ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ

ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)
στην ΑΝΑΠΤΥΞΗ and ΑΣΦΑΛΕΙΑ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

"Αξιολόγηση διαφορετικών αλγορίθμων ομαδοποίησης στην Ιεραρχική Kademlia"

Γεώργιος Νίτσος

221-p3312119

ΑΘΗΝΑ, Σεπτέμβριος 2023

Abstract

This study explores the potential improvements in network performance that Hierarchical Distributed Hash Tables (H-DHT) could offer compared to Distributed Hash Tables (DHTs). DHTs serve as vital components in numerous peer-to-peer (P2P) systems, fulfilling crucial functions like routing, data storage, and data dissemination. Moreover, hierarchical DHTs modify the traditional structure by introducing advantages such as fault isolation, effective caching, request aggregation while retaining key benefits found in flat topologies, such as load balancing and fault tolerance.

Our investigation initially evaluates the performance of some popular clustering algorithms on a real-world scenario, using a latency dataset of internet endpoints. The objective is to partition the network into several clusters while minimizing the intra-cluster latency. Metrics such as average cluster size, cluster size deviation, and cluster count has been considered to assess the clustering quality of the algorithms.

Then, a dedicated study of Hierarchical Kademlia (H-Kademlia) is conducted. Kademlia is a widely known DHT, used in various decentralized applications, including file sharing systems, blockchain networks, and distributed storage platforms. H-Kademlia is a novel implementation of vanilla Kademlia constructed according to the Canon paradigm, a design pattern proposed to convert a flat DHT into a hierarchical one.

Finally, utilizing a PeerNet P2P network simulator the impact of Hierarchical Kademlia in various node grouping scenarios, as generated by the initial phase of our study, is evaluated. The evaluation involves a comparison of network performance, focusing on factors such as latency and the number of hops needed for a node to either locate a key or store a value. This comparison spans different clustering configurations and is contrasted with the performance of the standard Kademlia protocol.

Περίληψη

Η εργασία αυτή μελετάει τα πλεονεκτήματα στην απόδοση των ομότιμων (Peer-to-Peer) δικτύων που προκύπτουν από την χρήση Hierarchical Distributed Hash Tables (H-DHT) έναντι των Distributed Hash Tables (DHT). Τα DHT είναι βασικό κομμάτι των ομότιμων δικτύων καθώς είναι υπεύθυνα για την δρομολόγηση των αιτημάτων, την αποθήκευση και το διαμερισμό των δεδομένων στους κόμβους του δικτύου. Οι H-DHT αναδιαμορφώνουν την επίπεδη δομή των DHT σε ιεραρχική, επιδιώκοντας πλεονεκτήματα όπως η απομόνωση σφαλμάτων, η αποτελεσματική προσωρινή αποθήκευση δεδομένων, η μαζική αποστολή αιτημάτων που προσφέρονται από τα χαρακτηριστικά της ιεραρχικής δομής, διατηρώντας παράλληλα τα πλεονεκτήματα της επίπεδης δομής όπως η ανοχή στα σφάλματα και η ομοιόμορφη κατανομή του φόρτου των αιτημάτων.

Το πρώτο μέρος της εργασίας ερευνά και αξιολογεί τα αποτελέσματα διαφορετικών μεθόδων ομαδοποίησης (clustering) με βάση ένα σύνολο δεδομένων με χρόνους απόκρισης κόμβων του διαδικτύου. Σκοπός των αλγόριθμων ομαδοποίησης είναι να χωρίσουν τους κόμβους σε ομάδες ώστε να ελαχιστοποιείται ο χρόνος απόκρισης μεταξύ των κόμβων της ομάδας. Δείκτες όπως το μέσο μέγεθος ομάδας και η τυπική απόκλιση του μεγέθους των ομάδων χρησιμοποιούνται για την αξιολόγηση της ποιότητας τηε ομαδοποίησης.

Το δεύτερο μέρος της εργασίας μελετά το Hierarchical Kademlia (H-Kademlia). Το Kademlia είναι ένα από τα πλέον γνωστά DHT καθώς χρησιμοποιείται για τον διαμερισμό αρχείων, την λειτουργία αλυσίδων καταχώρισης και την αποθήκευση περιεχομένου σε αποκεντρωμένα συστήματα. Το H-Kademlia αποτελεί μια υλοποίηση της Kademlia βάση του προτύπου σχεδιασμού Canon, που περιγράφει ένα τρόπο μετατροπής της επίπεδης τοπολογίας δικτύου σε ιεραρχική.

Τέλος χρησιμοποιείται ο προσομοιωτής λειτουργίας ομότιμων δικτύων PeerNet για να αξιολογηθεί η απόδοση του H-Kademlia στους διαφορετικούς τύπους ομαδοποίησης των κόμβων που παρήχθησαν στο πρώτο μέρος της εργασίας. Η αξιολόγηση της απόδοσης του δικτύου εστιάζει στο χρόνο απόκρισης και τον αριθμό των βημάτων που χρειάζεται ένας κόμβος για ένα εντοπίσει ή να αποθηκεύσει μια τιμή στο δίκτυο. Η αξιολόγηση χρησιμοποιεί ως βάση την απόδοση του «επίπεδου» Kademlia και εξετάζει τις διαφορές στην απόδοση του H-Kademlia που προκύπτουν από τους διαφορετικούς τύπους ομαδοποίησης.

Table of Contents

1. Introduction

This thesis investigates the potential advantages of employing a Hierarchical Distributed Hash Table (DHT) [2], Hierarchical Kademlia [12], within a Peer-to-Peer content distribution network. Hierarchical Kademlia is built upon the Canon paradigm [2], aiming to harness the benefits derived from its hierarchical structure, such as caching, fault tolerance, and request aggregation.

The primary objective is to explore how different strategies for grouping network nodes into clusters can impact the performance of Hierarchical Kademlia and, consequently, the overall network performance. This investigation places particular emphasis on the process of clustering random network nodes into non-overlapping groups, while minimizing intra-cluster latency.

To achieve this, several well-known clustering algorithms, including K-Means [10], DBSCAN [14], Spectral [13], and Louvain [15], are executed on top of a real-world latency dataset. The outcomes are visualized and assessed based on key criteria, including intra-cluster latency, cluster count, and cluster size deviation.

Subsequently, the results obtained from the clustering process are applied in experiments conducted within a Peer-to-Peer network simulator named PeerNet [7]. PeerNet simulates the network traffic of a Peer-to-Peer network and produces traces. Hierarchical Kademlia is implemented within the PeerNet simulator and initialized based on the clustering results. The PeerNet traces from various simulated scenarios are assessed and compared to each other, as well as against the baseline (non-hierarchical) Kademlia version.

2. Background and Related Work

2.1. Inter Planetary File System

The InterPlanetary File System (IPFS) [1] is a Peer-to-Peer system designed and deployed to store data files in a fully distributed, decentralized, and collaborative way. IPFS replicates every file across multiple nodes in the network to handle nodes departing from the network and safeguard the content against unauthorized modifications. Furthermore, an on-demand replication policy establishes a clear relationship between a file's replication factor and its popularity, effectively tackling scalability challenges for highly popular items. In IPFS, files are divided into immutable blocks, with the content of each block being hashed to generate a 256-bit Content Identifier (CID). The CID of a block is used to identify the nodes responsible for storing its metadata, including references to all nodes currently storing a copy of that block. This mapping of CIDs to nodes is accomplished through a Distributed Hash Table (DHT).

2.2. Distributed Hash Table

A hash table is a data structure used for mapping keys to values. A distributed hash table (DHT) is a type of hash table that is constructed and managed by nodes within a distributed system. DHTs play a pivotal role in numerous distributed peer-to-peer (P2P) systems, contributing to functions such as routing, data storage, and data distribution. Various implementations of DHT exist, with the most commonly used ones being Chord [6], Kademlia [3], Content-Addressable Network [4] and Pastry [5].

2.2.1. Kademlia

System Description

Kademlia [3] is a Peer-to-Peer key-value storage and lookup system. In the Kademlia Distributed Hash Table (DHT), each node is assigned a 128-bit ID in a uniform and random manner. Kademlia offers a lookup algorithm that, given a key, progressively identifies nodes that are closer to that key. This algorithm converges to the key in a logarithmic number of steps. Within Kademlia, configuration information naturally spreads as a byproduct of key lookups. Kademlia employs parallel and asynchronous queries to prevent delays caused by timeouts due to failed nodes. The method by which nodes keep a record of each other's presence is designed to withstand certain fundamental denial of service attacks. Many of the advantages of Kademlia are a result of its use of the XOR metric for measuring the distance between points in the key space. XOR is symmetric, enabling Kademlia participants to receive lookup queries from the same distribution of nodes contained in their routing tables. A modified version of Kademlia is used by IPFS to facilitate its operations.

Node State

In Kademlia, nodes maintain contact details about one another to facilitate the routing of query messages. For each value of i within the range $0 \leq i < 128$, each node maintains a collection of <IP address, UDP port, Node ID> triples representing nodes with distances falling between 2^i and $2^{(i+1)}$ from itself, as illustrated in Figure 1. These collections are referred as "k-buckets", with each bucket being meticulously organized based on the last time a node was encountered. Nodes that were seen least recently occupy the top of the list, while the most recently seen nodes are placed at the end.

When a Kademlia node receives any message from another node, it updates the appropriate k-bucket for the sender's node ID, as shown in Figure *2*.

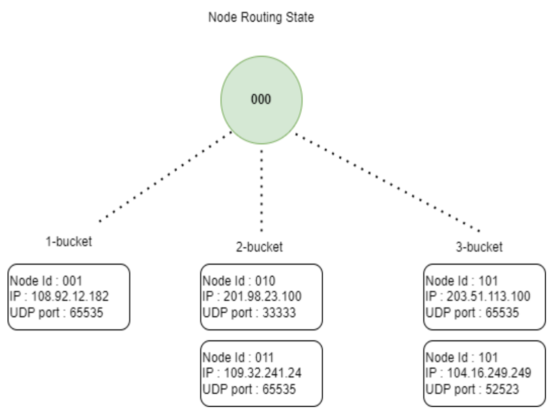

Figure 1 Node K-bucket

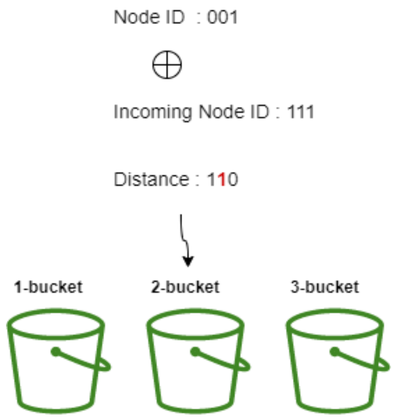

Figure 2 K-bucket insertion example

When the sending node is already present in the recipient's k-bucket, the recipient relocates it to the end of the list. If the node isn't already in the relevant k-bucket, and that k-bucket contains fewer than 'k' entries, the recipient inserts the new sender at the end of the list. However, if the relevant k-bucket is full, the recipient initiates communication with the least-recently seen node within that k-bucket to determine the next course of action. If the least-recently seen node fails to respond, it is removed from the k-bucket, making room for the new sender, who is then placed at the end of the list.

Protocol

The Kademlia protocol consists of four Remote Procedure Calls (RPCs): PING, STORE, FIND_NODE, and FIND_VALUE. PING checks if a node is online. STORE directs a node to store a <key, value> pair. FIND_NODE accepts a 128-bit ID as an argument, and in response, the recipient of the RPC provides <IP address, UDP port, Node ID> triples for the k nodes that are closest to the target ID, according to its k-buckets (that is, the closest nodes that it is aware of). Similarly, FIND_VALUE returns <IP address, UDP port, Node ID> triples, for the nodes closest to the requested key. However, if the node has already stored the value locally, it returns the requested value.

The most important procedure of Kademlia is node lookup, during which a node tries to identify the k nearest nodes to a specified node ID. Kademlia employs a recursive algorithm for conducting node lookups. The initiation of the lookup process involves selecting the nodes closest to the target node ID from the initiating node’s k-buckets, followed by dispatching parallel, asynchronous FIND_NODE RPCs to the α nodes that have been chosen. Here, α represents a system-wide concurrency parameter, often set to a value like 3. To store a <key, value> pair, a node locates the k nearest nodes to the key and dispatches STORE RPCs to them, accordingly.

Caching

In Kademlia, once a lookup operation succeeds, the requesting node stores the <key, value> pair at the nearest node to the key, provided that this node did not previously return the value. Due to the one-way nature of the network topology, future searches for the same key are highly likely to encounter cached entries before reaching out to the closest node. Consequently, during periods of increased popularity for a particular key, the system might end up caching it at multiple nodes. To prevent excessive caching, the expiration time of a <key, value> pair within a node's database decreases exponentially and inversely proportional to the number of nodes situated between the current node and the node whose ID is closest to the key ID.

Content Retention

In Kademlia, each node reissues its <key, value> pair STORE RPCs every hour, to guarantee the enduring presence of these associations. Additionally, Kademlia imposes a requirement on the original publishers of a <key, value> pair to reissue it at least once every 24 hours; otherwise, it is subject to expiration. This approach restricts the accumulation of outdated or stale information within the network.

