеров = 6, процент не кластеризованных наблюдений = 6. Без изменения параметра cluster_method поиск не давал результата.

В чем отличия метода OPTICS от метода DBSCAN?

DBSCAN алгоритм кластеризации, основанной на плотности — если дан набор точек в некотором пространстве, алгоритм группирует вместе точки, которые тесно расположены (точки со многими близкими соседями), помечая как выбросы точки, которые находятся одиноко в областях с малой плотностью (ближайшие соседи которых лежат далеко). Однако у DBSCAN присутствует проблема обнаружения содержательных кластеров в данных, имеющих различные плотности. В OPTICS для предотвращения этого для каждой точки запоминается специальное расстояние, представляющее плотность, которую следует принять для кластера, чтобы точки принадлежали одному кластеру

3. Визуализируйте полученный результат, а также постройте график достижимости (reachable plot)

```
clust = OPTICS(max_eps=2, min_samples=3, cluster_method='dbscan').fit(scaled_data)
```

```
from sklearn.cluster import cluster_optics_dbscan
import matplotlib.gridspec as gridspec
labels_050 = cluster_optics_dbscan(reachability=clust.reachability_,
                                   core_distances=clust.core_distances_,
                                   ordering=clust.ordering_, eps=0.5)
labels_200 = cluster_optics_dbscan(reachability=clust.reachability_,
                                   core_distances=clust.core_distances_,
                                   ordering=clust.ordering_, eps=2)

X = pca_data
space = np.arange(len(X))
reachability = clust.reachability_[clust.ordering_]
labels = clust.labels_[clust.ordering_]

plt.figure(figsize=(10, 7))
G = gridspec.GridSpec(2, 3)
ax1 = plt.subplot(G[0, :])
ax2 = plt.subplot(G[1, 0])
ax3 = plt.subplot(G[1, 1])
ax4 = plt.subplot(G[1, 2])
```

```
# Reachability plot
colors = ['g.', 'r.', 'b.', 'y.', 'c.']
for klass, color in zip(range(0, 5), colors):
    Xk = space[labels == klass]
    Rk = reachability[labels == klass]
    ax1.plot(Xk, Rk, color, alpha=0.3)
ax1.plot(space[labels == -1], reachability[labels == -1], 'k.', alpha=0.3)
ax1.plot(space, np.full_like(space, 2., dtype=float), 'k-', alpha=0.5)
ax1.plot(space, np.full_like(space, 0.5, dtype=float), 'k-.', alpha=0.5)
ax1.set_ylabel('Reachability (epsilon distance)')
ax1.set_title('Reachability Plot')

# OPTICS
colors = ['g.', 'r.', 'b.', 'y.', 'c.']
for klass, color in zip(range(0, 5), colors):
    Xk = X[clust.labels_ == klass]
    ax2.plot(Xk[:, 0], Xk[:, 1], color, alpha=0.3)
ax2.plot(X[clust.labels_ == -1, 0], X[clust.labels_ == -1, 1], 'k+', alpha=0.1)
ax2.set_title('Automatic Clustering\nOPTICS')

# DBSCAN at 0.5
colors = ['g', 'greenyellow', 'olive', 'r', 'b', 'c']
for klass, color in zip(range(0, 6), colors):
    Xk = X[labels_050 == klass]
    ax3.plot(Xk[:, 0], Xk[:, 1], color, alpha=0.3, marker='.')
ax3.plot(X[labels_050 == -1, 0], X[labels_050 == -1, 1], 'k+', alpha=0.1)
ax3.set_title('Clustering at 0.5 epsilon cut\nDBSCAN')

# DBSCAN at 2.
colors = ['g.', 'm.', 'y.', 'c.']
for klass, color in zip(range(0, 4), colors):
    Xk = X[labels_200 == klass]
    ax4.plot(Xk[:, 0], Xk[:, 1], color, alpha=0.3)
ax4.plot(X[labels_200 == -1, 0], X[labels_200 == -1, 1], 'k+', alpha=0.1)
ax4.set_title('Clustering at 2.0 epsilon cut\nDBSCAN')

plt.tight_layout()
plt.show()
```

