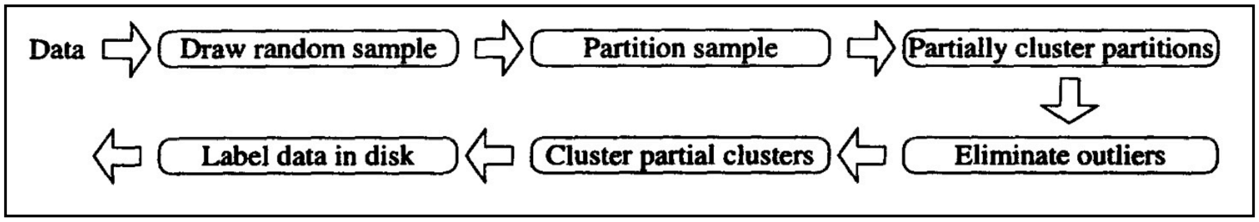
The CURE Algorithm for Clustering (Sec. 7.4)
Data Mining CSC 84040 Final Project
Craig Chosney

1. A description of the problem to be solved

The most simple and widely used algorithm for clustering is the K-Means algorithm, which works by partitioning a given set of objects into *k* different clusters by converging them to a local minimum. The K-Means Algorithm performs well on data sets in which data is aligned to axes but it is very susceptible to outliers. The cluster quality reduces drastically when the clusters are oddly shaped because K-Means favors spherical shapes and does not correctly identify the other shapes [1]. In summary, traditional clustering algorithms favor clusters with spherical shapes and similar sizes, or are very fragile in the presence of outliers. The CURE algorithm was proposed by Guha, Rastogi, and Shim [2] in 2001 to be more robust to outliers and to identify clusters having non-spherical shapes and wide variances in size.

2. A description of the algorithm

The CURE (Clustering Using Representatives) algorithm assumes Euclidean space. It uses collections of representative points to represent clusters rather than centroids. It works by first taking a small sample of the data and clustering it in main memory. Next, a small set of points from each cluster is selected to be “representative” points. These points are chosen as far from one another as possible. Finally, each of the representative points are moved a fixed fraction of the distance between its location and the centroid of its cluster. This step requires a Euclidean space, since otherwise there might not be any notion of a line between two points. These first three steps are the algorithm’s initialization. To complete the algorithm we then merge two clusters if they have a pair of representative points, one from each cluster, that are sufficiently close. The distance that defines “close” can be selected by the algorithm’s user. This merging step may repeat until there are no more sufficiently close clusters. Finally, each point *p* is brought from secondary storage and compared with the “representative” points. Each *p* is assigned to the cluster of the representative point that is closest to *p* [4].



Overview of CURE [2]

3. An analysis of the algorithm's performance:
- Why does it work?

The CURE algorithm works because it utilizes multiple representative points for each cluster that are generated by selecting well scattered points from the cluster and shrinking them toward the center of the cluster by a specified fraction. This enables CURE to adjust well to the geometry of clusters having non-spherical shapes and wide variances in size. To handle large databases, CURE employs a combination of random sampling and partitioning that allows it to handle large data sets efficiently. Random sampling, coupled with outlier handling techniques, also makes it possible for CURE to filter outliers contained its the data set effectively. Furthermore, the labeling algorithm in CURE uses multiple random representative points for each cluster to assign data points on a disk. This enables it to correctly label points even when the shapes of clusters are non-spherical and the sizes of the clusters vary [2].

- What is its cost?

Time complexity (best-case) for the CURE algorithm is $O(n^2)$ when the dimensionality of data points is small. When the dimensionality of data points is larger the time complexity (worst-case) is $O(n^2 \log n)$.

The space complexity of the CURE algorithm is $O(n)$ since both the heap and the $k-d$ tree require linear space.

- Pros and cons?