2.3. Hierarchical Distributed Hash Tables

As discussed in P. Ganesan et al. [2], DHTs are flat non-hierarchical structures, but scaling has always revolved around exploiting hierarchy. That paper argues that the best of both worlds can be obtained by designing hierarchically structured DHTs based on the Canon paradigm. The benefits of a flat design such as the even distribution of functionality, load balancing among the participating nodes and the absence of a single point of failure, can be retained, while also harnessing the advantages inherent in hierarchical designs, such as fault isolation, enhanced security, efficient caching, and optimized bandwidth utilization. A multi-level DHT constructed according to the Canon paradigm achieves:

- Convergence of inter-domain paths, i.e., when messages originate from a domain A and are directed towards a node in a different domain, they all follow the same path as they exit domain A and proceed to their destination node. This characteristic simplifies caching and enables the aggregation of requests.

- Locality of intra-domain paths, i.e., in the case where messages originate from a domain A and are destined for a node within the same domain, none of these messages exit domain A. This attribute enhances the robustness of the architecture in the face of network failures and enhances communication performance.

Artigas et al. [16] distinguish two main hierarchical DHT designs. The horizontal design as shown in Figure 3, in which all nodes act in equal roles, and the vertical design, in which a relatively small group of super-peers behave as proxies, interconnecting clusters of unstable peers as shown in Figure 4.

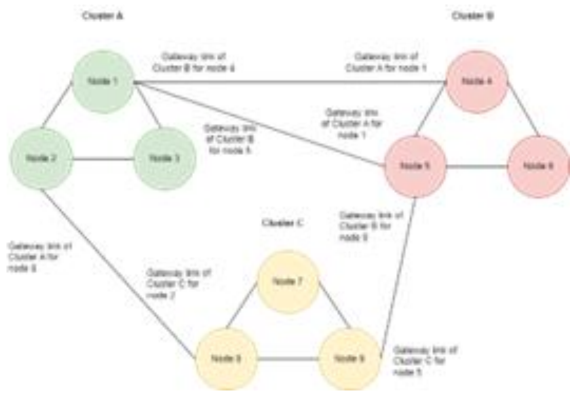
Cluster A
Cluster B
Cluster C
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Node 7
Node 8
Node 9
Gateway link of Cluster B for node 4
Gateway link of Cluster A for node 1
Gateway link of Cluster B for node 5
Gateway link of Cluster A for node 1
Gateway link of Cluster B for node 9
Gateway link of Cluster A for node 8
Gateway link of Cluster C for node 2
Gateway link of Cluster C for node 5

Figure 3 Horizontal Hierarchical DHT design

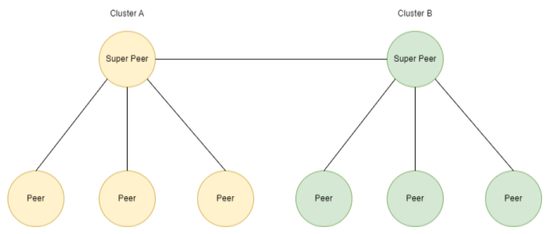

Figure 4 Vertical Hierarchical DHT design

In the current study the implementation of Hierarchical-Kademlia [12] follows the horizontal design of Canon, which will be used to evaluate the performance of a H-DHT for the IPFS network. This implementation of Kademlia requires the nodes of the network to be grouped into clusters and introduces a modified K-bucket maintenance process which presents a local and remote peer categorization. A local peer belongs to the same cluster as the reference peer, whereas a remote peer belongs to a different cluster. The K-bucket maintenance process of H-Kademlia assures that the remote peers maintained in the reference peer's K-buckets are those to which that reference peer is the XOR-based closest among the rest of the peers of the cluster. To achieve this goal, a peer, peer A, performs the following checks before inserting a candidate peer, peer C, in its K-buckets:

- In the scenario where peer C is in a different cluster, peer A actively seeks peers from its own cluster that are closer in proximity to peer C. If a suitable local peer is identified, the decision is made not to include peer C in the K-buckets.

- If peer C is in the same cluster as peer A, it is incorporated into the K-bucket following Kademlia's guidelines. Once peer C is added, peer A proceeds to eliminate from its K-buckets any pre-existing remote peers that happen to be closer to peer C than peer A is.

2.4. Clustering

Clustering [9] in general means grouping a set of objects in such a way that objects in the same cluster (i.e., group) are more similar to each other than to those in other clusters. Clustering can be achieved by various algorithms that differ significantly in their understanding of what constitutes a cluster (that is, what similarity means) and how to efficiently find these clisters. Popular notions of clusters include groups with small distances between cluster members and dense areas of a data space. Clustering can therefore be formulated as a multi-objective optimization problem. The appropriate clustering algorithm and parameter settings depend on the individual data set and the intended use of the clusters.

This study aims to employ the most popular clustering algorithms in literature to segment the network into clusters of nodes according to their latency. The outcomes are presented visually and assessed.

2.4.1. Partition-Based Clustering

The basic idea of this kind of clustering algorithms, as described by Xu et al. [8], is to regard the center of the data points as the center of the corresponding cluster. K-means [10] and K-medoids [11] are the two most common partition-based clustering algorithms. The core idea of K-means is to update the center of a cluster, which is represented by the center of the data points, by iterative computation; the iterative process is continued until some criterion for convergence is met. K-medoids is an improvement of K-means that deals with discrete data, which takes the data point closest to the center of the data points, as the representative of the corresponding cluster. Typical clustering algorithmss based on partition also include PAM [17], CLARA [18], CLARANS [19].

K-Means

The K-Means algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the *inertia* or within-cluster sum-of-squares. This algorithm requires the number of desired clusters as an input. It scales well to large numbers of samples and has been used across a wide range of application areas in many different fields.

The k-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j of the samples in the cluster. The means are commonly called the cluster "centroids". They are not, in general, points from X, although they belong to the same space.

The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum-of-squares criterion:

$$\sum_{i=0}^{n} \quad min_{\mu_j \in C}(||x_i - \mu_j||^2)$$

Inertia is thus a measure of how internally coherent clusters are.

In practice, the algorithm has three steps. The first step chooses initial centroids; the simplest method is to choose actual samples from the dataset. After initialization, K-means loops between the two other steps. The first step assigns each sample to its nearest centroid. The second step creates new centroids by taking the mean value of all the samples assigned (provisionally clustered) to each previous centroid. The difference between the old and the new centroids are computed, and the algorithm repeats these last two steps until this value is less than a threshold. In other words, it repeats until the centroids do not change significantly.

The advantages of the K-means algorithm include:

- **Simplicity and Ease of Implementation**: K-means is relatively simple to understand and implement, making it a good starting point.
- **Scalability**: The algorithm is scalable and can handle large numbers of data points and clusters without significant computational overhead.
- **Interpretability**: The resulting clusters are easy to interpret, making it useful for exploratory data analysis. You can visually inspect the clusters and understand their characteristics.
- **Versatility**: K-means can be applied to various types of data, including numerical and continuous data, as well as multi-dimensional data.
- **Convergence**: K-means is guaranteed to converge to a local minimum, and with proper initialization techniques, it can often converge to a meaningful clustering solution.

The disadvantages of the K-means algorithm include:

- **Sensitivity to Initialization**: The algorithm's convergence to a local minimum can be influenced by the initial placement of cluster centroids. If the initial centroids are chosen poorly, K-means may converge to suboptimal solutions.
- **Assumption of Equal Sized and Spherical Clusters**: K-means assumes that clusters are of equal size and spherical in shape. This assumption might not hold for all types of data, leading to poor results when clusters are irregularly shaped or have different sizes.
- **Sensitivity to Outliers**: K-means can be sensitive to outliers, which can disproportionately affect the placement of centroids and subsequently the cluster assignments.

- **Number of Clusters (K) Needs to be Specified**: The number of clusters K needs to be predetermined by the user, which might not be straightforward in some cases.
- **Inefficient with Categorical Data**: K-means is designed for numerical data and may not be directly applicable to categorical data without appropriate preprocessing.

K-medoids

The ***k*-medoids** algorithm is a variation of k-means. Both algorithms aim to minimize the distance between points the points assigned to a cluster and its center, but *k*-medoids chooses actual data points as centers (medoids), thereby allowing for better interpretability of the cluster centers than in *k*-means, where the center of a cluster is not necessarily an actual data point. Furthermore, *k*-medoids can be used with arbitrary dissimilarity measures, whereas *k*-means generally requires Euclidean distance for efficient solutions. Because *k*-medoids minimizes a sum of pairwise dissimilarities rather than a sum of squared Euclidean distances, it is more robust to noise and outliers than *k*-means.

The advantages of the K-medoids algorithm include:

1. **Robustness to Outliers:** K-medoids is more robust to outliers compared to K-means because medoids are actual data points from the dataset. This makes it less sensitive to outliers' influence on cluster formation.
2. **Better Handling of Non-Euclidean Distances:** K-medoids can handle non-Euclidean distances, such as Manhattan distance or other custom distance metrics, more effectively compared to K-means.
3. **Interpretable Cluster Representatives:** The medoids in K-medoids are actual data points from the dataset, making them more interpretable and representative of the cluster's characteristics.
4. **Less Affected by Initialization:** K-medoids is less sensitive to initialization compared to K-means, because medoids are chosen from the actual data points, reducing the risk of converging to poor local optima.

The disadvantages of the K-medoids algorithm include:

1. **Higher Computational Complexity:** K-medoids has a higher computational complexity than K-means, as it involves pairwise distance calculations between data points, which can be computationally expensive for large datasets.
2. **Number of Clusters (K) Needs to be Specified:** Similar to K-means, the number of clusters K needs to be specified in advance for K-medoids.
3. **Slow Convergence:** The medoid update step in K-medoids can lead to slower convergence compared to the centroid update step in K-means.

2.4.2. Density-Based Clustering

The basic goal of these clustering algorithms is to create clusters wherever the data space is denser. Examples of such algorithms are DBSCAN [14], OPTICS [20] and Mean-shift [21]. DBSCAN is the most popular density-based algorithm.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm creates clusters in areas of high density, which are separated by areas of low density. The clusters created by DBSCAN can be of any shape, as opposed to k-means which assumes that clusters are convex. The central concept of DBSCAN is *core samples*, which are samples that are in areas of high density. A cluster is therefore a set of *core samples*, each close to each other, plus a set of non-core samples that are close to a core sample (but are not in themselves core samples). The algorithm requires two input parameters, minimum samples, and epsilon, which formally define what we mean when we say “dense”. Minimum samples are the minimum number of data points that must be within an epsilon radius neighborhood of a data point, for it to be considered a core point. Higher minimum samples or lower epsilon indicate that higher data density is needed to form a cluster.

More formally, a core sample is defined as a sample in the dataset such that a minimum number of other samples are located within a distance of epsilon from that sample; these are the neighbors of the sample. A cluster is a set of core samples, built by recursively taking a core sample, finding all its neighbors that are core samples, and so on. A cluster also has a set of non-core samples; intuitively, these samples are on the fringes of a cluster.

Any core sample is by definition part of a cluster. Any sample that is not a core sample and is at least epsilon in distance from any core sample, is considered to be an outlier by the algorithm.

While the minimum samples parameter primarily controls how tolerant the algorithm is to noise, the parameter epsilon is crucial for the data set and distance function and usually cannot be left at a default value, as it controls the size of the local neighborhood of the samples. When it is too small, most data will not be clustered at all. When it is too large, it causes close clusters to be merged into one cluster, and eventually the entire data set to be returned as a single cluster. Some heuristics for choosing this parameter have been discussed in the literature, for example looking for a knee in the nearest neighbor distances plot.

The advantages of the DBSCAN algorithm include:

1. **Ability to Find Arbitrary Shapes:** DBSCAN can identify clusters with irregular shapes and does not assume that clusters are globular or convex.

2. **Robust to Noise:** Noise points are identified as points that do not belong to any cluster, which is useful for data with outliers or irrelevant points.
3. **No Need to Specify the Number of Clusters:** DBSCAN does not require the user to specify the number of clusters beforehand, making it well-suited for cases where the number of clusters is unknown.
4. **Does Not Rely on Distance Metrics:** DBSCAN can work with any metric that defines a notion of distance, making it applicable to various types of data, including non-Euclidean spaces.

The disadvantages of DBSCAN include:

1. **Sensitive to Parameter Settings:** The performance of DBSCAN can be sensitive to the choice of parameters, especially of the epsilon value. Setting these parameters can require domain knowledge or experimentation.
2. **Difficulty with Varying Densities:** DBSCAN can struggle when dealing with clusters of varying densities, as the same epsilon value might lead to either over-clustering in dense areas or under-clustering in sparse areas.
3. **Not Scalable to Very High-Dimensional Data:** Like many clustering algorithms, DBSCAN's performance can degrade in very high-dimensional spaces.

2.4.3. Spectral Clustering

Spectral Clustering [13] uses the connectivity between data points to form clusters. It uses eigenvalues and eigenvectors of the data matrix to project the data into a lower dimension space, so as to cluster the data points. It uses a graph representation of data, where the data points are represented as nodes and the similarity between the data points are represented by an edge.

The first step in Spectral Clustering is to build a similarity graph for the data, in the form of an adjacency matrix A. The adjacency matrix can be built in two ways:

- **Epsilon-neighborhood:** A parameter epsilon is fixed beforehand. Then, each point is connected to all the points which lie in its epsilon-radius. If all distances between any two points are similar, then the weights of the edges (the distances) are not stored, leading to an undirected and unweighted graph.
- **K-Nearest Neighbors:** A parameter k is fixed beforehand. Then, for any two vertices u and v, an edge is directed from u to v if v is among the k-nearest neighbors of u. This leads to the formation of a weighted and directed graph; the graph does not have to be symmetric.

