K-Means Extensions for Clustering Categorical Data on Concept Lattice

Mohammed Alwersh*, László Kovács
Department of Information Technology, University of Miskolc, Miskolc, Hungary

***Abstract*—Formal Concept Analysis (FCA) is a key tool in knowledge discovery, representing data relationships through concept lattices. However, the complexity of these lattices often hinders interpretation, prompting the need for innovative solutions. In this context, the study proposes clustering formal concepts within a concept lattice, ultimately aiming to minimize lattice size. To address this, The study introduces introduce two novel extensions of the k-means algorithm to handle categorical data efficiently, a crucial aspect of the FCA framework. These extensions, namely K-means Dijkstra on Lattice (KDL) and K-means Vector on Lattice (KVL), are designed to minimize the concept lattice size. However, the current study focuses on introducing and refining these new methods, laying the groundwork for our future goal of lattice size reduction. The KDL utilizes FCA to build a graph of formal concepts and their relationships, applying a modified Dijkstra algorithm for distance measurement, thus replacing the Euclidean distance in traditional k-means. The defined centroids are formal concepts with minimal intra-cluster distances, enabling effective categorical data clustering. In contrast, the KVL extension transforms formal concepts into numerical vectors to leverage the scalability offered by traditional k-means, potentially at the cost of clustering quality due to oversight of the data's inherent hierarchy. After rigorous testing, KDL and KVL proved robust in managing categorical data. The introduction and demonstration of these novel techniques lay the groundwork for future research, marking a significant stride toward addressing current challenges in categorical data clustering within the FCA framework.**

Keywords*—Clustering algorithms; categorical data; k-means; cluster analysis; formal concept analysis; concept lattice

I. Introduction

The technique of data clustering involves an unsupervised classification method that aims to group a set of unlabeled objects into meaningful clusters based on their similarities and differences. This process requires objects within the same cluster to exhibit high similarity, while objects in different clusters should have significant differences. Similarities and dissimilarities between objects are evaluated by considering attribute values that describe the objects, often using distance measures. Typically, objects can be represented as vectors in a multidimensional space, with each dimension representing a feature. When numerical features describe objects, geometric distance measures such as Euclidean or Manhattan distance can be used to define their similarity. However, these distance measures are unsuitable for categorical data, including values like gender or location. In recent years, there has been increasing interest in clustering categorical data [1-3]. For categorical data, a comparison measure, rather than a distance measure, is commonly used [4]. However, this metric does not differentiate between different attribute values since it only measures equality between pairs of values [5].

The widely acclaimed k-means algorithm [6] excels in simplicity and efficiency, particularly for large numerical datasets. However, it is restricted by its inability to handle categorical data directly. To overcome this limitation, adaptations such as the k-modes [3], k-representative [7], and k-centers algorithms [8] have been introduced. The k-modes algorithm uses simple matching similarity measures and substitutes "means" with "modes" for cluster centers. This modification, however, can result in multiple modes within a cluster, which can influence the algorithm's performance. To navigate this, the k-representative algorithm proposed in [9], presents "cluster centers" uniquely suited to categorical data. In this approach, the defining attribute of a cluster's representative stems from the diversity of categorical values within the cluster itself [9]. Although these adaptations of the k-means algorithm share a similar clustering process, they define "cluster center" and "similarity measure" differently for categorical data, which could potentially result in information loss when categorical data are directly transformed into vector space.

This paper introduces two novel extensions of the k-means algorithm for clustering categorical data: K-means Dijkstra on Lattice (KDL) and K-means Vector on Lattice (KVL). Existing methods such as k-means and its variants are restricted by their inability to adequately handle categorical data, often requiring data transformation techniques that can result in information loss. Additionally, they struggle with representing complex, hierarchical relationships inherent in the data. Our proposed methods, KDL and KVL, aim to overcome these limitations by focusing on learning and integrating the representation of categorical values based on their inherent graph structure. This approach not only preserves the richness of the categorical data but also effectively captures the potential similarity and hierarchical relationships between the values. KDL utilizes Formal Concept Analysis to construct a graph, with nodes symbolizing formal concepts and edges representing hierarchical relationships [10]. A customized Dijkstra algorithm identifies the shortest path between formal concepts, replacing the Euclidean distance from traditional k-means. The centroids are those formal concepts with the least sum of intra-cluster distances. This method accurately identifies data patterns and relationships, providing insights often missed by conventional vector representations. KVL, the second extension, transforms formal concepts into numerical concept description vectors and applies traditional k-means. It evaluates concept similarity, groups related concepts, and positions each

cluster's center as the mean of its concept description vectors. This approach excels in scalability by converting categorical data into numerical formats, enabling efficient analysis of larger datasets. Both KDL and KVL significantly advance our understanding of complex datasets by providing a unique perspective on clustering categorical data. These novel extensions of the k-means algorithm pave the way for further research in this field. The primary contributions of this study are the introduction of KDL and KVL as innovative extensions of the k-means algorithm, the comparative evaluation of these methods against existing algorithms, and the laying of groundwork for future advancements in clustering categorical data within the Formal Concept Analysis frameworks.

The remainder of this paper is structured as follows: Section II provides an overview of Formal Concept Analysis (FCA), elucidating key terminologies and notions. Section III delves into Dijkstra's Algorithm and its variations employed for addressing shortest-path problems. Section IV then explores the k-means algorithm and its extended versions tailored for categorical data. This segues into Section V, where the proposed clustering methodologies are thoroughly discussed. The experimental results and their implications are exhibited in Section VI. The paper concludes in Section VII, summarizing the study's main points and potential future directions.

II. FCA: Key Terminology and Notions

Formal Concept Analysis (FCA) emerged as a distinctive mathematical field in the early 1980s. Central to its application are particular diagrams known as line or Hasse diagrams, which are utilized to depict information via concept lattices [10]. To enhance the comprehension of the study, the study provide a succinct overview that delineates the critical concepts and definitions within FCA, supplemented by an easily digestible example. The terms and foundations of FCA discussed in this paper are grounded in the work of [11].

Definition 1. Formal Context: A formal context is characterized as a tripartite structure (G, M, I), where G represents a collection of objects, M stands for a set of attributes, and $I \subseteq G \times M$, embodies the incidence relationship between G and M. For each object g $\in G$ and attribute $m \in M$, a binary relation gIm $((g,m) \in I)$ signifies that object g possesses attribute m. Typically, a cross table illustrates a formal context, where rows depict the object names and columns exhibit the attribute names. The presence or absence of a cross demonstrates the existence or non-existence of an incidence relationship between G and M, respectively. This context is also referred to as a binary context, as demonstrated in Table I.