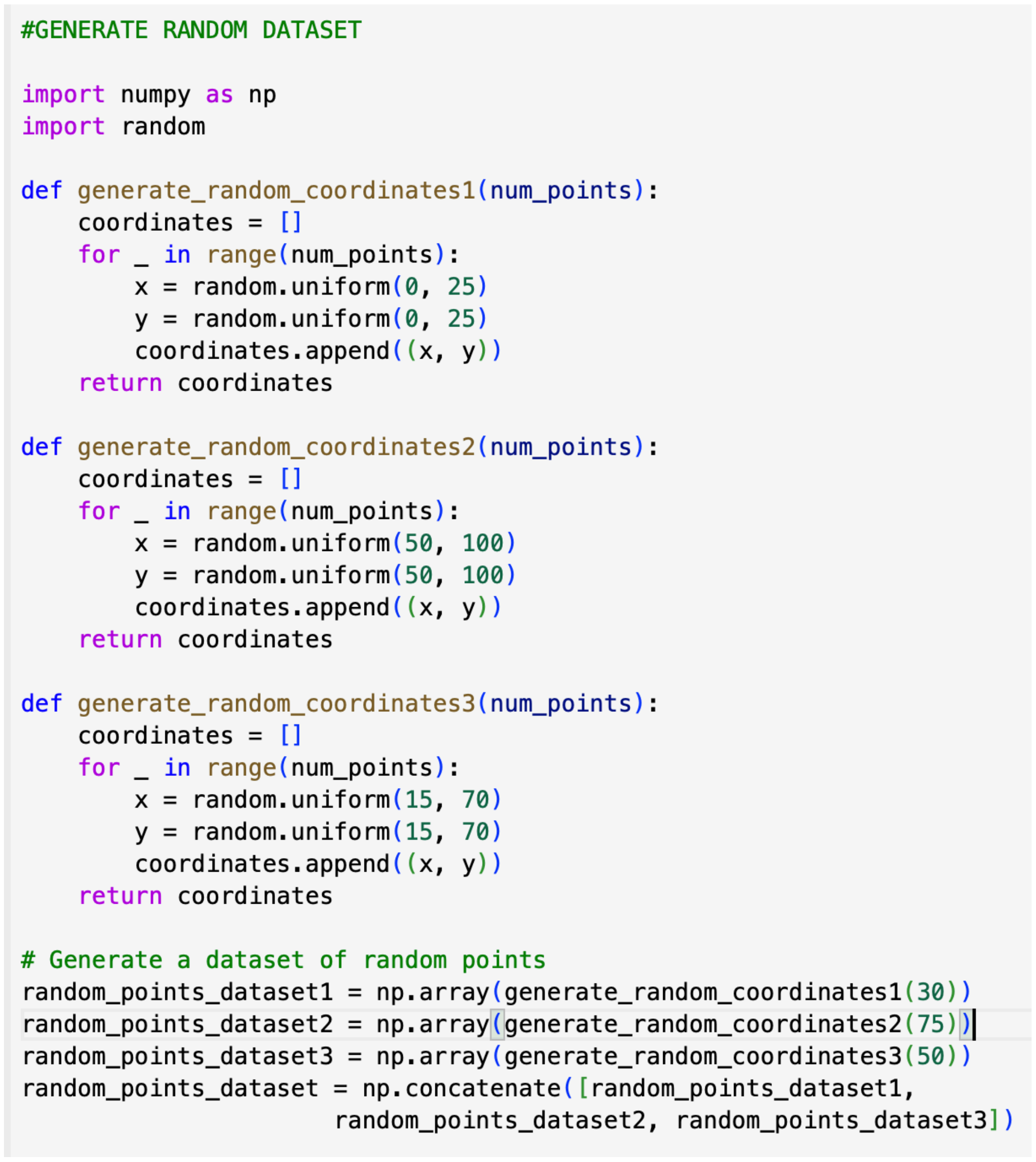
Pros of the CURE algorithm:
- Handling of Arbitrary Shapes — CURE is capable of identifying clusters of arbitrary shapes and sizes.
- Outlier Robustness — by considering representative points, CURE minimizes the impact of outliers on the final clustering result.
- Scalability — by using a random sampling technique to identify representative points, CURE can be more efficient for use on larger datasets.
- Memory Efficiency — as CURE operates on representative points, it doesn’t require storing the entire dataset in memory during clustering, thus making it more memory-efficient compared to some other clustering algorithms.
- Hierarchical Clustering — CURE’s hierarchical nature allows users to explore different levels of granularity in clustering results. This hierarchical structure enables the algorithm to capture clusters at various scales, thus giving users a more nuanced understanding of the data.

Cons of the CURE algorithm:
- Parameter Sensitivity — because CURE requires the specification of certain parameters, such as the number of clusters or the fraction of points to use as

representatives, the subjective selection of these parameters can impacts the quality of the clustering results.

- Computationally Intensive — CURE can be computationally intensive, especially for very large datasets. The processing of representative points and performing hierarchical clustering can require significant computational resources.
- Difficulty in Determining the Number of Clusters — Determining the optimal number of clusters can be challenging with CURE. Choosing the right number of clusters can be subjective and problem-dependent.
- Interpretability — Hierarchical clustering may not always provide straightforward interpretations. Understanding and visualizing the hierarchical structure may be challenging, making it harder for users to interpret the results.