The second step is to project the data into a lower dimensional space, to account for the possibility that members of the same cluster may be far away in a given dimensional space. Thus, the dimensional space is reduced, making those points closer in the reduced dimensional space and allowing then to be clustered together. This projection is achieved by computing the Graph Laplacian Matrix.

Finally, the data are clustered using any traditional clustering technique – typically K-Means Clustering.

The advantages of Spectral Clustering include:

1. **Assumption-Less:** Spectral clustering does not assume that the data follow some property. Thus, this technique can be used for a more generic class of clustering problems.
2. **Dimensionality Reduction**: The algorithm uses eigenvalue decomposition to reduce the dimensionality of the data, making it easier to visualize and analyze.
3. **Cluster Shape**: Spectral clustering can handle non-linear cluster shapes, making it suitable for a wide range of applications.

The disadvantages of Spectral Clustering include:

- **Not Scalable:** As it involves building matrices and computing eigenvalues and eigenvectors, the algorithm is time consuming for dense datasets.
- **Noise Sensitivity**: It is sensitive to noise and outliers, which may affect the quality of the resulting clusters.
- **Number of Clusters:** The algorithm requires the user to specify the number of clusters beforehand (as in k-means).
- **Memory Requirements**: Significant memory is needed to store the similarity matrix, which can be a limitation for very large datasets.

2.4.4. Community Detection

Community detection algorithms use mathematical optimization or heuristic techniques to maximize a predefined objective function. This function quantifies the quality of the detected communities, often encapsulating a balance between internal connectivity and external separation. This optimization relies on identifying an arrangement of nodes that maximizes intra-group interactions while minimizing inter-group ties.

Louvain Community Detection Algorithm

The Louvain Community Detection algorithm [15] is used to extract the community structure of a network. It is a heuristic method based on modularity optimization. The algorithm works in two steps. On the first step it makes each node into a separate community, and then it tries to find the maximum positive modularity gain by moving each node to all its neighbor communities. If no positive gain is achieved by such a move, the node remains in its original community.

The modularity gain obtained by moving an isolated node i into a community Ccan be calculated by the following formula:

$$\Delta Q = \frac{k_{i,in}}{2m} - \gamma \frac{\Sigma_{tot} \cdot \quad k_i}{2m^2}$$

Where m is the size of the graph, $k_{i,in}$is the sum of the weights of the links from to i nodes in C, k_i is the sum of the weights of the links incident to node i, Σ_{tot} , is the sum of the weights of the links incident to nodes in C and γ is the resolution parameter.

This phase continues until no individual move can improve the modularity.

In the second phase, a new network is built, whose nodes are the communities found in the first phase. The weights of the links between the new nodes are given by the sum of the weights of the links between nodes in the corresponding two communities. Once this phase is complete, we reapply the first phase method to create bigger communities with increased modularity.

These phases, shown in Figure 5 [15], are executed until no modularity gain is achieved or is less than the predefined threshold.

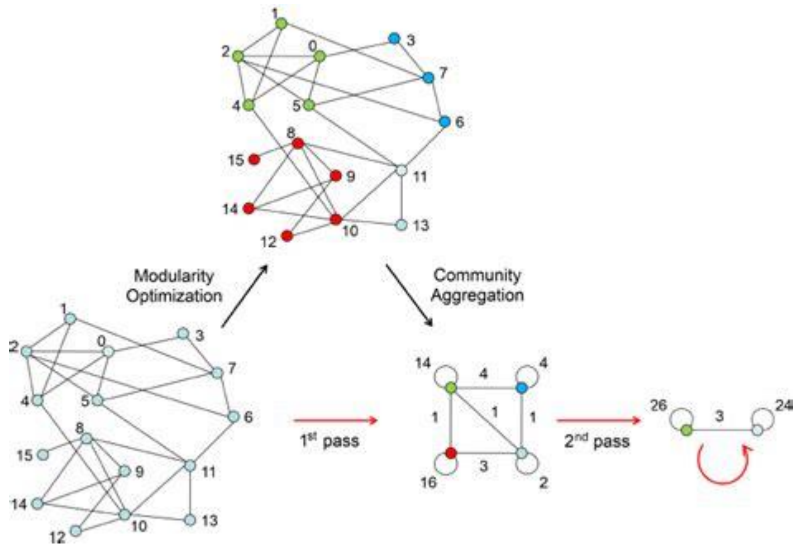

Figure 5 Louvain algorithm phases

The advantages of the Louvain algorithm include:

1. **Fast and Scalable:** The Louvain algorithm is known for its computational efficiency and scalability. It can handle large networks with millions of nodes and edges, making it suitable for analyzing real-world complex systems.
2. **Modularity Optimization:** The algorithm maximizes the modularity of the network, which quantifies the quality of the community structure. Modularity measures the difference between the actual number of edges within communities and the expected

number of edges in a random network, making it a meaningful way to assess community structure.

3. **Empirical Success:** The Louvain algorithm has been successfully applied to a wide range of real-world problems, from social network analysis to biological networks and recommendation systems.

The disadvantages of the Louvain algorithm include:

1. **Resolution Limit:** The Louvain algorithm suffers from the resolution limit problem, which means it might not be able to detect smaller communities within larger ones. This can lead to suboptimal results when dealing with networks where communities have vastly different sizes.
2. **Dependency on Initialization:** The algorithm's results can be influenced by the initial partition of nodes into communities. Different initializations might lead to different final community assignments and solutions. The proposed solution is to run the algorithm multiple times with different initializations.
3. **Deterministic Nature:** The algorithm is deterministic, meaning that it will always produce the same community structure for a given network and initialization. While this can be advantageous for reproducibility, it might miss certain nuances or alternative community structures that are equally valid.
4. **Parameter Choices:** The Louvain algorithm has some parameters that need to be set, such as the resolution parameter that controls the size of communities. Choosing appropriate parameter values can be a challenge and might require some trial and error.
5. **Optimization Complexity:** Despite being efficient, the algorithm's optimization process can become complex as the network size increases, and the algorithm might struggle with very dense networks or networks with specific structures.

3. Clustering implementation and evaluation

The clustering algorithms were implemented in Python using scikit-learn libraries [22] for K-means, K-medoids, DBSCAN and Spectral and the NetworkX library for Louvain [23] .

3.1. Clustering evaluation on test datasets

First, the different clustering algorithms will be tested and evaluated against some known test datasets to verify their performance and assess their inherent characteristics. Each dataset presents different challenges and provides us with a better understanding of an algorithm's advantages and limitations. In detail, the K-means, K-medoids, DBSCAN and Spectral algorithms will be evaluated. These datasets do not have a graph representation, so it was not possible to execute the Louvain algorithm on them.

3.1.1. Blobs

The blobs dataset is comprised of three distinct, well-separated clusters that exhibit a spherical shape. For the K-means, K-medoids, and Spectral algorithms, the cluster count was pre-established to 3. In datasets like these, accurately determining the correct number of clusters significantly influences the clustering's quality. All three algorithms K-Means, K-Medoids, and Spectral effectively segmented the nodes into the evident clusters, as demonstrated in Figure 6, Figure 7 and Figure 9. Notably, DBSCAN, without a preset cluster number, autonomously detected the three clusters and even identified specific samples as noise, thus excluding them from the cluster assignments as shown in Figure 9.

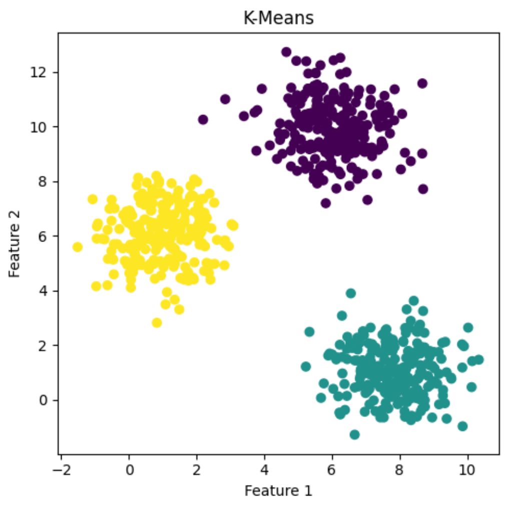

Figure 6 K-Means algorithm applied on Blobs dataset

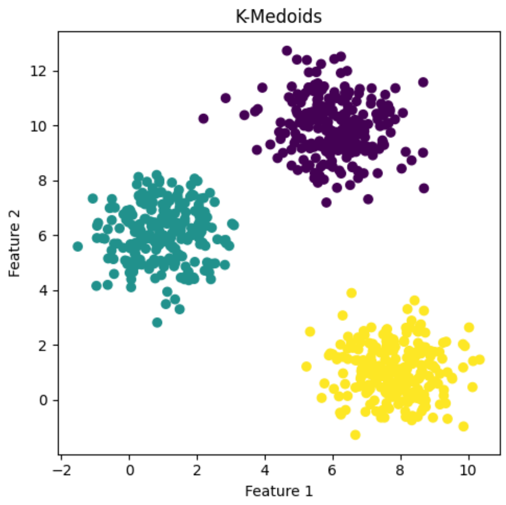

Figure 7 K-Medoids algorithm applied on Blobs dataset

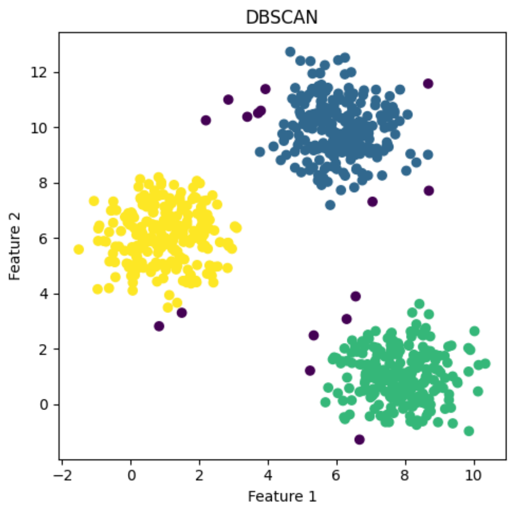

Figure 8 DBSCAN algorithm applied on Blobs dataset

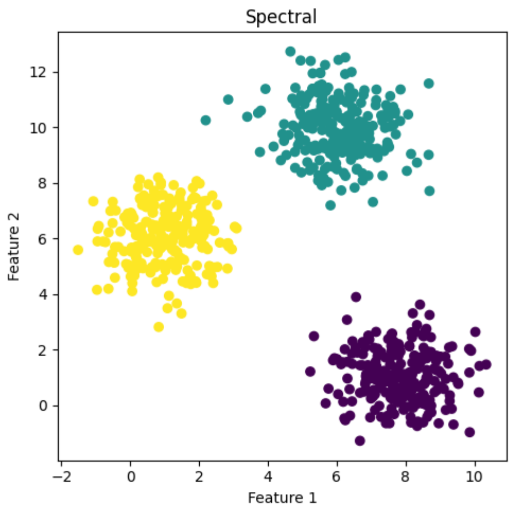

Figure 9 *Spectral algorithm applied on Blobs dataset*

3.1.2. Circles

The circles dataset serves as a foundational dataset to assess the clustering algorithm's capacity to identify embedded shapes. For the K-means, K-medoids, and Spectral algorithms, the number of clusters was predefined to 2. K-Means and K-Medoids partitioned the data into two balanced clusters, as illustrated in Figure 10 and Figure 11. However, this outcome arises from the inherent limitation of partition-based algorithms, which assume spherical clusters and consequently struggle to detect intricate patterns within the dataset. In contrast, Spectral Clustering (Figure 13) and DBSCAN (Figure 12) demonstrated the ability to discover such clusters, showcasing their capability to identify embedded shapes within the data. As in the previous dataset, the successful clustering achieved by the Spectral algorithm was attributed to the appropriate selection of the predefined number of clusters.