Expanding on Table I shows a compact formal context in which the object set G includes $\{o_1, o_2, o_3, o_4, o_5\}$, and the attribute set M comprises $\{a_1, a_2, a_3, a_4, a_5, a_6\}$. The cross (x) at the intersection of the object g row and the attribute m column indicates that the object g possesses the attribute m, while its absence signifies a lack of relationship between g and m. For instance, within this formal context, object o_1 is associated with the attributes $\{a_2, a_6\}$.

Definition 2. Derivation Operators: In a formal context (G, M, I), derivation operators $\uparrow = 2^G \to 2^M$, $\downarrow = 2^M \to 2^G$, are established for every subset of objects A within G and a subset of attributes B within M. They are precisely defined as $A^{\uparrow} = \{m \in M \mid \forall g \in A: (g,m) \in I\}$, $B^{\downarrow} = \{g \in G \mid \forall m \in B: (g,m) \in I\}$. The upward operator $A^{\uparrow}$ represents the collective attributes shared by all objects in A, and the downward operator $B^{\downarrow}$ comprises all objects that possess all attributes in B.

These derivation operators $A^{\uparrow}$ and $B^{\downarrow}$ are also referred to as A' and B'. For example, within the context provided in Table I, it can be easily discerned that:

$$\{o_1\}^{\uparrow} = \{a_2, a_6\}$$

$$\{o_1, o_3\}^{\uparrow} = \{a_6\}$$

$$\{a_1\}^{\downarrow} = \{o_3\}$$

$$\{a_2, a_3\}^{\downarrow} = \{o_5\}$$

Definition 3. Formal Concepts: In a provided context denoted as k=(G, M, I), a formal concept is recognized as a pair (A, B), where A, a subset of G, is referred to as the 'extent' part of the formal concept (A, B), while B, a subset of M, is known as the 'intent' part of the formal concept (A, B), provided that $A' = B, B' = A$. For instance, considering the formal context illustrated in Table I, the pair ({o1, o3}, {a6}) emerges as a formal concept, where {o1, o3} represents the extent part and {a6} embodies the intent part.

Definition 4. Concept Lattices: Formal concepts can be arranged based on the subconcept-super concept relation $\leq$, expressed as follows: $(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2$ (or equivalently $B_1 \subseteq B_2$) , where (A_1, B_1) is a subconcept (more specific) and (A_2, B_2) is a super concept (more general). Within a formal context K, the assembly of all formal concepts, in conjunction with the partial order $\leq$, generally defines a complete lattice, referred to as a concept lattice [10]. The notation $\mathcal{B}(G, M, I)$ signifies the concept lattice derived from a formal context (G, M, I).

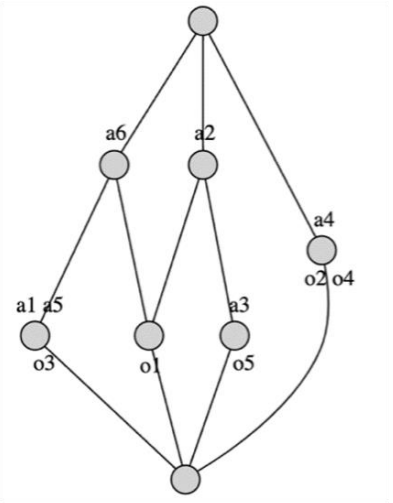

Fig. 1. Concept lattice corresponding to the formal context from Table I.

By the initial part of the core theorem on concept lattices [10], a concept lattice $\mathcal{B}(G,M,I)$ is identified as a complete lattice where the infimum and supremum are present for any arbitrary set. This is represented as $(A_1,B_1) \wedge (A_2,B_2) = (A_1 \cap B_2, (B_1 \cup B_2)'')$ and $(A_1,B_1) \vee (A_2,B_2) = ((A_1 \cup A_2)'', B_1 \cap B_2)$.

TABLE I. FORMAL CONTEXT

Objects/ Attributes	a_1	a_2	a_3	a_4	a_5	a_6
o_1		x				x
o_2				x		
o_3	x				x	x
o_4				x		
o_5		x	x			

The concept lattice derived from the formal context showcased in Table I is portrayed in Fig. 1 through a line diagram. The lattice is constituted by formal concepts, which are generated depending on the formal context and their relationship defined by the subconcept-superconcept paradigm [11]. Every node in the line diagram symbolizes a formal concept. These formal concepts in the diagram can be categorized into object concepts, described as $(\{g\}'', \{g\}')$, or attribute concepts, defined as $(\{m\}', \{m\}'')$, indicating the concepts tied to objects or attributes, respectively. An object concept is represented as $\gamma(g)$, while an attribute concept is signified as $\mu(m)$. When labeling $\mathcal{B}(G,M,I)$, The convention adhered to in this study specifies that each object g gets its label g for the corresponding formal concept $\gamma(g)$, and each attribute m is labeled as m for the formal concept $\mu(m)$. Certain concept nodes carry objects below them, while others have attributes above them. More often than not, object labeling is placed underneath a node, while attribute labeling is positioned above it in a line diagram. It's worth highlighting that not every concept within a particular context is necessarily an object or an attribute concept. Any concept has the potential to be an object concept, an attribute concept, a blend of the two, or neither [12,13].

III. DIJKSTRA'S ALGORITHM AND ITS VARIATIONS FOR SHORTEST PATH PROBLEMS

Dijkstra's algorithm was introduced in 1959 by Dutch computer scientist Edsger W. Dijkstra; Dijkstra's algorithm serves as a leading method for finding optimal paths. This algorithm, notable for its efficiency and effectiveness, has found broad application across various fields. One crucial application is in routing through the Internet's vast interconnected web, determining the shortest paths between network nodes. The algorithm operates on a directed weighted graph denoted as $G = \{V, E\}$, where V is the set of vertices and E the set of edges. Each edge e has an associated weight, Edge-Cost (e), representing the traversal cost. It's essential for the algorithm's efficiency that all weights are non-negative. By applying Dijkstra's algorithm, one can efficiently ascertain the shortest path from a source vertex to every other vertex in the graph. Such information finds applications in diverse fields, including transportation, logistics, and network communication [14].