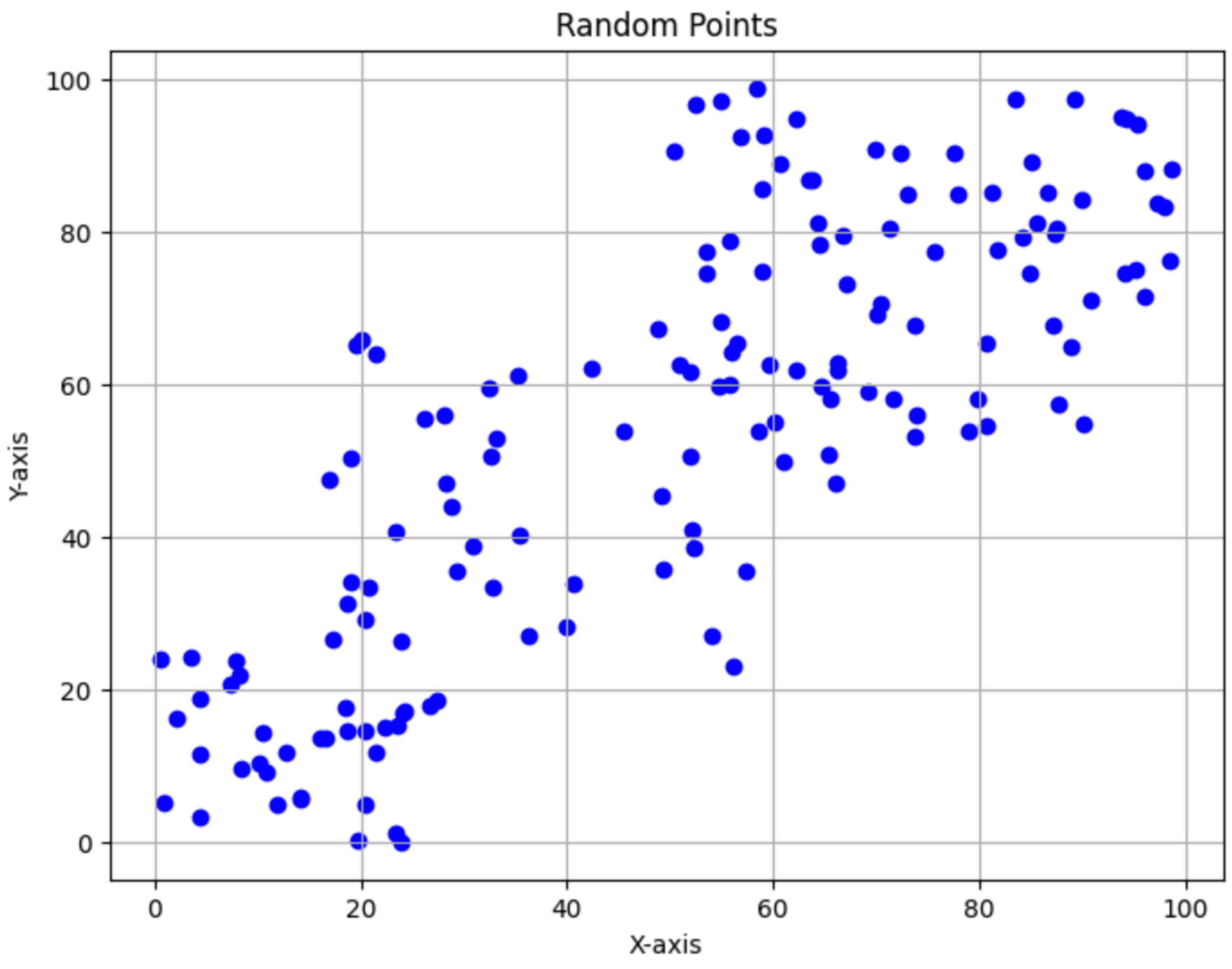
4. A concrete example of how the algorithm works:
- Choose a suitable dataset or generate one



```python
#GENERATE RANDOM DATASET

import numpy as np
import random

def generate_random_coordinates1(num_points):
    coordinates = []
    for _ in range(num_points):
        x = random.uniform(0, 25)
        y = random.uniform(0, 25)
        coordinates.append((x, y))
    return coordinates

def generate_random_coordinates2(num_points):
    coordinates = []
    for _ in range(num_points):
        x = random.uniform(50, 100)
        y = random.uniform(50, 100)
        coordinates.append((x, y))
    return coordinates

def generate_random_coordinates3(num_points):
    coordinates = []
    for _ in range(num_points):
        x = random.uniform(15, 70)
        y = random.uniform(15, 70)
        coordinates.append((x, y))
    return coordinates

# Generate a dataset of random points
random_points_dataset1 = np.array(generate_random_coordinates1(30))
random_points_dataset2 = np.array(generate_random_coordinates2(75))
random_points_dataset3 = np.array(generate_random_coordinates3(50))
random_points_dataset = np.concatenate([random_points_dataset1,
                     random_points_dataset2, random_points_dataset3])
```

```
[2]  !pip install matplotlib
     import matplotlib.pyplot as plt
```

```
[158] # Extract x and y coordinates from the dataset for plotting
      x_coords = [point[0] for point in random_points_dataset]
      y_coords = [point[1] for point in random_points_dataset]

      # Create a scatter plot of the random points
      plt.figure(figsize=(8, 6))
      plt.scatter(x_coords, y_coords, color='blue')
      plt.title('Random Points')
      plt.xlabel('X-axis')
      plt.ylabel('Y-axis')
      plt.grid(True)
      plt.show()
```