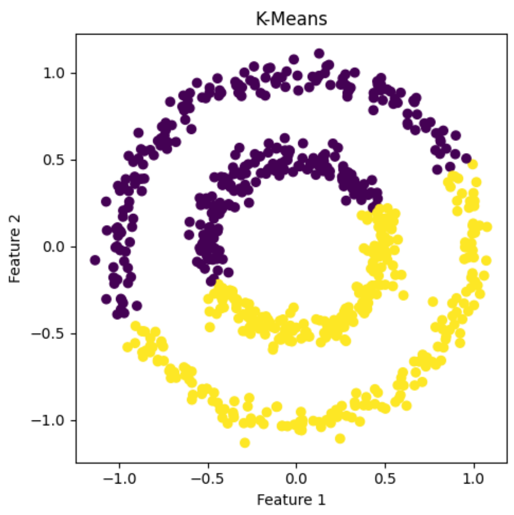

Figure 10 K-Means algorithm applied on Circles dataset

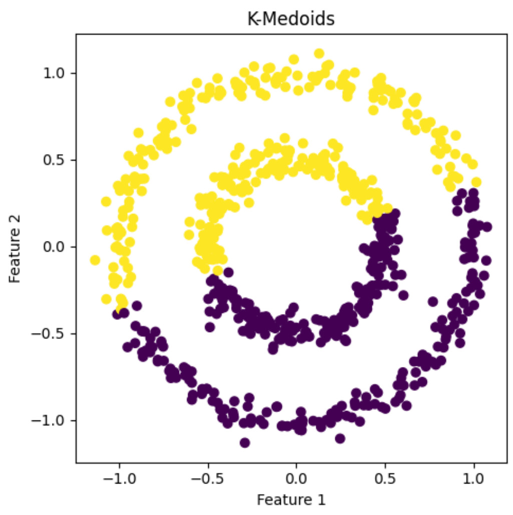

Figure 11 K-Medoids algorithm applied on Circles dataset

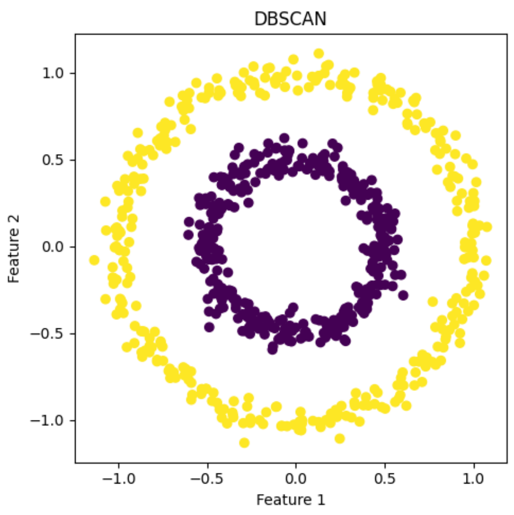

Figure 12 DBSCAN algorithm applied on Circles dataset

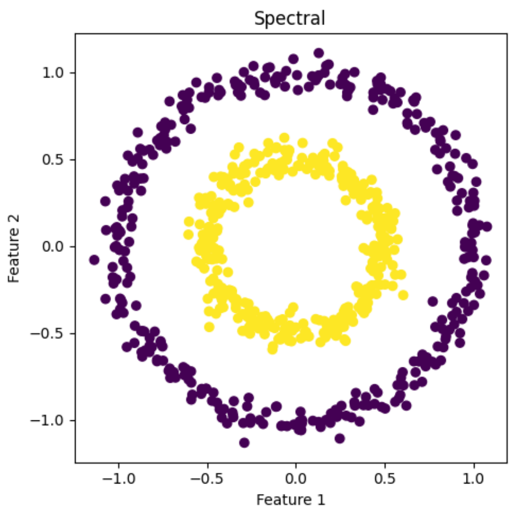

Figure 13 Spectral algorithm applied on Circles dataset

3.1.3. Elongated blobs

The elongated blobs dataset serves as a test for evaluating the algorithms' capability to identify clusters that are well-defined but not of a spherical shape. For the K-means, K-medoids, and Spectral algorithms, the predetermined number of clusters was set to 3. While K-means and K-medoids do manage to capture the primary trends of the clusters, their clustering quality falls short. As evidenced in Figure 14 and Figure 15, a few samples from the clusters positioned at the lower part of the diagram become intermingled between the clusters. It can be reasonably assumed that if the elongation of the blobs were to increase, the clustering performance of K-means and K-medoids would deteriorate further, underscoring their limitations in identifying non-spherical clusters.

Spectral, on the other hand, exhibits a fine clustering performance, as depicted in Figure 17. It appears that the dataset projection into a different-dimensional space allows the algorithm to effectively detect the underlying trends.

Like Spectral, DBSCAN successfully identifies the clusters formed by the samples, as shown in Figure 16, and designates a few samples as outliers. DBSCAN's good performance is attributed to the uniform sample density across all three clusters. However, the predefined epsilon parameter, representing the radius around a sample's neighbors, if not fine-tuned, could lead to the merging of the two clusters (yellow and green samples) positioned at the lower end of the diagram.

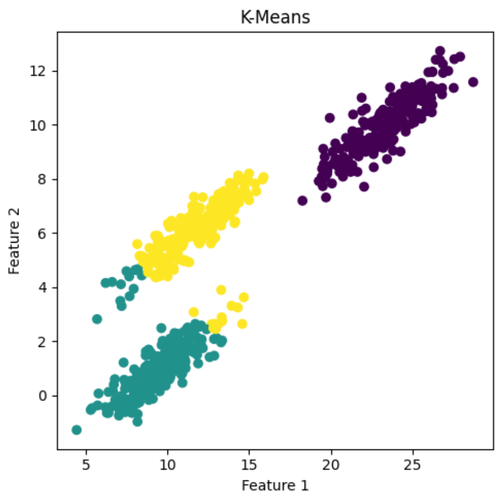

Figure 14 K-Means algorithm applied on Elongated Blobs dataset

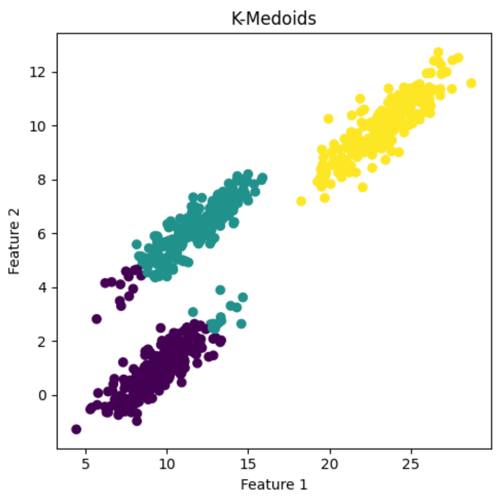

Figure 15 K-Medoids algorithm applied on Elongated Blobs dataset

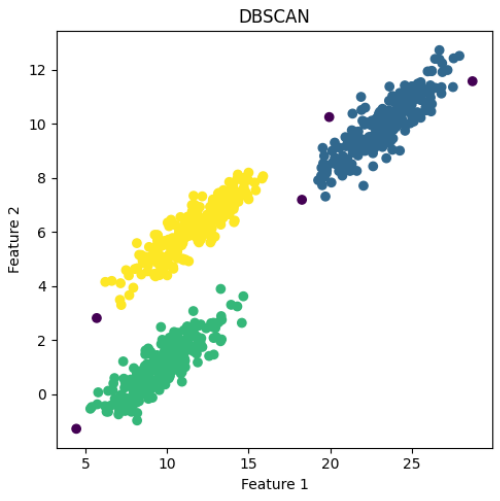

Figure 16 DBSCAN algorithm applied on Elongated Blobs dataset

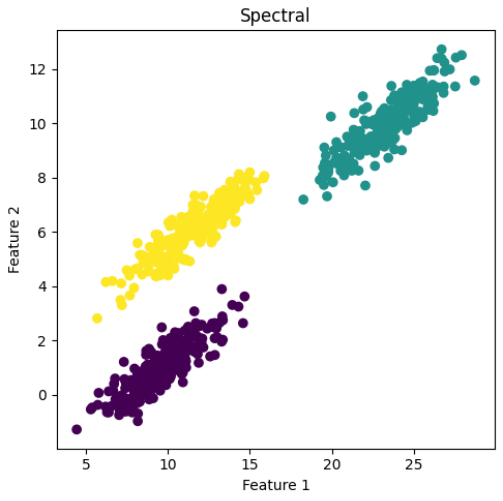

Figure 17 Spectral algorithm applied on Elongated Blobs dataset

3.1.4. Single blob

The single blob dataset seems to be the most interesting, as samples do not exhibit any obvious trend. For K-means, K-medoids, and Spectral algorithms, the predetermined number of clusters was set to 3. All three algorithms partitioned the dataset into 3 equal sized groups showcasing their ability of producing balanced clusters, even if there is no meaningful interpretation for them. Conversely, DBSCAN fails to identify substantial shifts in sample density, resulting in the formation of a solitary cluster as shown in Figure 20. Changing the input parameters of DBSCAN mainly affects the number of samples that are going to be considered as noise rather than affecting the number of the clusters.

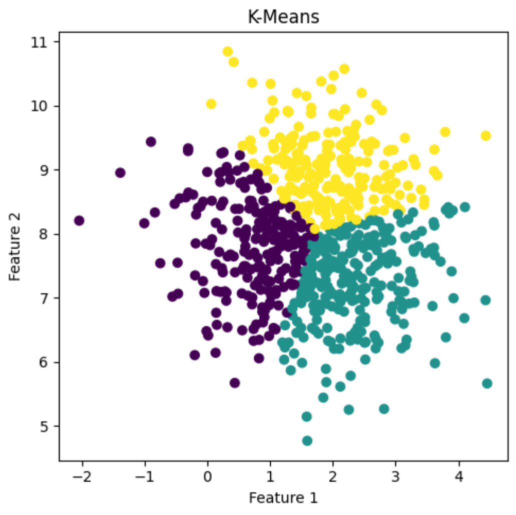

Figure 18 K-means algorithm applied on Single blob

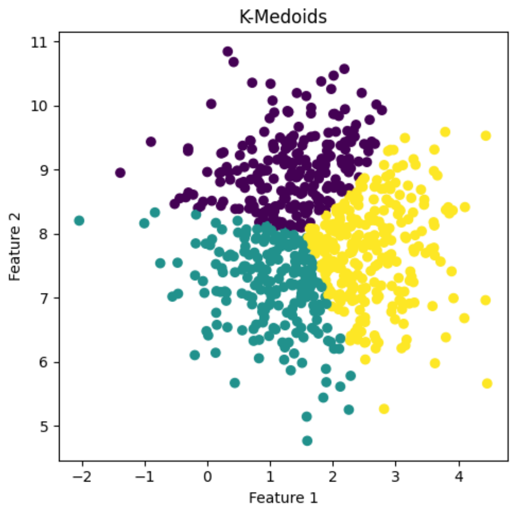

Figure 19 K-Medoids algorithm applied on Single blob

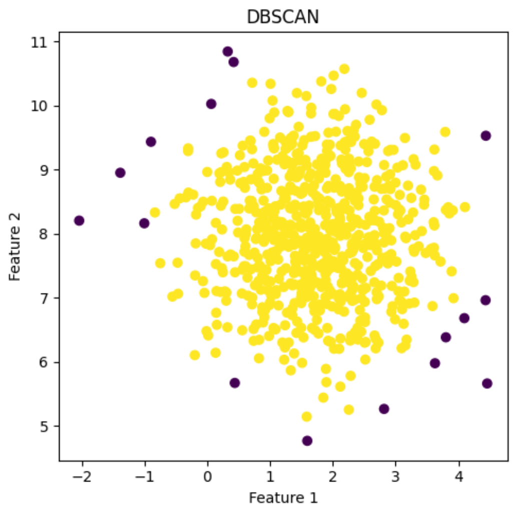

Figure 20 DBSCAN algorithm applied on Single blob

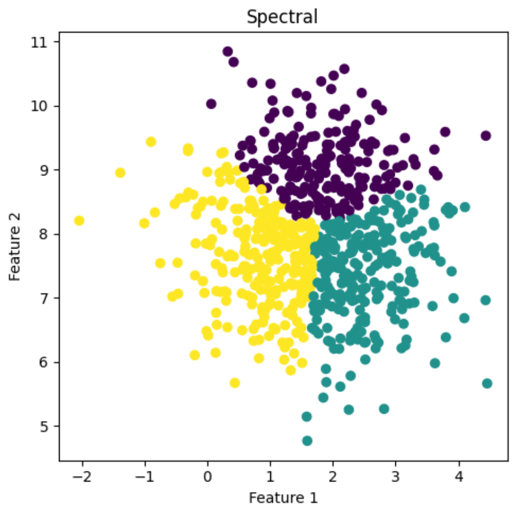

Figure 21 Spectral algorithm applied on Single blob

3.1.5. Discussion

Based on the outcomes, the spectral algorithm emerges as exceptionally versatile. It exhibited the capability to identify intricate shapes and density patterns, all while achieving a well-balanced partition of the single blob dataset as configured. However, it could be argued that the algorithm's versatility stems from its complex nature.

In contrast, the partition-based clustering algorithms, as anticipated, struggled to identify embedded shapes within the circular dataset. Furthermore, their clustering performance was subpar when dealing with non-spherically shaped clusters, such as in the case of the elongated blobs dataset.

The DBSCAN algorithm demonstrated its ability to identify non-convex shapes and outlier samples. Yet, it displayed no partition ability in the single blob dataset, which could be considered a drawback for our study case, as the network latencies dataset could have a similar form.

3.2. Clustering evaluation of the king latency dataset

The king latency dataset includes latencies between 1740 random internet nodes. The latencies were collected using the king tool [24], which was specifically developed for estimating latencies between end hosts. Consequently, the dataset utilized in the subsequent testing mirrors real-world scenarios, accurately representing latency measurements between nodes. The dataset is a square table denoted as T, where each entry $T_{i,j}$ corresponds to the latency between node i and node j. The dataset can be interpreted in two ways:

- as a weighted adjacency matrix of a graph
- as a high-dimensional space containing 1740 samples, with each sample having 1740 features.

The metrics that will be used to evaluate the quality of clustering produced by the algorithms described above are:

- Average cluster latency: Describes the average latency between the network nodes in a cluster as if they were the only nodes in the network. This metric is computed by adding all intra cluster latencies and dividing them by double the number of nodes in the cluster (due to the bidirectionality of the network)
- Average network latency: This metric signifies the mean latency that would result if all nodes were grouped into a single cluster, as in vanilla Kademlia. The value, as expected, remains constant across all experiments, offering insight into the extent to which the latency between two nodes within the same cluster is reduced compared to a random node on the network.
- Average number of nodes in cluster: It is computed by the division of the size of the network with the number of the clusters.

- Cluster size deviation: This metric is utilized to offer context regarding the uniformity of the network partitioning. It is calculated by determining the normalized deviation among the sizes of the clusters.

These metrics are displayed on the top left of the experiment diagrams (e.g., Figure 23).