The algorithm proceeds by assigning temporary and visited states to vertices and updating their distances from the source. It marks vertices as visited, updating the temporary vertices' distances as it progresses. After visiting all vertices, the algorithm terminates. However, it has limitations: it involves a blind search, leading to inefficient resource use and longer operations. Moreover, it cannot handle negative edges, leading to potential inaccuracies in shortest path calculations [15]. The efficiency of Dijkstra's algorithm can be expressed using Big-O notation. The complexity depends on the number of vertices ($|V| = n$) and the updates for priority queues ($|E|$). Different data structures can be used for the priority queue, resulting in different complexities:

- If Fibonacci heap is used, the complexity is $O(|E| + |V| \log |V|)$, with DeleteMin operations taking $O(\log |V|)$.
- If a standard binary heap is used, the complexity is $O(|E| \log |E|)$, with $|E|$ updates for the standard heap.
- If a priority queue is used, the complexity is $O(|E| + |V|^2)$, where the $|V|^2$ term arises from $|V|$ scans of the unordered set New Frontier to find the vertex with the least sDist value.

There are numerous variants of Dijkstra's algorithm, each addressing specific needs and modifications. Among the most well-known is the Bellman-Ford algorithm [16], which caters to negative-weighted graphs. The Floyd-Warshall algorithm [17], employs dynamic programming to find shortest paths, accommodating both positive and negative edge weights. The Johnson algorithm uses the Bellman-Ford algorithm to re-weight the graph, eliminating negative weights, then applies Dijkstra's algorithm, effectively reducing execution time for sparse graphs [18]. Lastly, the A* algorithm extends Dijkstra's by combining breadth-first search and heuristic methods, potentially increasing speed but failing to ensure absolute accuracy [14]. Depending on the problem's requirements and characteristics, one can choose the most suitable variant to obtain optimal results. Dijkstra's algorithm is utilized to measure distance in the particular use case of clustering categorical data within a concept lattice. This approach allows us to interpret the intricate structures of the lattice effectively, paving the way for efficient clustering and insight generation.

IV. K-MEANS ALGORITHM AND ITS EXTENSIONS FOR CATEGORICAL DATA

The k-means algorithm [6], is a widely used partitional or non-hierarchical clustering method. Given a set D of N numerical data objects, a natural number k (where k is less than or equal to N), and a distance measure between objects, the algorithm aims to find a partition of D into k non-empty and disjoint clusters. The objective is to minimize the sum of squared distances between each data object and its assigned cluster center.

Mathematically, the k-means algorithm can be formulated as an optimization problem. Let $U = [u_{i,j}]$ be the partition matrix, where $u_{i,j}$ is a binary indicator variable representing whether object X_i belongs to cluster S_j. Let $Z = \{Z_1, Z_2, \ldots, Z_k\}$ be the set of cluster centers. The squared

Euclidean distance $dis\ (\cdot,\cdot)$ between two objects X_i and Z_j is used as the distance measure [19].

The objective function to be minimized is given by P problem, as specified in Eq. (1):

$$P(U,Z) = \sum_{j=1}^{k}\sum_{i=1}^{N} u_{i,j} dis(X_i, Z_j) \quad (1)$$

The objective function presented in Eq. (1) is to be minimized, and this operation is constrained by specific conditions as outlined in Eq. (2):

$$\sum_{j=1}^{k} u_{i,j} = 1,\ 1 \le i \le N. \quad (2)$$

$$u_{i,j} \in \{0,1\}, 1 \le i \le N, 1 \le j \le k$$

where $U = [u_{i,j}]_{N\times k}$ represent a partition matrix, where each element $u_{i,j}$ equals 1 if object X_i belongs to cluster S_j and 0 otherwise. $Z = \{Z_1, Z_2, \ldots, Z_k\}$ denotes the set of cluster centers. The function $dis\ (\cdot, \cdot)$ calculates the squared Euclidean distance between two objects.

The k-means algorithm follows a four-step process until the objective function $P(U,Z)$ converges to a local minimum:

- Initialize cluster centers as $Z^0 = Z_1^0, \ldots, Z_k^0$, and set $t = 0$.
- With fixed cluster centers Z^t, solve $P(U, Z^t)$ to obtain partition matrix U^t. Each object X_i is assigned to the cluster with the nearest cluster center.
- With fixed partition matrix U^t, generate updated cluster centers Z^{t+1} to minimize P (U^t, Z^{t+1}). The new cluster centers are computed as the mean of the objects within each cluster.
- If convergence is reached or a stopping criterion is satisfied, output the final result and terminate. Otherwise, increment t by 1 and go back to step 2.

By iteratively updating the partition matrix and the cluster centers, the k-means algorithm converges to a local minimum, providing an effective approach for clustering numerical data objects. As mentioned above, the K-means algorithm [3] is primarily designed for numerical data. However, the algorithm's direct application to categorical data presents challenges due to the need for a natural numerical representation for categorical variables. Several extensions and adaptations of the K-means algorithm have been proposed to overcome this limitation to enable its use with categorical data. One widely used extension is the K-modes algorithm [3]. Unlike the original K-means algorithm, which relies on the Euclidean distance, the K-modes algorithm utilizes a dissimilarity measure specifically tailored for categorical variables. Instead of computing distances based on coordinates in a multidimensional space, the K-modes algorithm uses a simple matching distance measure and defines "cluster centers" as modes. In the K-modes algorithm, the dissimilarity between two categorical objects, X and Y described by M categorical attributes, is computed by counting the total number of matching attribute values between the two objects. The dissimilarity measure is defined as shown in Eq. (3):

$$dis(X,Y) = \sum_{i=1}^{M} \delta(X_i, Y_i) \quad (3)$$

where,

$$\delta(X_i, Y_i) = \begin{cases} 0 & if\ X_i = Y_i \\ 1 & if\ X_i \neq Y_i \end{cases}$$

For a cluster of categorical objects, $\{X_1, \ldots, X_N\}$, where $X_i = (x_{i1}, \ldots, x_{iM})$ and $1 \le i \le N$, the K-modes algorithm defines the mode $Z = (o_1, \ldots, o_M)$ of the cluster by assigning o_m, $1 \le m \le M$, as the most frequently appearing value within $\{x_{1m}, \ldots, x_{Nm}\}$. The authors in [3] introduced these modifications to develop the K-modes algorithm, which resembles the K-means method for clustering categorical data. However, it should be noted that the mode of a cluster is not generally unique, which introduces instability into the algorithm depending on the selection of modes during the clustering process.