- *Implement the algorithm*

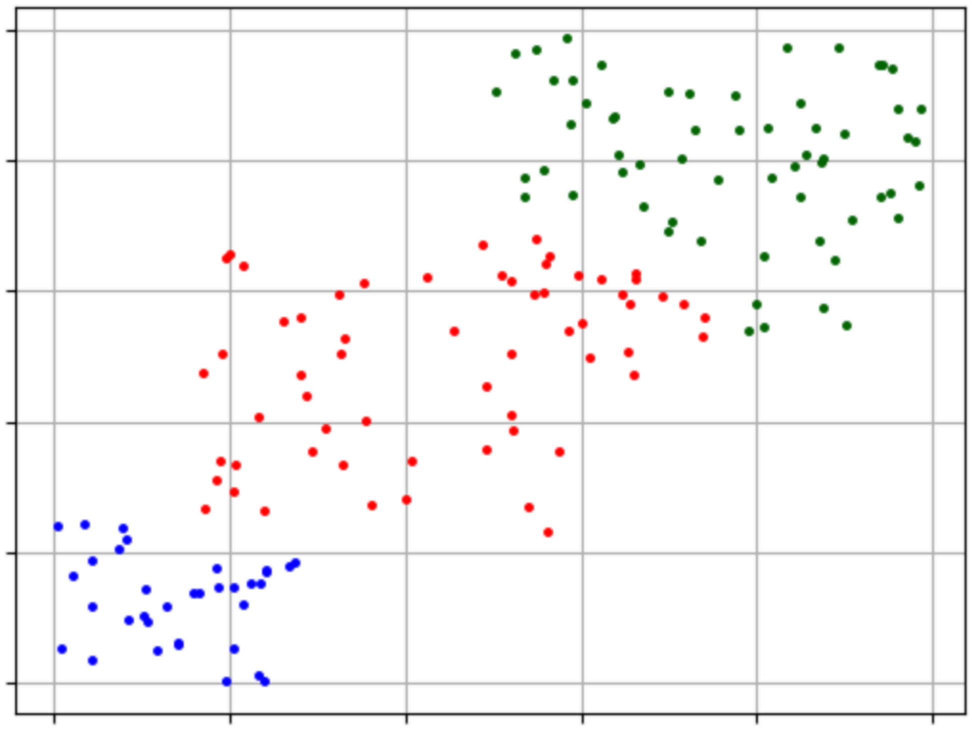
```
!pip install pyclustering
```

```
[142] from pyclustering.cluster import cluster_visualizer;
      from pyclustering.cluster.cure import cure;
      from pyclustering.utils import read_sample;
      from pyclustering.samples.definitions import FCPS_SAMPLES;
```

```
[159] # Allocated 3 clusters here

      cure_instance = cure(random_points_dataset, 3)
      cure_instance.process()
      clusters = cure_instance.get_clusters()
```

```
# Visualize allocated clusters

visualizer = cluster_visualizer()
visualizer.append_clusters(clusters, random_points_dataset)
visualizer.show()
```

- *Show the results obtained from the algorithm*



REFERENCES

1. Bharadwaj, Prakhar, Ragesh Gupta, Ravi Gurjar, and Anurag Singh. “Importance of CURE Clustering Algorithm over K-Means Clustering Algorithm for Large Data-Set.” In *2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC)*, 421–426. IEEE, 2023.

2. Guha, Sudipto, Rajeev Rastogi, and Kyuseok Shim. “Cure: An Efficient Clustering Algorithm for Large Databases.” *Information systems* (Oxford) 26, No. 1 (2001): 35–58.

3. Laksono, Muhammad Agung Tri, Yudha Purwanto, and Astri Novianty. “DDoS Detection Using CURE Clustering Algorithm with Outlier Removal Clustering for Handling Outliers.” In *2015 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, 12–18. IEEE, 2015.

4. Leskovec, Jurij, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets*. Third edition. Cambridge, United Kingdom ; Cambridge University Press, 2020.