The experiments were conducted with varying input parameters. For K-means, K-medoids, and Spectral algorithms, configurations were set to segment the network into 3, 5, 10, 20, 40, 60, 80, and 100 clusters respectively. For the Louvain and DBSCAN algorithms, the number of clusters could not be predefined. Therefore, manipulation of their input parameters was necessary to generate diverse cluster counts. For the sake of brevity, in each experiment, cluster visualization was performed solely for cluster counts of 3, 10, and 20, whenever feasible. The choice of 3 was based on the visual observation that the network appears to be composed of 3 distinct components. Opting for 20 aligns with Kademlia's replication factor, suggesting that if clusters were evenly distributed, each might potentially house a replica of a file within the cluster. The selection of 10 serves as an intermediate value between 3 and 20. An additional diagram is presented to illustrate the impact of the number of clusters on the average cluster latency.

3.2.1. Visualization of the dataset

Visualization serves the purposes of enhancing interpretability, validating, and improving the quality of clustering results. To facilitate visualization, the high-dimensional space with 1740 features is reduced to three dimensions using Principal Component Analysis (PCA) decomposition. This reduction is done by utilizing the PCA functionality provided by the scikit-learn library [22]. The accuracy achieved by the PCA decomposition was 80%. In the 3D plot shown in Figure 22 , every dot represents a sample (node) and the distance between two samples is directly proportional to the latency observed between the network nodes they represent.

In the spatial depiction of our network, illustrated in Figure 22, several dense sample regions become evident, followed by some sparsely located samples. Furthermore, a few even sparser samples are observed, scattered beyond the densely populated areas.

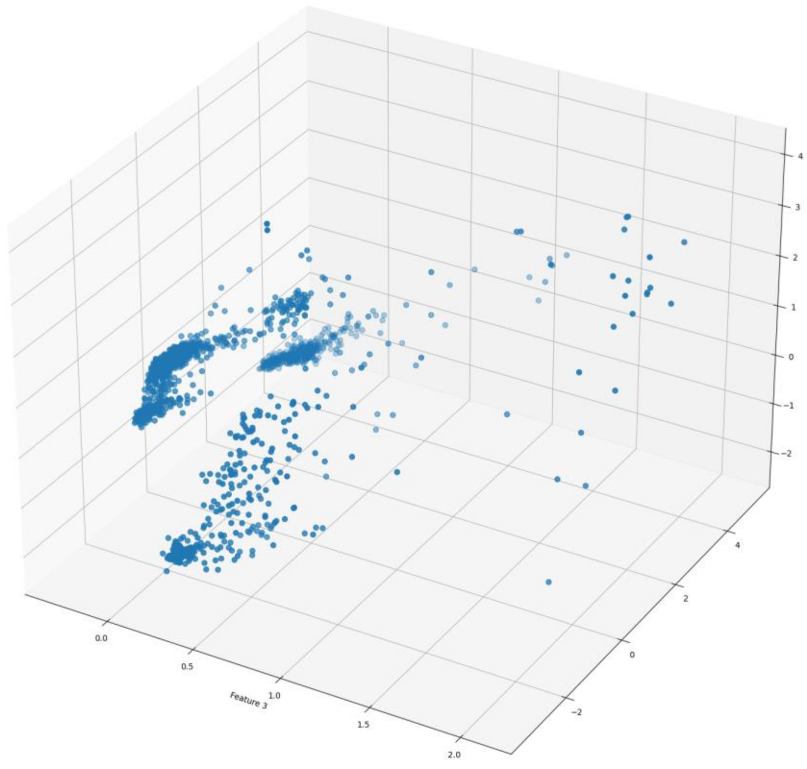

Figure 22 Visualization of the nodes in the king latency dataset

3.2.2. K-Means

The K-means algorithm exhibited swift and resilient performance with minimal parameter tuning, reinforcing its status as a versatile choice for a diverse range of applications. In Figure 23, it is evident that the algorithm has partitioned the nodes in 3 different kinds of clusters. Cluster no. 0, marked as blue, has 1310 nodes, accounting for approximately 75% of the entire network. Within this cluster, two densely concentrated sample regions exist that could potentially constitute separate clusters. This cluster demonstrates the best intra-cluster latency, as depicted in Figure 23 by the close proximity of nodes. Cluster no. 1, marked as orange, has 399 nodes, positioned more sparsely and extending across a larger portion of space. Its latency surpasses the average network latency. Finally, cluster no. 2, marked as green, appears to be the most sparsely populated. Consequently, its latency is the highest among all clusters, including only a minimal number of nodes.

In Figure 24 and Figure 25 the nodes are partitioned into 10 and 20 clusters respectively. The average cluster latency in both cases is less than a half of the network latency. As observed before, the clusters formed in the denser parts display low intra cluster latency while the clusters in the sparser parts display higher average latency than the network average latency, sometimes 2 to 3 times greater.

In Figure 26 we can see that the intra-cluster latency diminishes asymptotically as the number of clusters increases. This reasoning holds because the network can be divided into smaller, denser groups of low-latency nodes. This prevents these groups from extending over large areas, thereby avoiding the inclusion of nodes placed far apart in terms of latency.

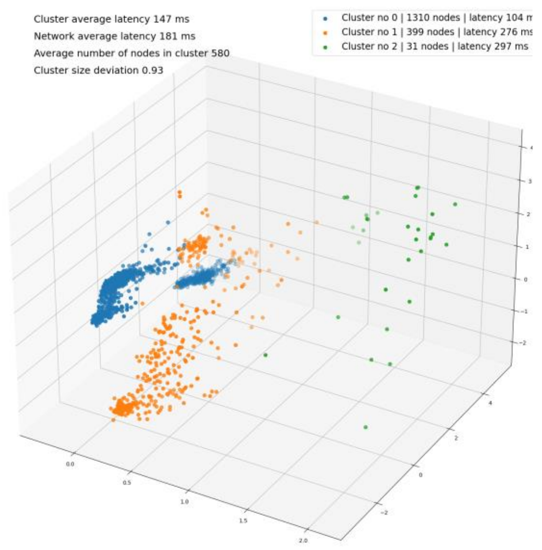

Figure 23 K-Means clustering for 3 clusters on the king latency dataset

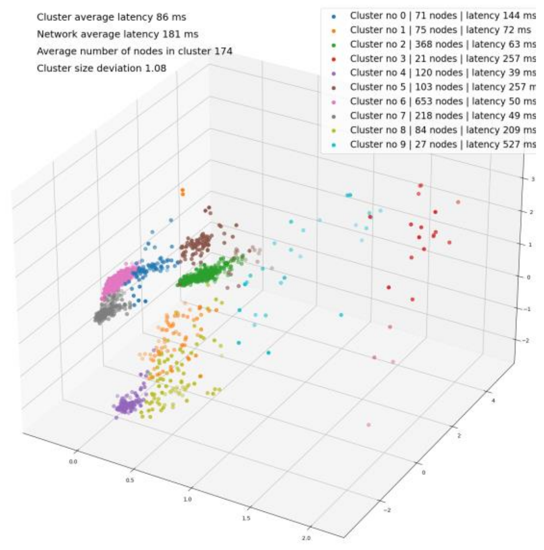

Figure 24 K-means clustering for 10 clusters on the king latency dataset

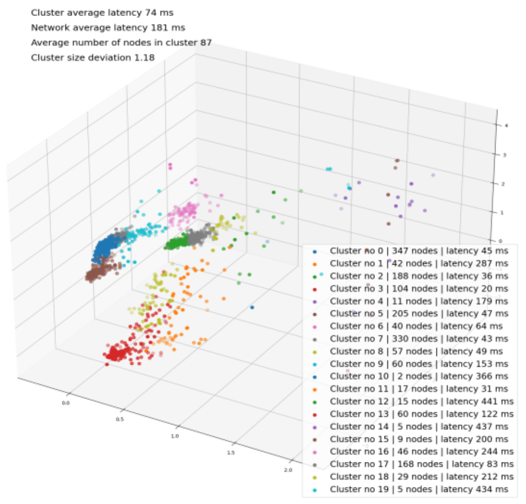

Figure 25 K-means clustering for 20 clusters on the king latency dataset

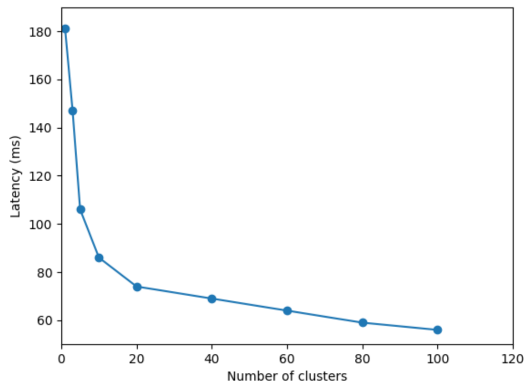

Figure 26 Intra-cluster latency as a function of the number of clusters for the K-Means algorithm

3.2.3. K-Medoids

K-medoids struggled to create effective clusters when applied directly to the raw dataset. Consequently, the dataset's features were standardized using the StandardScaler [22] method.

As shown in Figure 27, K-medoids for 3 clusters achieved better average cluster latency than K-means and produced more balanced clusters in terms of size. The blue cluster, has 374 nodes, occupies a substantial portion of the space, and exhibits a notably high average latency. On the other hand, the orange cluster stands out as the densest and most populated, displaying the lowest latency among the three clusters. Meanwhile, the green cluster is dense, and its latency is notably lower compared to the overall network average. It could be argued that the better quality of the clustering is due to the preprocessing of the data.

In Figure 28 and Figure 29 the network nodes are grouped into 10 and 20 clusters respectively. The average cluster latency has improved significantly, similarly to K-means, but only slightly worse. This could be due to the restriction of placing the center of the cluster into a specific node rather than in a point in the space that would yield better results.

Figure *30* exhibits a similar pattern to K-means regarding the relationship between the number of clusters and intra-cluster latency. However, it is worth noting that the average latency appears slightly elevated, and between the cluster counts of 80 and 100 there is a deviation from the diagram's trend, resulting in increased latency instead.

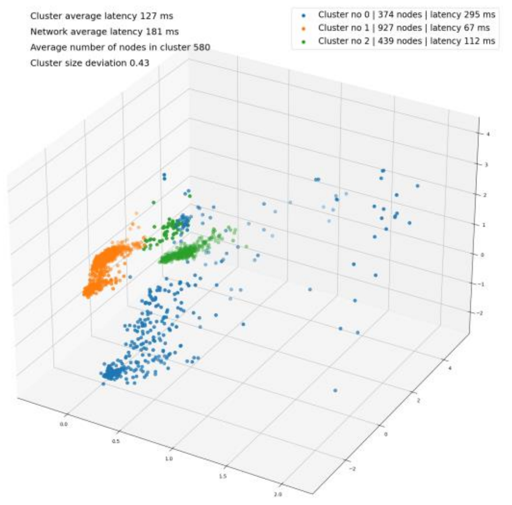

Figure 27 K-Medoids clustering for 3 clusters on the king latency dataset

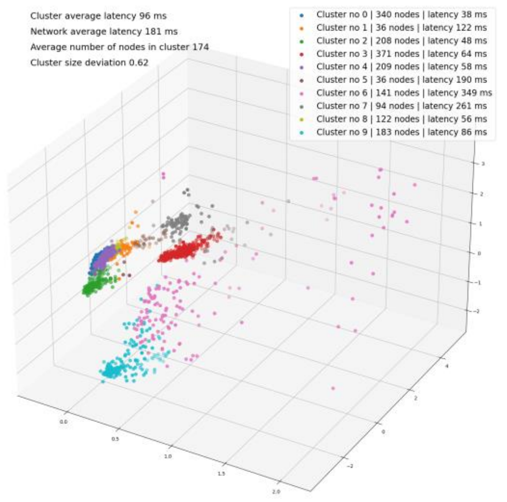

Figure 28 *K-Medoids clustering for 10 clusters on the king latency dataset*

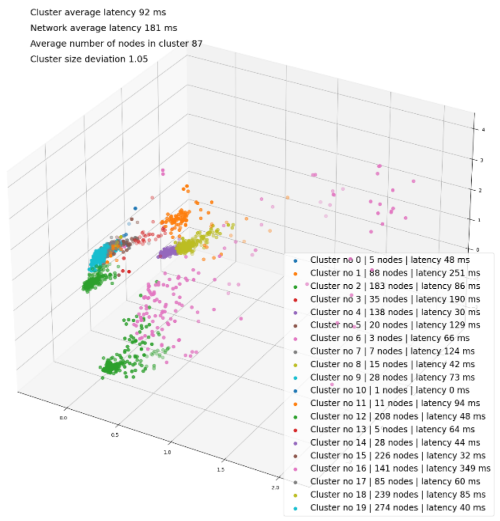

Figure 29 K-Medoids clustering for 20 clusters on the king latency dataset

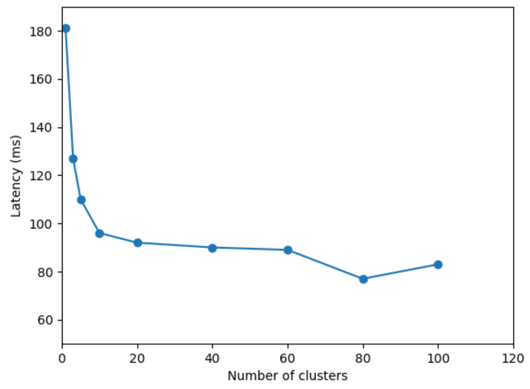

Figure 30 Intra-cluster latency as a function of the number of clusters for the K-Medoids algorithm

3.2.4. Spectral Clustering

The Spectral algorithm [22] required data preprocessing like K-medoids and a more complex configuration to partition the network. Specifically, the same data scaling method was used as in K-medoids and the affinity was configured to '*nearest_neighbors*'.