The k-Representative algorithm [7] is a further extension of the K-means algorithm which incorporates the idea of cluster representatives. Rather than utilizing modes as cluster centers, this concept was brought forward by [7], defining representatives in the following manner. Let's take a cluster S, comprised of categorical objects, expressed as $S = \{X_1, \ldots, X_p\}$. Each object X_i can be represented as $(x_{i1}, \ldots, x_{iM})$ with the condition $1 \le i \le p$. For each attribute m ranging from 1 to M, O_m^S symbolizes the set of categorical values derived from $x_{1m}, \ldots, x_{pm}$ within the cluster, that means O_m^S denotes the set of unique categorical values for attribute m within a specific cluster S .This is essentially a collection of all distinct categories for attribute m across all objects in the cluster S. For instance, suppose a cluster S consisting of the following three objects:

- Object 1: (Red, Circle, Large)
- Object 2: (Blue, Circle, Medium)
- Object 3: (Red, Square, Medium)

Upon examining the attribute 1 (color), then O_1^S would be {Red, Blue}, since these are the unique color values within cluster S. Similarly, O_2^S for attribute 2 (shape) would be {Circle, Square}, and O_3^S for attribute 3 (size) would be {Large, Medium}. The representative of cluster S, denoted by $Z_S = (z_1^S, \ldots, z_M^S)$, is characterized as:

$$z_m^S = \{(o_{ml}, fS(o_{ml})) \mid o_{ml}\ is\ an\ element\ of\ O_m^S\} \quad (4)$$

Here, , $fS(o_{ml})$ signifies the proportional frequency of category o_{ml} within cluster S. This is computed by dividing the number of objects in S, possessing the category o_{ml} for the m^{th} attribute, by the total count of objects in S. This is represented by $\#S(o_{ml})$ and p, respectively:

$$S(o_{ml}) = \#S(o_{ml})\ /\ p \quad (5)$$

In essence, each z_m^S is a distribution over O_m^S, defined by the proportional frequencies of categorical values inside the cluster. The k-Representative algorithm employs a simple matching measure, δ, to determine the dissimilarity between an object $X = (x_1, \ldots, x_M)$ and the representative Z_S . The dissimilarity $dis(X, Z_S)$ is characterized as defined in Eq. (6):

$$dis(X, Z_S) = \sum_{m=1}^{M} \sum_{o_{ml} \in O_m^S} fS(o_{ml}) \cdot \delta(x_m, o_{ml}) \quad (6)$$

In this context, the dissimilarity $dis(X, Z_S)$ is predominantly influenced by two key factors: the proportional frequencies of categorical values within the cluster and the basic matching of these categorical values. The proportional frequencies reflect the relative importance and prevalence of different categories within the cluster, while the matching mechanism assesses the similarity between the categorical values of the data object X and the representative center Z_S. By considering both the proportional frequencies and the matching aspect, the dissimilarity measure captures the distinctive characteristics and relationships of categorical variables within the cluster. To illustrate this, let's continue with the example above and consider a new object 4: (Blue, Circle, Small). The dissimilarity between object 4 and the representative Z_S of the cluster S would be calculated as follows: The representative Z_S for the previous example cluster would be:

- For attribute 1 (Color): {('Red', 0.67), ('Blue', 0.33)}
- For attribute 2 (Shape): {('Circle', 0.67), ('Square', 0.33)}
- For attribute 3 (Size): {('Large', 0.33), ('Medium', 0.67)}

Let's calculate the dissimilarity between Object 4 and the representative Z_S using Eq. (6). For each attribute, the contribution to the overall dissimilarity is individually calculated:

For attribute 1 (Color):

o_{ml} in O_1^S: {('Red'), ('Blue')}

$fS(o_{ml})$: {0.67, 0.33}

δ ('Blue', 'Red') =1, δ ('Blue', 'Blue') =0

Contribution for attribute 1:

$fS('\text{Red}') \cdot$ δ ('Blue', 'Red') + $fS('\text{Blue}') \cdot$ δ ('Blue', 'Blue') = $0.67 \cdot 1 + 0.33 \cdot 0 = 0.67$

For attribute 2 (Shape):

o_{ml} in O_2^S: {('Circle'), ('Square')}

$fS(o_{ml})$: {0.67, 0.33}

δ ('Circle', 'Circle') = 0, δ ('Circle', 'Square') =1

Contribution for attribute 2:

$fS('\text{Circle}') \cdot$ δ ('Circle', 'Circle') + $fS('\text{Square}') \cdot$ δ ('Circle', 'Square')= $0.67 \cdot 0 + 0.33 \cdot 1 = 0.33$

For attribute 3 (Size):

o_{ml} in O_3^S: {('Large'), ('Medium')}

$fS(o_{ml})$: {0.33, 0.67}

δ ('Small', 'Large') = 1, δ ('Small', 'Medium') =1

Contribution for attribute 2:

$fS('\text{Large}') \cdot$ δ ('Small', 'Large') + $fS('\text{Medium}') \cdot$ δ ('Small', 'Medium') = $0.33 \cdot 1 + 0.67 \cdot 1 = 1$

Finally, sum up the contributions from all attributes:

$$dis(Object4, V_S) = 0.67 + 0.33 + 1 = 2$$

Therefore, the dissimilarity between object 4 and the representative Z_C of cluster S is 2. Based on this dissimilarity, the cluster assignment of object 4 can be determined by comparing its dissimilarity with other cluster representatives. Object 4 will be assigned to the cluster with the lowest dissimilarity, indicating the cluster it is most similar to.

Several extensions have been proposed to tackle specific issues related to categorical data clustering. The k-Centers algorithm [8] is an extension of the K-means algorithm, defining the cluster center as a set of probability distributions estimated using a kernel density estimation method. Dissimilarities between data objects and cluster centers are calculated using indicator vectors and squared Euclidean distance, providing an effective method for clustering categorical data while preserving the key principles of K-means. Other extensions have been proposed to address specific challenges associated with clustering categorical data. These extensions include fuzzy K-modes [20], scalable K-modes [21], and probabilistic K-modes [22], among others. Fuzzy K-modes permit soft assignments, allowing data points to belong to multiple clusters with various degrees of membership. Scalable K-modes enhance computational efficiency for large-scale categorical datasets, while probabilistic K-modes incorporate probabilistic models for uncertainty in cluster assignments, giving a more refined view of cluster membership.