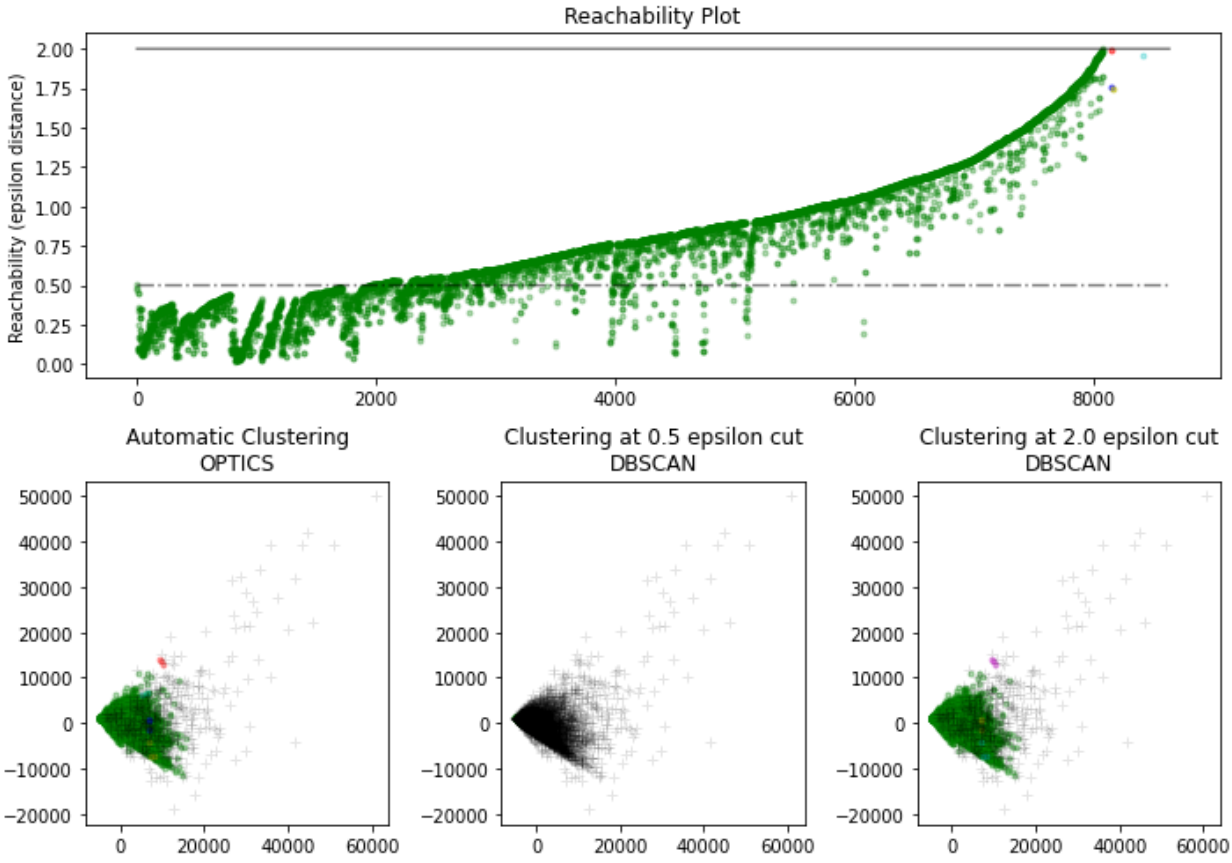

Рис. 6. График достижимости

4. Исследуйте работу метода OPTICS с использованием различных метрик (выберите не менее 5 метрик)

```
[132] clustering = OPTICS(max_eps=2, min_samples=3, cluster_method='dbscan', metric='cityblock').fit(scaled_data)
      clusters_count = len(set(clustering.labels_)) - 1
      fails_percent = list(clustering.labels_).count(-1) / len(list(clustering.labels_)) * 100
      print(clusters_count, fails_percent)

      55 39.49745252431681
```

```
[133] clustering = OPTICS(max_eps=2, min_samples=3, cluster_method='dbscan', metric='cosine').fit(scaled_data)
      clusters_count = len(set(clustering.labels_)) - 1
      fails_percent = list(clustering.labels_).count(-1) / len(list(clustering.labels_)) * 100
      print(clusters_count, fails_percent)

      0 0.0
```

```
[134] clustering = OPTICS(max_eps=2, min_samples=3, cluster_method='dbscan', metric='euclidean').fit(scaled_data)
      clusters_count = len(set(clustering.labels_)) - 1
      fails_percent = list(clustering.labels_).count(-1) / len(list(clustering.labels_)) * 100
      print(clusters_count, fails_percent)

      6 6.310792033348772
```

```
clustering = OPTICS(max_eps=2, min_samples=3, cluster_method='dbscan', metric='l1').fit(scaled_data)
clusters_count = len(set(clustering.labels_)) - 1
fails_percent = list(clustering.labels_).count(-1) / len(list(clustering.labels_)) * 100
print(clusters_count, fails_percent)
```

```
55 39.49745252431681
```

```
clustering = OPTICS(max_eps=2, min_samples=3, cluster_method='dbscan', metric='l2').fit(scaled_data)
clusters_count = len(set(clustering.labels_)) - 1
fails_percent = list(clustering.labels_).count(-1) / len(list(clustering.labels_)) * 100
print(clusters_count, fails_percent)
```

```
6 6.310792033348772
```

```
clustering = OPTICS(max_eps=2, min_samples=3, cluster_method='dbscan', metric='manhattan').fit(scaled_data)
clusters_count = len(set(clustering.labels_)) - 1
fails_percent = list(clustering.labels_).count(-1) / len(list(clustering.labels_)) * 100
print(clusters_count, fails_percent)
```

```
55 39.49745252431681
```

Метрики cityblock, l1, manhattan дали одинаковые результаты. Также одинаковые результаты дали метрики euclidean и l2. При этом метрика cosine не смогла разделить данные на кластеры. Скорее всего это связано с тем, что под каждую метрику параметры нужно подбирать отдельно.

4. ВЫВОДЫ

В ходе проделанной работы изучены и применены методы кластериации DBSCAN и OPTICS. Исследовано влияние параметров методов на результаты кластеризации. Произведена визуализация результатов кластеризации.