As shown in Figure 31, the Spectral algorithm achieved slightly better results than K-means and slightly worse results than K-medoids. The sizes of the clusters are similar to the ones produced by K-medoids. This can be attributed to the fact that both algorithms used a "milder" dataset since the preprocessing allowed them to be less affected by the "noise". Like the K-medoids algorithm, the spectral algorithm produces two low latency dense clusters, the blue and the orange, and a sparse one with high latency.

Figure 32 and Figure 33 present the cluster divisions generated by the Spectral algorithm for cluster counts of 10 and 20. The outcomes closely resemble those of K-means, where in most groups exhibit density and maintain low intra-cluster latency, whereas a few clusters demonstrate latency exceeding the network's average. A slight discrepancy arises in terms of cluster size deviation.

Figure 34 exhibits a similar pattern to the other partition-based algorithms regarding the relationship between the number of clusters and intra-cluster latency. Spectral performs slightly better in the lower cluster count band of the diagram whereas k-means performs a bit better in the higher cluster count band. This could be the sign of resolution lost by the "milder" dataset, but either way the difference in the average cluster latency is negligible.

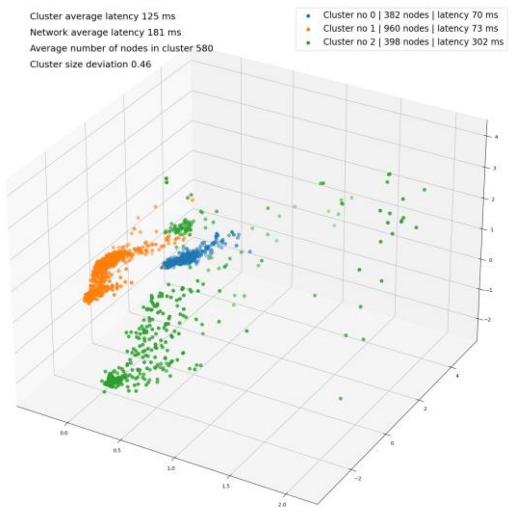

Figure 31 Spectral clustering for 3 clusters on the king latency dataset

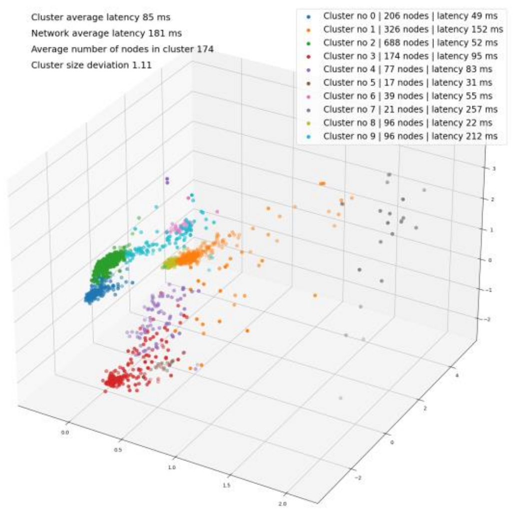

Figure 32 Spectral clustering for 10 clusters on the king latency dataset

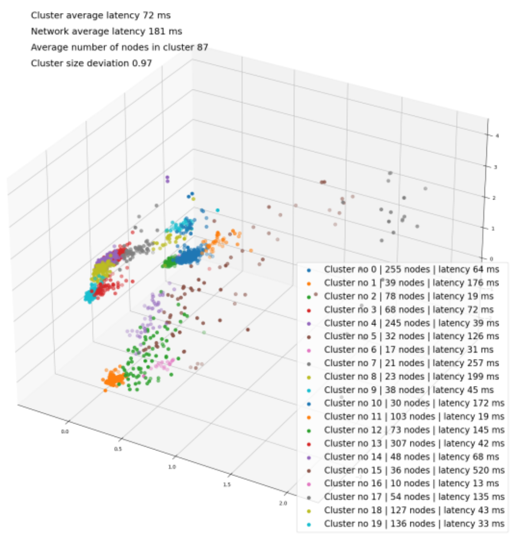

Figure 33 Spectral clustering for 20 clusters on the king latency dataset

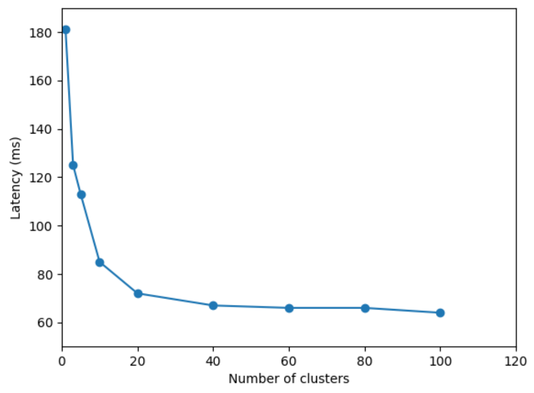

Figure 34 Intra-cluster latency as a function of the number of clusters for the Spectral algorithm

3.2.5. DBSCAN

The DBSCAN [22] algorithm automatically calculates the number of clusters; thus, the final cluster count cannot be directly controlled. To produce comparable results to the clustering algorithms the epsilon and minimum samples parameters are used to indirectly control the cluster count. Recall that epsilon is the distance representing the radius of the node neighborhood, while minimum samples is the minimum number of nodes within a cluster.

By adjusting the epsilon value to 1.500.000 and varying the minimum samples parameter between 40 and 90 the DBSCAN algorithm manages to partition the dataset in 3 and 4 clusters.

However, comprehending the concept of radius in such a high-dimensional space remains a challenge. Although slightly distinct minimum sample values yielded similar clustering outcomes, larger deviations from the parameters resulted in instances of over-clustering and under-clustering. Nodes identified as outliers by the algorithm were treated as a distinct cluster on their own.

As depicted in Figure 35, the algorithm identified two distinct clusters: the orange and the green. The orange cluster contains most nodes and displays pockets of density, resulting in lower latency. The green cluster exhibits a more uniform density distribution. Notably, the blue cluster contains the outliers, contributing to the high intra-cluster latency. For this figure, the minimum samples parameter was set to 40.

Figure 36 showcases the outcomes of DBSCAN using a minimum samples parameter set to 90. The algorithm successfully identified three distinct clusters. Among them, the green and orange clusters exhibit denser characteristics, whereas the red cluster displays a sparser arrangement. The outlier cluster has expanded compared to the previous experiment. This expansion can be attributed to DBSCAN's minimum samples criterion, which necessitates samples to have a denser neighborhood to be recognized as central nodes and consequently form clusters around them.

In Figure 37, a similar pattern for intra-cluster latency seems to emerge, akin to the previous algorithms, although the available data are not sufficient to confirm this observation. It was not possible to produce clusters of 10 or 20 nodes by tuning the epsilon and minimum samples parameters of the DBSCAN algorithm. Therefore, the available data for the DBSCAN algorithm are limited and accurate conclusions cannot be drawn for the performance of the algorithm as the number of clusters increases.

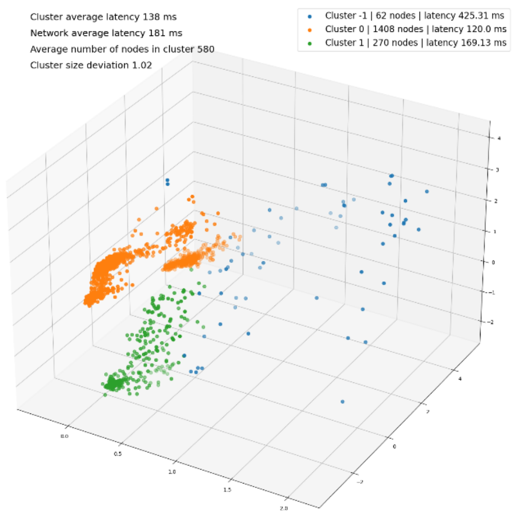

Figure 35 DBSCAN clustering on the king latency dataset epsilon - 1.500.000 and minimum samples - 40

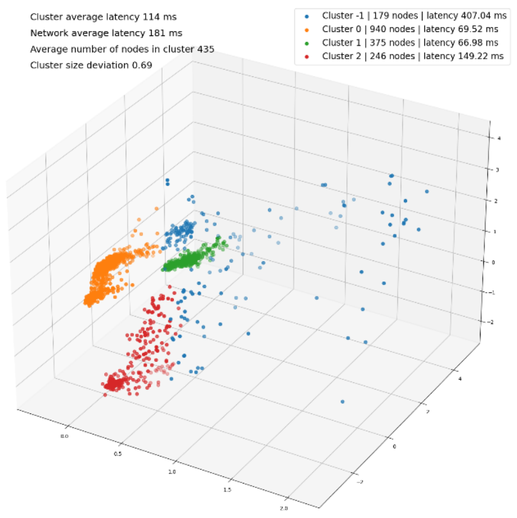

Figure 36 DBSCAN clustering on the king latency dataset epsilon - 1.500.000 and minimum samples – 40

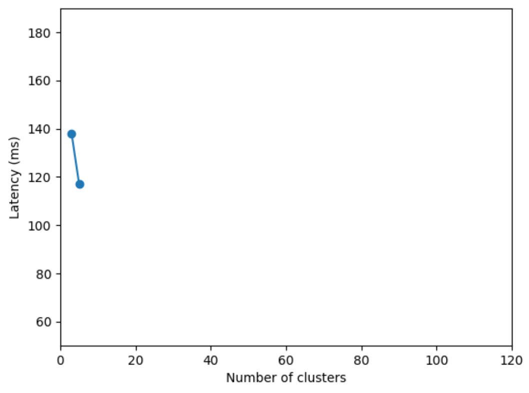

Figure 37 Intra-cluster latency as a function of the number of clusters for the DBSCAN algorithm

3.2.6. Louvain

The Louvain [23] algorithm necessitated the transformation of the King latency matrix into a graph. Like DBSCAN the clusters count cannot be directly set. To overcome this limitation, the cluster count was indirectly influenced through the manipulation of the resolution parameter. By default, the resolution is set to 1. Lowering the resolution leads to the formation of larger communities (clusters), while raising the resolution results in the creation of smaller communities. However, it is important to note that achieving the exact number of clusters as observed with the previous algorithms and surpassing the threshold of 20 communities proved to be challenging.

Figure 38 illustrates the three clusters generated by the Louvain algorithm, where the resolution was set to 0.6. As with the preceding algorithms, a sizable and dense cluster—the blue one—materializes, accompanied by two comparatively smaller clusters. A noteworthy observation is that within the sparsely populated region of the diagram, nodes belong to more than one clusters, unlike the previous algorithms where the nodes with greater distance -located on the right side of the figures- were grouped in one cluster. Furthermore, the average cluster latency is quite low, considering the number of clusters.

Figure 39 and Figure 40 display the results for cluster count of 9 and 17 respectively corresponding to a resolution of 1,2 and 1,4. From an average cluster latency standpoint the clustering seems to be good, though the visualization raises concerns regarding over-clustering in the denser part of the diagram. Especially in Figure 40 clusters no 4, 5, 9 and 15 are indication of higher than optimal value of resolution.

In Figure *41*, a similar pattern to the K-Means, K-Medoids and spectral algorithms emerges, although the available data might not be sufficient to confirm this observation.

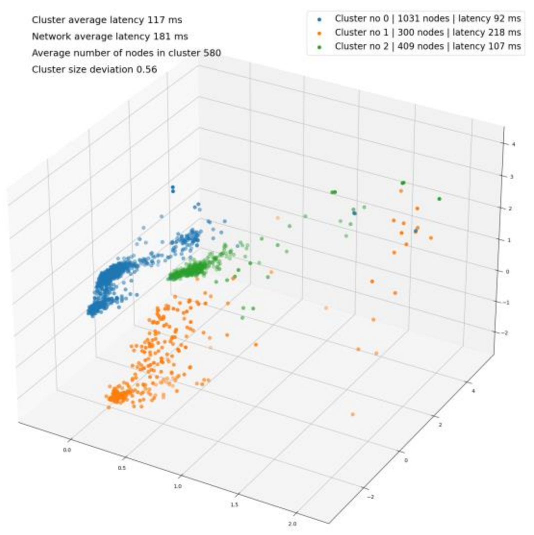

Figure 38 Louvain clustering on the king latency dataset with sresolution of 0,6

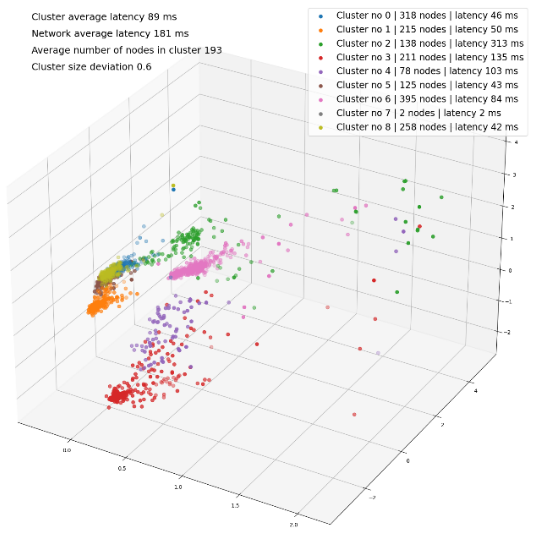

Figure 39 Louvain clustering on the king latency dataset with resolution of 1,2

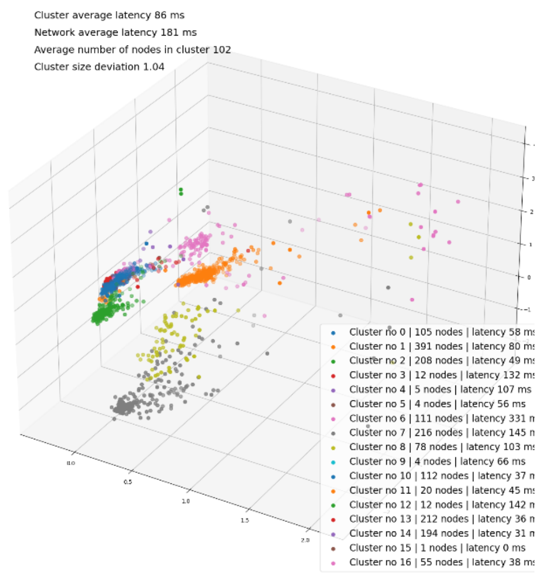

Figure 40 Louvain clustering on the king latency dataset with resolution of 1,4

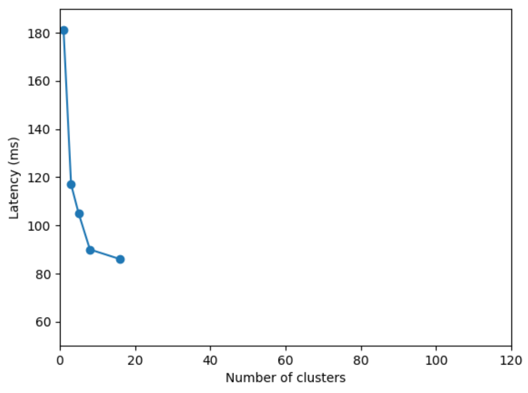

Figure 41 Intra-cluster latency as a function of the number of clusters for the Louvain algorithm