The development of these extensions demonstrates the ongoing efforts to adapt the K-means algorithm for categorical data clustering. These approaches not only consider the unique characteristics of categorical variables but also address challenges such as missing data, scalability, and uncertainty in cluster assignments. The availability of these extensions expands the applicability of the K-means algorithm, allowing it to be effectively utilized in a wide range of domains where categorical data analysis is prevalent.

V. Proposed Clustering Methods

A. K-means Dijkstra on Lattice (KDL)

The K-means Dijkstra on Lattice (KDL) method uniquely merges the structural representation of Formal Concept Analysis (FCA) with the computational efficiency of a customized version of Dijkstra's algorithm. This innovative procedure addresses the distinctive challenges associated with clustering categorical data while duly considering the data's inherent structure. KDL method harmoniously integrates the mathematical rigor of FCA and the algorithmic strength of Dijkstra's approach, crafting a new path in categorical data analysis. The general procedure of KDL as follows:

- Data Conversion to Formal Context: In the initial phase, the categorical data is transformed into a formal context. This context is represented by a binary matrix, where rows correspond to distinct objects, and columns represent various attributes. A value of 1 signifies that a given object belongs to a certain attribute category, and a 0 indicates the lack of such a relationship.
- Formal Concept Derivation: The FCA is then applied to this formal context. It uses the "NextClosure" algorithm

[23], which ensures that all possible formal concepts in the context are generated. These concepts represent significant associations between objects and attributes and capture the underlying structure and dependencies within the data. The hierarchical relationships among these concepts are represented in a lattice structure or graph, which is constructed using the "Ipred" algorithm, modified to suit the needs of the current study. “Ipred” algorithm is very fast for building the Hasse diagram. For more information about this algorithm, refer to [24]. In the quest for an analytical solution to count concepts, Schüt [25] proposed an upper approximation for the count, expressed as $|C| \leq 3/2 \cdot 2^{\sqrt{(|I|+1)}} - 1$, where $|C|$ denotes the number of concepts and $|I|$ represents the number of entries in the formal context. This approximation accommodates not merely the number of objects or attributes but also the overall size of the context, providing a potentially more precise estimate of the concept count [26].

- Assigning Edge Weights: This step is critical for defining the cost of moving between concepts within the lattice. Each transition has an associated cost, which can vary depending on the direction of movement. For example, transitioning from a parent concept to a child concept can be assigned a higher cost (let’s say a cost of 2), than moving from child to parent (let’s say a cost of 1), reflecting the importance of specific transitions in categorical attributes.
- Using Dijkstra's Algorithm for Distance Measurement: The method employs the Dijkstra algorithm to measure distance within the concept lattice. It calculates the minimum cost of the shortest path between two formal concepts in a concept lattice using the weights assigned to the edges. This provides an effective measurement for the optimal path within the concept lattice.
- Calculation and Updating of Cluster Centroids: The centroids of the clusters, which need to be formal concepts themselves, are calculated and continually updated until the values stabilize. The representative centroid of a cluster is the formal concept with the smallest sum of distances to all other concepts within the cluster, minimizing the overall clustering cost function.

In the context of the proposed clustering method utilizing Formal Concept Analysis (FCA) and Dijkstra’s algorithm, an important property arises, for any pair of concepts c_1 and c_2 within the concept lattice, there always exists a path connecting c_1 to c_2. The concept lattice, constructed through FCA, represents the hierarchy of formal concepts derived from the categorical data. The property asserts that no matter which two concepts are chosen within the lattice, there is always a path connecting them. This means there is a sequence of edges to traverse, moving from one concept to another, ultimately leading from c_1 to c_2. By relying on the subconcept-super concept relation ($\leq$) transitivity in the lattice prove that a path exists between any two concepts in a lattice as described in Definition 4 in Section II. Let's consider two concepts $c_1$and c_2 in the lattice. If c_1 and c_2 are directly connected (i.e., $c_1 \leq c_2$ or $c_2 \leq c_1$), then a path exists between them as they are adjacent concepts in the lattice. If c_1 and c_2 are not directly connected, consideration can be given to all concepts that are reachable from c_1 in the lattice. Let's denote this set as $R(c_1)$. Similarly, the set of all concepts reachable from c_2 as can be defined $R(c_2)$. Since the lattice is a partially ordered set, $R(c_1)$ and $R(c_2)$ are subsets of the lattice. Now, let's consider the intersection of $R(c_1)$ and $R(c_2)$, denoted as $R(c_1) \cap R(c_2)$. If their intersection is not empty, it means that there exists at least one concept that is reachable from both c_1 and c_2. Let's denote this concept as c. Since c is in $R(c_1)$, there is a path from c_1 to c. Similarly, because c is within $R(c_2)$, a path leads from c to c_2. Thus, by connecting these two paths, a continuous path is formed from c_1 to c_2.

If $R(c_1) \cap R(c_2)$ is empty, no concepts can be reached from both c_1 and c_2. Nonetheless, establish a pathway between c_1 and c_2 by considering the concepts that are reachable from each and locating a shared concept that functions as an intermediary. This process can be conducted iteratively, broadening the search for concepts reachable from both c_1 and c_2 until a common concept is discovered. By exploiting the transitivity of the subconcept-super concept relationship and considering the accessibility of concepts within the lattice, a path between any two lattice concepts can always be found.

The concept lattice structure inherently guarantees a path between any pair of concepts, rooted in its construction, which encapsulates all potential combinations of objects and attributes as formal concepts. As a result, the lattice forms a connected structure, allowing for traversal from one concept to another through a series of links. This trait is vital for the suggested clustering method, ensuring the computation of the least costly shortest path between any formal concept pair using the Dijkstra-based distance measure. It confirms that all concepts in the lattice can be accessed, and their dissimilarities compared, facilitating precise cluster assignments based on categorical profiles. The clustering method efficiently leverages this connectivity within the concept lattice, capturing hierarchical relationships, semantic constraints, and directional costs embedded in the data. This inherent lattice connectivity aids in delivering meaningful and accurate cluster assignments, ensuring all concepts within the lattice remain interconnected and accessible.

1) Dijkstra-based distance measure description: The Dijkstra-Based Distance Measure is a key component of the K-means Dijkstra on Lattice (KDL) method, serving as a more efficient substitute for the Euclidean distance metric in standard K-means algorithms. The KDL method incorporates Dijkstra's algorithm on a lattice structure generated from categorical data via Formal Concept Analysis (FCA). The algorithm computes the shortest path between two formal concepts in the lattice, considering both cost and path direction. Notably, upward transitions (parent-to-child concept) may incur higher costs than downward transitions (child-to-parent). The algorithm utilizes a min-heap-based priority queue for optimization.

Formally, the Hasse diagram built from a concept lattice $\mathcal{B}(C, <)$, can be represented as $\mathcal{H}(C, E)$, where C is the set of

formal concepts, and E denotes the edges representing the hierarchical relationships among them. The start and end formal concepts are denoted as C_s and C_e, respectively. The cost to reach a concept c from the start concept C_s is represented as $d(c)$. The algorithm also defines cost functions "UpCost" and "DownCost," representing the costs of moving upwards and downwards in the lattice. The Dijkstra-based distance measure uses a priority queue Q, based on a min-heap, where each element is a pair $(d(c), c)$ sorted by $d(c)$. It also maintains a set V to keep track of the nodes already visited. The cost function $f: C \times C \rightarrow \mathbb{R} \cup \{\infty\}$ is defined with c and c' being two formal concepts in the lattice:

$$f(c, c') = \begin{cases} UpCost, & if\ c \supseteq c' \\ DownCost, otherwise \end{cases} \quad (7)$$

Subsequently, the Dijkstra-based distance measure, represented as $d: C \times C \rightarrow \mathbb{R} \cup \{\infty\}$, computes the minimum cost path from the start concept C_s to the end concept C_e in the lattice:

$$d(C_s, C_e) = min\{\sum_{i=1}^{n-1} f(c_i, c_{i+1}) \mid (c_1, c_2, \dots, c_n) is\ a\ path\ from\ C_s\ to\ C_e\} \quad (8)$$

Here, n is the number of formal concepts in a specific path from C_s to C_e. For each possible path from C_s to C_e, sum the costs from each step c_i to c_{i+1} in the path, and $d(C_s, C_e))$ is the minimum of these sums over all possible paths. The algorithm functions as follows:

Algorithm 1: The Dijkstra-based distance measure algorithm on the concept lattice.

Inputs: C_s, C_e, $\mathcal{H}(C, E)$, UpCost, DownCost.
Output: minimum cost from C_s to C_e
Initialize:
    For each c in $\mathcal{H}$:
        $d(c) \leftarrow \infty$
    EndFor
    $d(C_s) \leftarrow 0$
    Initialize P and $V \leftarrow \emptyset$
    Insert $(0, C_s)$ into Q
While $Q \neq \emptyset$ *do*:
    $(d(c), c) \leftarrow Dequeue(Q)$
    If $c = Ce$ *then*:
        Return $d(C_e)$.
    EndIf
    If c not in V then:
        Add c to V
    EndIf
    For each neighbor u of c do:
        If neighbor not in V:
            If c is a superset of u then:
                $cost \leftarrow d(c) + UpCost$
            Else:
                $cost \leftarrow d(c) + DownCost$
            EndIf
            If $cost < d(u)$ *then*:
                $d(u) \leftarrow cost$
                $P(u) \leftarrow c$
                Enqueue $(d(u), u)$ *into* Q
            EndIf
        EndIf
    EndFor
EndWhile

In Algorithm 1, several symbols are introduced for clarity. The symbol C_s denotes the starting concept in the concept lattice, while C_e represents the ending concept. $\mathcal{H}(C, E)$refers to the concept lattice itself, composed of concepts C and edges E. The terms 'UpCost' and 'DownCost' specify the costs for upward and downward movements within the lattice, respectively. $d(c)$ is used to signify the current shortest path distance from C to any given concept c A predecessor map is denoted by P, where $P(c)$ reveals the predecessor of a concept c in the shortest path originating from C_s. Finally, Q and V serve as a priority queue for upcoming nodes to visit and a set for nodes already visited, respectively. The algorithm always returns the minimum cost of the shortest path between C_s and C_e due to the property of the lattice structure, which ensures a path exists between any two concepts.

This approach has a time complexity of $O\ (E\ +\ Clog(C))$, where E is the number of edges (relationships between formal concepts), and C is the total count of formal concepts in the lattice. By leveraging the lattice structure, the cost function, and an efficient min-heap-based priority queue, the Dijkstra-Based Distance Measure provides a more accurate representation of dissimilarities in categorical data. This results in an optimized clustering process and yields more accurate and meaningful cluster assignments.

2) Cluster centers: Defining the cluster centers, or centroids, in a concept lattice is vital for effectively implementing the K-means Dijkstra on Lattice (KDL) method. These centroids need to be formal concepts within the lattice. The continual updating and calculation of these representative centroids significantly influence the minimization of the overall clustering cost function. To formally describe this, consider a cluster S composed of a set of formal concepts $\{c_i\}$ where $i = 1,2,\dots,|S|$. The representative formal concept, denoted as Z, is defined as the concept within S that minimizes the sum of the distances to all other concepts in the same cluster. This can be mathematically expressed as:

$$Z = argmin_{Z \in S}\left(\sum_{i=1}^{|S|} d(c_i, Z)\right) \quad (9)$$

Here,

c_i represents each concept within the cluster S

In Eq. (9), $d(c_i, Z)$ is the Dijkstra-based distance from the potential centroid Z to each concept c_i within the cluster. The argmin operation is employed to find the formal concept Z in S that yields the smallest sum of distances to all other concepts in the cluster S. It is important to note that Z inherently belongs to the cluster S, which allows for a more efficient calculation of the minimal sum of distances to all other concepts within S. Furthermore, the existence of a center for any set of formal concepts is ensured due to the properties of the Dijkstra-based distance measure. This consistency makes the method universally applicable, regardless of the specific set of formal concepts under consideration. This method of defining cluster centers in the concept lattice adheres to mathematical rigor while being practically feasible, offering a systematic way to

manage and interpret complex categorical datasets. This method enhances the interpretability of clustering results by identifying representative formal concepts for each cluster, fostering more comprehensive and insightful data analysis.