3.2.7. Discussion

Taking into consideration the outcomes derived from the experiments, K-means emerges as the more suitable option for the examined dataset. It demonstrated noteworthy attributes, including speed, adept handling of high-dimensional spaces, robustness against outliers, and at least equal cluster quality compared to other algorithms. Furthermore, K-means has been extensively studied and offers a wide range of variations, including versions tailored for distributed peer-to-peer networks.

K-medoids, sharing similarities with K-means, showcased results that were akin but slightly less favorable. It's worth mentioning the additional computational resources required by K-medoids. On the other hand, DBSCAN struggled to effectively manage the dataset's diverse density variations, Additionally, its capability to detect noise points, which is typically advantageous, turned out to be a drawback in this study. Louvain, while capable of handling fully connected weighted graphs, is better suited in sparsely connected graphs where modularity becomes more evident. In both Louvain and DBSCAN the ability to discover the number of clusters by their own proved troublesome for this case study. Spectral clustering demonstrated comparable outcomes to K-means, yet it came with computational overhead and configuration complexities.

3.3. Hierarchical Kademlia evaluation on PeerNet

The PeerNet evaluation tool is a peer-to-peer network simulator. It can support fine-grained configuration of the DHT protocols and applications while presenting good scalability. PeerNet supports the configuration of the number of peers, the correlated link latencies, the number of requests that are going to be executed during the simulation, the frequency of node failure, as well as the deployment of DHT protocols such as Hierarchical Kademlia. PeerNet generates network traffic traces during simulations, offering insights into the performance of the simulated Peer-to-Peer network. The metrics that are tracked during the simulation are:

- Inter to intra ratio lookup: The ratio of the lookup requests targeting a node inside the cluster vs the requests targeting a node outside the cluster.
- Inter to intra store ratio: The ratio of the store requests targeting a node inside the cluster vs the requests targeting a node outside the cluster.
- K-Bucket size: The average number of entries in the k-buckets of a node.
- Lookup all: The average hops required to look up a key in the network.
- Lookup latency: The latency in milliseconds of looking up a key in the network.
- Store latency: The latency in milliseconds for storing a value in the network.
- Store: The average hops required to store a value in the network

For the simulation, Hierarchical Kademlia [12] was deployed into PeerNet. Hierarchical Kademlia instances were initialized based on the grouping of nodes produced by the clustering algorithms on the king dataset. Additionally, as an evaluation baseline, random clustering and vanilla Kademlia simulations are used. The simulation parameters used for PeerNet and H-Kademlia were:

- H-Kademlia replication factor, K was set to 20.
- H-Kademlia system-wide concurrency parameter, α was set to 3.
- H-Kademlia address space was set to 128 bits.

- Nodes and latencies in Peernet were initialized based on the king matrix dataset.
- PeerNet was tasked with executing 50.000 requests.
- PeerNet's node frequency failure was set to 0 (no churn).

3.3.1. Results

In Figure 42 and Figure 43 inter to intra lookup and store ratio is shown. It can be observed that all the clustering algorithms display the same trend, a proportional increase in inter-cluster requests compared to intra-cluster requests as the cluster count increases. This can be rationalized by considering that a greater number of clusters result in smaller cluster sizes, leading to more keys residing outside the clusters. Consequently, requests tend to exit the clusters to locate the target key.

The steeper increase in Spectral compared to the rest of the algorithms could be attributed to the more balanced, in terms of size, clusters produced as indicated by the cluster size deviation value. This observation gains further support from the behavior of random clustering, which exhibits an even faster increase in ratio due to its clusters being inherently of equal size.

The store ratio is greater than the lookup ratio, indicating that network traffic from lookups is more confined in the cluster compared to traffic from store requests. This difference can be explained by the fact that lookup requests are resolved as soon as the value is found, whereas store requests need to locate the 20 nodes closest to the value. As a result, store requests are more likely to exit the cluster.

In vanilla Kademlia (No cluster) the ratio is zero as there is no distinction between inter and intra requests.

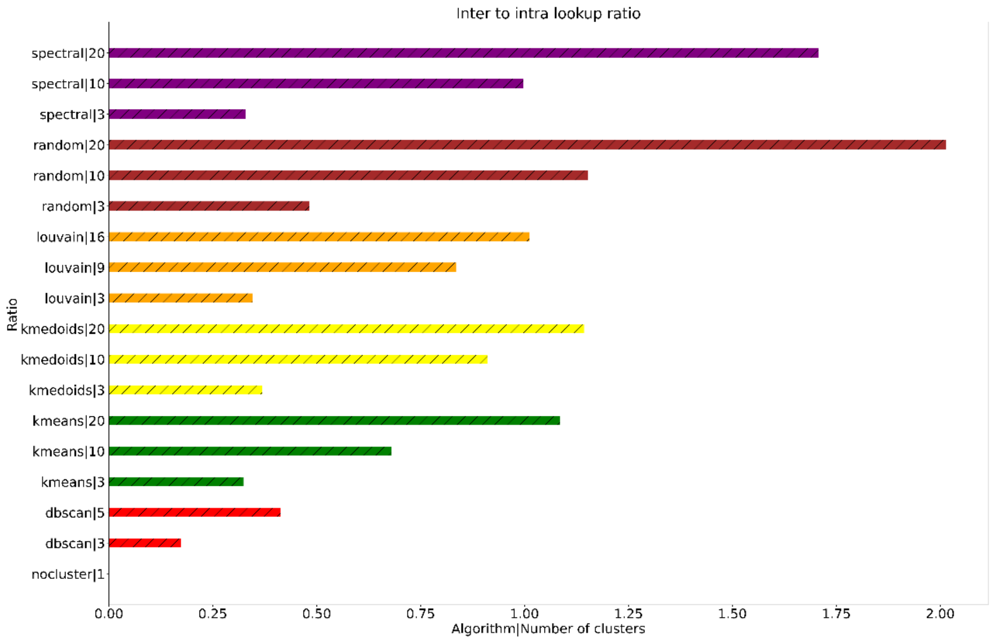

Figure 42 Inter to Intra lookup ratio

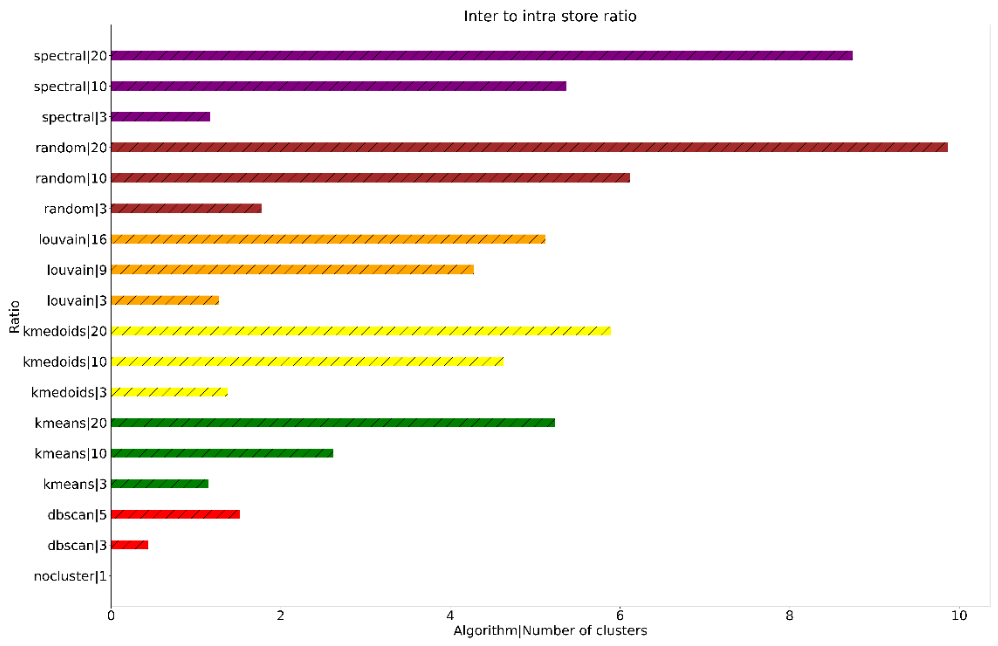

Figure 43 Inter to Intra store ratio

Figure 44 shows that increasing the cluster count results in less records in the node's k-buckets. The trend stands true for all the algorithms. The trend can be justified by the fact that nodes often reject remote peers because a closer local peer exists. Additionally, smaller-sized clusters result in fewer direct insertions of nodes into the k-buckets.

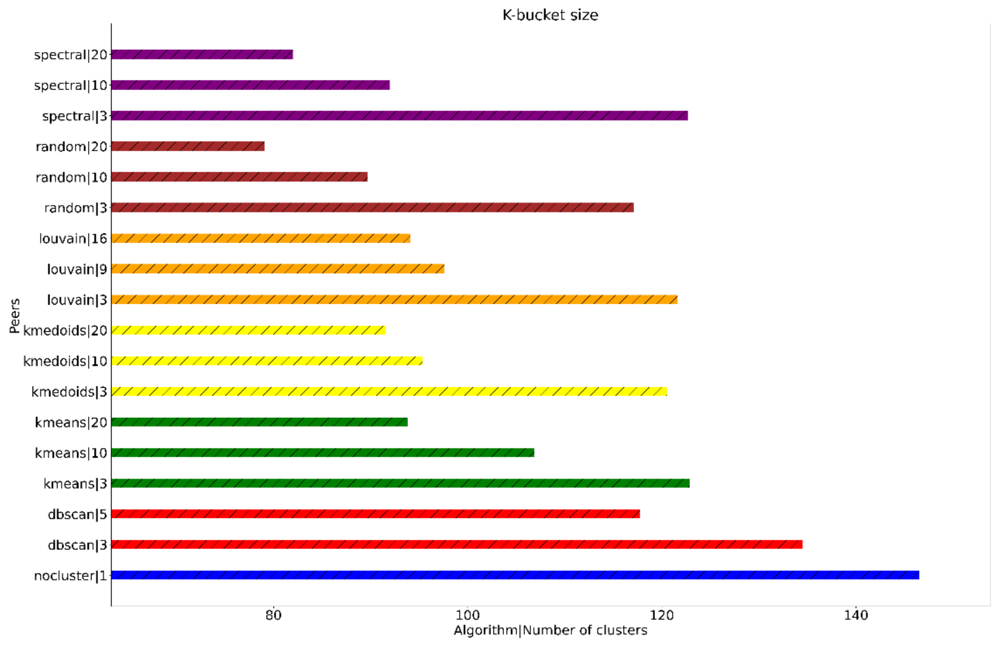

Figure 44 K-Bucket size

Figure 45 and Figure 46 display the differences in latency and hops for a node to complete a look up request in the network for the various combinations of algorithms and clusters counts. Both figures exhibit a consistent trend, indicating that an increase in cluster count results in additional hops. This is due to the more frequent utilization of gateway nodes, resulting in a slight latency increase. All clustering algorithms, except random, achieve lower latency compared to vanilla Kademlia. This is because many requests are resolved within the cluster, where latency tends to be lower than the average network latency. It is worth noting that, in most cases, for a cluster count of three, the H-Kademlia lookup requests are approximately 15% faster compared to the standard Kademlia implementation.