3) The clustering algorithm: The K-Means Dijkstra on Lattice (KDL) clustering algorithm, grounded in Formal Concept Analysis (FCA) and the Dijkstra-based distance measure, can be articulated through the following sequential stages:

Algorithm 2: K-Means Dijkstra on Lattice (KDL) clustering algorithm

Inputs: k, the number of clusters; $\mathcal{B}$, the lattice of formal concepts.
Output: The resulting clusters $\{S_1, S_2, \ldots, S_k\}$.
Initialize:
Select k formal concepts $\{c_1, c_2, \ldots, c_k\}$ from the lattice $\mathcal{B}$ randomly as the initial centroids of the k clusters.
Assignment:
For each formal concept $c \in \mathcal{B}$ *do*:
Assign c to the cluster S_i for which the Dijkstra-based distance measure $d(c, Z_i)$ is minimized, where Z_i is the centroid of cluster S_i.
Using Equations (7, 8)
Centroid Update:
For each cluster S_i *do*:
Recalculate the centroid Z_i as the formal concept c that minimizes the total distance to all other concepts within S_i
Using Equation (9)
Iteration:
While centroids change between iterations *do*:
Repeat steps 2 and 3.
Finalization:
Output the resulting clusters $\{S_1, S_2, \ldots, S_k\}$.

4) Cost analysis of k-means dijkstra on lattice (KDL) method: This section analyzes the computational complexity of the proposed K-Means Dijkstra on Lattice (KDL) method. The computational cost of each step will be evaluated, from the initialization of clusters to their final assignment. This analysis will provide insights into the efficiency and scalability of the KDL method.
Given the parameters:

- K represents the number of clusters.
- N represents the number of objects.
- A represents the number of attributes.
- C represents the number of concepts.
- E represents the number of edges in the lattice.
- B represents the maximum number of border elements in the lattice construction.

The proposed K-Means Dijkstra on Lattice (KDL) method commences with data preprocessing, wherein the categorical dataset transforms a formal context. This stage involves a binary translation of each dataset entry, leading to a time complexity of $O(NA)$. Subsequently, a lattice is generated from the formal concepts obtained in the previous stage. This phase necessitates looping over all border elements for each concept, inducing a worst-case time complexity of $O(CB)$. The final stage of the process encompasses a K-means-like clustering operation. This phase includes iterations over all the concepts to assign clusters and update centroids. The computation of the shortest paths between pairs of concepts within the lattice primarily influences this stage's time complexity. This is achieved using Dijkstra's distance algorithm. Assuming I iterations are required to reach convergence, the time complexity for computing the shortest paths between all pairs of concepts culminates in $O(IKC(E + C\ log\ C))$. To summarize, the KDL method's overall time complexity, significantly influenced by the data preprocessing, lattice construction, and lattice-based clustering stages, can be approximated as $O(NA + CB + IKC(E + C\ log\ C))$. It is worth noting that this is a rough estimation, with actual time complexity potentially varying based on the characteristics and data distribution within the formal context. However, focusing on the dominant term for the sake of simplification, the time complexity of the KDL method becomes $O(IKC(E + C\ log\ C))$.

B. K-means Vector on Lattice (KVL)

The K-means Vector on Lattice (KVL) method is essential for converting categorical data into numerical data. Leveraging the classical k-means algorithm facilitates data grouping, making it instrumental for various data analysis operations, especially when dealing with predominantly categorical or non-numeric data. The essence of this method lies in its capacity to convert formal concepts, regardless of their abstract or categorical nature, into 'concept description vectors'. These vectors exist in a real-valued vector space, which not only makes them easily adaptable to standard mathematical procedures but also optimizes them for computational analysis. Each vector represents the formal concept from which it was derived, encapsulating its fundamental attributes. Every dimension within the vector signifies a different attribute of the concept, with its magnitude corresponding to the prevalence or significance of the attribute within the concept. This forms a compressed yet efficient way to contain the information intrinsic to the formal concept.

With the creation of concept description vectors, these entities can now be subjected to the k-means algorithm. This renowned clustering method partitions the data into a specified number 'k' of distinct clusters. Each cluster is identified by its centroid, which serves as the symbolic or physical center of the cluster. All data points, or concept description vectors in this context, within a specific cluster have a closer similarity to their cluster's centroid than to those of other clusters. This aids in the aggregation of analogous concepts, thereby facilitating insightful analysis of the data.

Definition 5. Concept Description Vector: Let $Y = (A, B)$ be a formal concept, where $A \subseteq G$, $B \subseteq M$, and a context $T = (G, M, I)$ has $|M| = q$, $|G| = r$, the incidence relation $I \subseteq G \times M$ can be represented as a binary matrix, where the rows correspond to the elements of G (objects), the columns correspond to the elements of M (attributes), and each entry of the matrix is either 1 or 0, indicating whether the relation (g, m) exists or not. Let's denote this matrix as I, with

dimensions $r \times q$, where r is the number of objects in G and q is the number of attributes in M. The rows of the matrix can be labeled by g_1, g_2,..., g_r and the columns by m_1, m_2,..., m_q, the matrix can be defined as shown in Table II:

The concept description vector is defined as $V_Y = (v_{m_1}, v_{m_2}, \ldots, v_{m_q})$. v_{m_h} (h ranging from 1 to q) is obtained as follows [5]:

$$v_{m_h} = \begin{cases} 1 & if\ m_h \in B \\ \frac{1}{r}\sum_{j=1}^{r} I(g_j, m_h) & if\ m_h \notin B, \forall\ g_j \in G \end{cases} \quad (10)$$

where $m_h \in M$.

The method to calculate each attribute within a concept is different. It depends on whether the attribute is in the intent of each concept or not. The concept vector is the base for getting the similarity between concepts. This vector is obtained from the context based on the intent of each concept. Depending on whether the attribute m_h is a part of the intent B. If m_h is a part of the intent, it is assigned a value of 1, indicating its high importance in defining the concept. If not, it's calculated as the average association of this attribute across all objects, represented as $\frac{1}{r}\sum_{j=1}^{r} I(g_j, m_h)$ $if\ m_h \notin B$. This can be perceived as the frequency or relevance of this attribute across the object set G. The computation of each attribute forms the concept description vector used to ascertain the similarity between concepts.

After defining the concept description vectors, the KVL method introduces the concept similarity measure. This measure, often referred to as Concept Similarity (CS), as explicated in Definition 6, is used to ascertain the proximity of these concepts. Concept Similarity is calculated using the Euclidean Distance between any two concept description vectors V_{Y1} and V_{Y2}. The CS equation helps us to quantify the closeness between two concepts, taking into account each component of their respective concept description vectors.