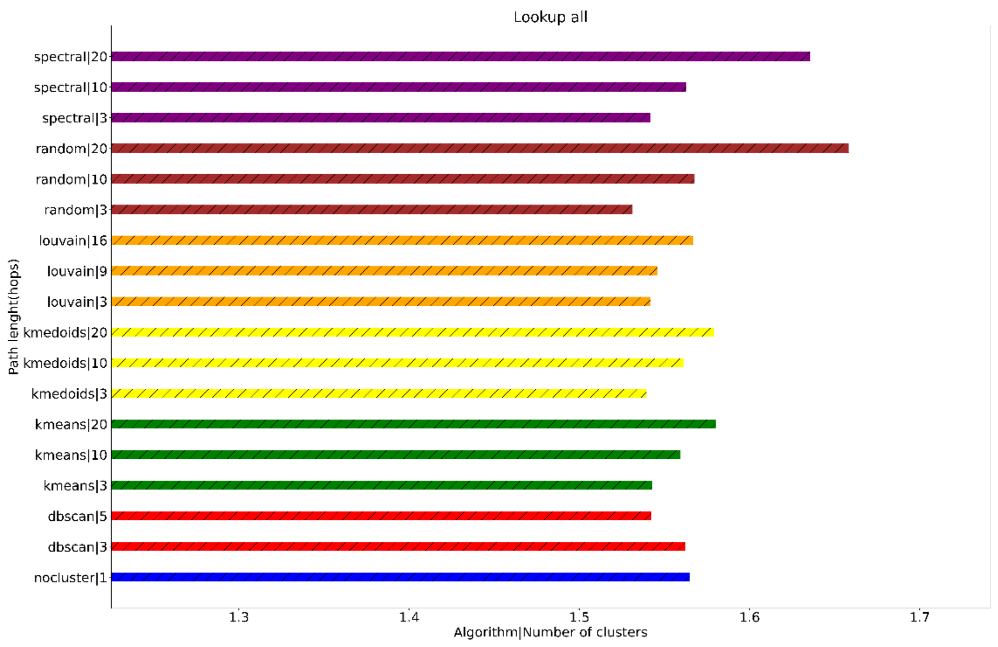

Figure 45 Lookup all

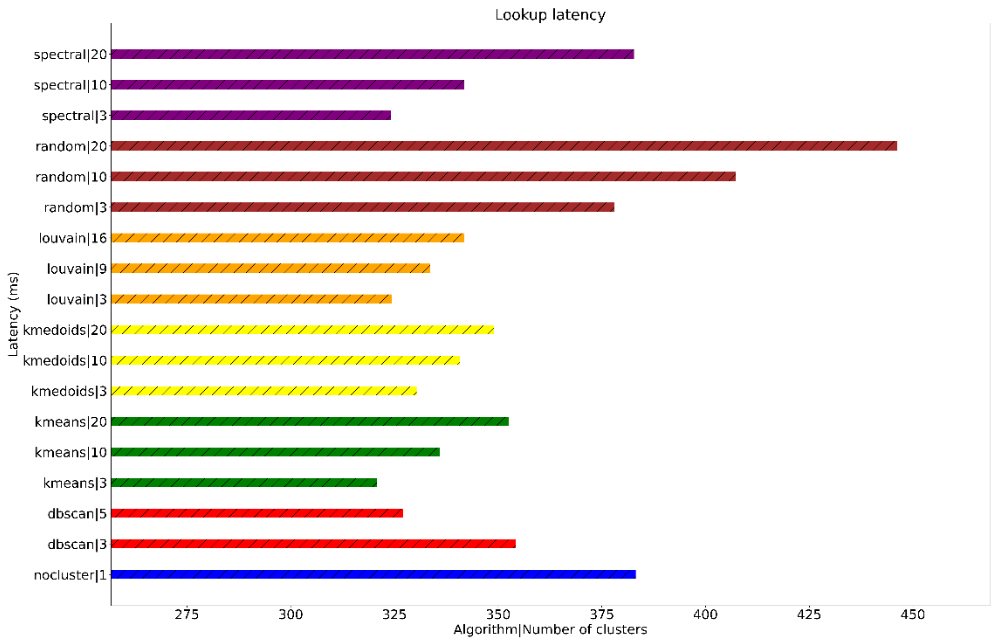

Figure 46 Lookup latency

Figure 48 and Figure 48 display the differences in latency and hops for a node to store a value in the network for the various combinations of algorithms and clusters counts. The impact of clustering setups appears to be milder on the store operation compared to lookup. Store involves locating 20 nodes within the network. When clustering is implemented, local nodes can be easily found, but it takes more time to locate remote nodes, leading to little or no improvement in performance.

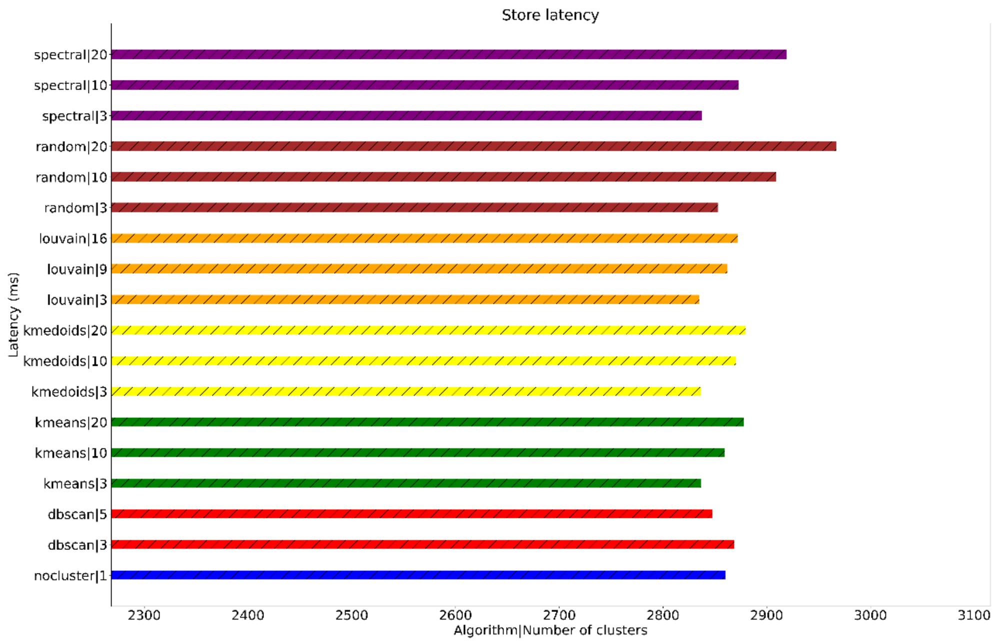

Figure 47 Store latency

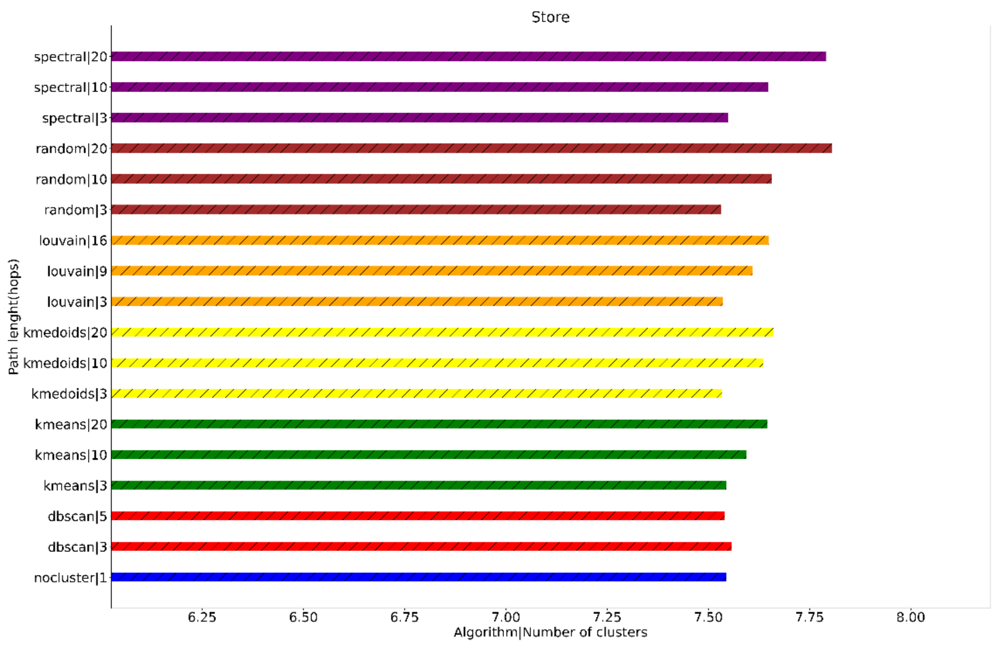

Figure 48 Store hops

4. Conclusions

This thesis explored the capability of popular clustering algorithms to partition the network, while minimizing the intra-cluster latency, and tested the performance of Hierarchical Kademlia on the various clustering setups, using the PeerNet simulator.

In the first part of the study the clustering algorithms K-means, K-medoids, DBSCAN, Spectral, Louvain were used to partition a real-world latency dataset multiple times, each time with a different configuration. The results indicated that these algorithms are able to cluster the nodes of the network effectively based on the objective of intra-cluster latency. K-Means emerged as the preferred option due to its effectiveness in managing high-dimensional data, resilience against outliers, and cluster quality that's at least as good as other algorithms. Additionally, it's worth noting that K-Means has been extensively researched and offers numerous adaptations, including versions designed for distributed peer-to-peer networks.

While the clustering algorithms effectively divided the network into clusters, with most of the clusters exhibiting significantly lower latency compared to the network's average, the performance of Hierarchical Kademlia was not superior to that of the vanilla Kademlia. Lookup requests, according to the simulations, are more confined within the cluster and in certain setups seem to be resolved slightly faster in Hierarchical Kademlia. An interesting thing to note is that a balance lies in the size of the clusters. Small clusters have low intra-cluster latency but cannot resolve many requests locally. Bigger clusters can resolve requests locally but have a higher intra-cluster latency.

Finally, it is worth looking into the concept of a Hybrid Kademlia, where certain nodes would employ Hierarchical Kademlia if they can be grouped into a sufficiently large cluster capable of resolving numerous local requests with lower network latency. Meanwhile, the remaining nodes would continue to utilize the standard version of Kademlia.

Another intriguing area of exploration would involve nodes in the network sharing their latency characteristics, such as their latencies to a set of predefined network endpoints. Each node could then utilize its limited knowledge of the low-dimensional space stored in its routing state to make informed decisions about which cluster would be the most advantageous for it to join.

5. References

[1]. J. Benet, “IPFS: content addressed, versioned, p2p file system,” arXiv preprint arXiv:1407.3561, 2014.

[2]. P. Ganesan, K. Gummadi, and H. Garcia-Molina, “Canon in G major: designing DHTs with hierarchical structure,” in 24th International Conference on Distributed Computing Systems, 2004. Proceedings., 2004, pp. 263–272.

[3]. P. Maymounkov, and D. Mazières, “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric.”

[4]. S. Ratnasamy, et al, "A Scalable Content-Addressable Network." In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 161-172. 2001.

[5]. A. Rowstron, and P. Druschel, “Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.”

[6]. I. Stoica, et al., "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications." In Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 149-160. 2001.

[7]. S. Voulgaris, “PeerNet.” https://github.com/PeerNet/PeerNet.

[8]. D. Xu, and Y. Tian, "A Comprehensive Survey of Clustering Algorithms." ACM Computing Surveys (CSUR) 45.3 (2013): 42.

[9]. Wikipedia, “Cluster Analysis.” Retrieved from https://en.wikipedia.org/wiki/Cluster_analysis

[10]. S.P. Lloyd (1982). Least squares quantization in PCM. IEEE Transactions on Information Theory, 28(2), 129-137

[11]. L. Kaufman, and P.J. Rousseeuw (1987). Clustering by means of medoids. In Y. Dodge (Ed.), Statistical Data Analysis Based on the L1-Norm and Related Methods (pp. 405-416). North-Holland.

[12]. Y. Thomas, et al., Peer Clustering for the InterPlanetary File System,” ACM Future of Internet Addressing and Routing (FIRA) Workshop, 2023.

[13]. J. Shi, and J. Malik, "Normalized cuts and image segmentation." IEEE Transactions on Pattern Analysis and Machine Intelligence 22.8 (2000): 888-905.

[14]. M. Ester, H.P. Kriegel, J. Sander, and X. Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In KDD'96 Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (pp. 226-231).

[15]. V.D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre (2008). Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment, 2008(10), P10008.

[16]. M. S. Artigas, P. G. Lopez, and A. F. Skarmeta, “A Comparative Study of Hierarchical DHT Systems,” in 32nd IEEE Conference on Local Computer Networks (LCN 2007), 2007, pp. 325–333.
[17]. L. Kaufman, and P.J. Rousseeuw (1987). Clustering by means of medoids. In Y. Dodge (Ed.), Statistical data analysis based on the L1-norm and related methods (pp. 405-416). North-Holland.
[18]. L. Kaufman, and P.J. Rousseeuw (1990). Finding Groups in Data: An Introduction to Cluster Analysis. Wiley.
[19]. R.T. Ng, and J. Han (2002). Clarans: A method for clustering objects for spatial data mining. IEEE Transactions on Knowledge and Data Engineering, 14(5), 1003-1016.
[20]. M. Ankerst, M.M. Breunig, H.P. Kriegel, and J. Sander (1999). OPTICS: Ordering Points To Identify the Clustering Structure. In ACM SIGMOD Record, 28(2), 49-60.
[21]. D. Comaniciu, and P. Meer (2002). Mean Shift: A Robust Approach Toward Feature Space Analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(5), 603-619.
[22]. Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.
[23]. A.A. Hagberg, D.A. Schult and P.J. Swart, “Exploring network structure, dynamics, and function using NetworkX”, in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008
[24]. K.P. Gummadi, S. Saroiu, and S.D. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. Department of Computer Science and Engineering, University of Washington, Seattle, WA, USA, 98195-2350.