Definition 6. Concept Similarity: Concept Similarity (CS) is calculated based on the concept description vector in Definition 5 using traditional Euclidean distance. For any two concept description vectors

$V_{Y_1} = (V_{Y_1 m_1}, V_{Y_1 m_2}, \ldots, V_{Y_1 m_q})$ and $V_{Y_2} = (V_{Y_2 m_1}, V_{Y_2 m_2}, \ldots, V_{Y_2 m_q})$, the Euclidean distance is defined as per Eq. (5):

$$CS(V_{Y_1}, V_{Y_2}) = \sqrt{(V_{Y_1 m_1} - V_{Y_2 m_1})^2 + (V_{Y_1 m_2} - V_{Y_2 m_2})^2 + \ldots + (V_{Y_1 m_q} - V_{Y_2 m_q})^2} \quad (11)$$

TABLE II. MATRIX CORRESPONDING TO THE RELATION I

Objects/Attributes	$1m_1$	m_2	...	m_q
g_1	$I(g_1, m_1)$	$I(g_1, m_2)$	...	$I(g_1, m_q)$
g_2	$I(g_2, m_1)$	$I(g_2, m_2)$	...	$I(g_2, m_q)$
...	...	...	...	...
g_r	$I(g_r, m_1)$	$I(g_r, m_2)$	...	$I(g_r, m_q)$

This framework of concept description vectors and concept similarity lays the groundwork for the k-means clustering algorithm. The algorithm takes the concept description vectors as inputs and leverages the concept similarity measure to identify which concepts most resemble each other. Concepts exhibiting high similarity are then grouped into clusters. The center of each cluster, represented by Z_i, is calculated as the mean of all concept description vectors within that cluster. Let's denote S_i as the i^{th} cluster ($i = 1, 2, \ldots, k$), where k is the number of clusters. The centroid of each cluster S_i can be defined as:

$$Z_i = \frac{1}{|S_i|}\sum_{j=1}^{|S_i|} V_{Y_j}, V_{Y_j} \in S_i \quad (12)$$

The k-means algorithm aims to minimize the within-cluster sum of squares (WCSS) of Euclidean distances. This objective function Q is as follows:

$$Q = \sum_{i=1}^{k}\sum_{j=1}^{|S_i|} ||V_{Y_j} - Z_i||^2], V_{Y_j}, Z_i \in S_i \quad (13)$$

The algorithm alternates between assigning each concept description vector to the nearest centroid and recalculating the centroid of each cluster using Eq. (11) and (12), until the clusters stabilize. The algorithm has converged, and the clusters are optimally partitioned concerning the given concept description vectors.

1) The clustering algorithm: The main idea is as follows. Suppose $T = (G, M, I)$ is a formal context and $V(T)$ is the set of all concept description vectors and the number of clusters is K. Firstly, the initial centers, $Z_t^0 = (A_t, B_t)$ $(t = 1,2,\ldots K)$, of K clusters are selected randomly, and the corresponding clusters are $S_t^0 = \{Z_t^0\}$. Secondly, join a concept description vector $v \in V(T)$ into one cluster according to the following rule: if the distance between v and Z_t^0 is lower than that of v and other centers, then, v is put into S_t^0. Each vector in $V(T)$ can be adjusted according to this rule. Thirdly, the new center of each cluster can be determined by calculating the mean value of vectors within each cluster and set it as a new center. Finally, repeat the above process till the twice computation of each cluster and center are the same. The algorithm steps are as follows:

Algorithm 3. K-means clustering of concepts.

Input: All the description vectors of concepts in $V(T)$, k.
Output: The clusters and corresponding centers.
Initialize:
Set $S_1^i \leftarrow \emptyset$, $S_2^i \leftarrow \emptyset$, ..., $S_k^i \leftarrow \emptyset$;
$i \leftarrow 0$,
Select initial center vectors of K clusters: Z_1^i, Z_2^i,..., Z_k^i;
Assignment:
For each $v \in V(T)$ *do*:
-Find t such that
$CS(distance)(v, Z_t^i) \leq CS(distance)(v, Z_j^i)$, $(j = 1,2, \ldots, k)$ then,
$v \in S_t^i$;
EndFor
Centroid Update:
For each S_t^i *do*:
$Z_t^{i+1} = \frac{1}{|S_t|}\sum_{s=1}^{|S_t|} v_s, v_s \in S_t$, using Equation (12)

$S_t^{i+1}=\{v \in V(T) | CS(v, Z_t^{i+1}) \leq CS(v, Z_j^{i+1})\}$
using Equation (11)

EndFor

Convergence Check:

If $Z_t^i=Z_t^{i+1}$, $S_t^i=S_t^{i+1}$, $t = 1,2,\ldots,K$, *then*
Go to step5.
Else:
$i = i + 1$,
Go to Step 2.

Output: clusters S_1^i, S_2^i, ..., S_k^i and the corresponding centers Z_1^i, Z_2^i, …, Z_k^i.

Algorithm 1 outlines the process for performing K-means clustering on the set of concept description vectors. The input to the algorithm is the set of all concept description vectors, and the output is the resulting clusters and their centers. The algorithm begins by randomly initializing the clusters and selecting the initial centers for the clusters. It then assigns each concept description vector to the cluster whose center it is closest to, based on the concept similarity measure in Eq. (11). It then updates the clusters' centers based on the mean of the concept description vectors within each cluster using Eq. (12) and repeats this process until the clusters stabilize. mapping the vectors back to the original concepts. Once the K-means clustering is done and the clusters are stabilized, the vectors within each cluster can be traced back to their original concepts. The mapping uses the concept description vectors created in the first step.

Algorithm 4: Mapping vectors to original concepts.

Input: The clusters S_1^i, S_2^i, …, S_k^i and the corresponding centers Z_1^i, Z_2^i,…, Z_k^i.
Output: Clusters of original concepts.
Initialize:
$NS_1, NS_2, \ldots NS_k \leftarrow \emptyset$,
For $t = 1$ *to* k *do*:
For each description vector in S_t^i *do:*
Map back to its original concept and add to NS_t.
Using the relationship between the concept description vector and the original concept established in Definition 5.
EndFor

Output: the new clusters $NS_1, NS_2, \ldots NS_k$, each containing the original concepts.

In this way, the numerical data obtained from the K-means clustering is transformed back into categorical data, giving us clusters of similar concepts rather than clusters of similar vectors. This method of approximation and mapping allows for efficient and meaningful clustering of concepts in a context, making it easier to understand and interpret the relationships and similarities between different concepts.

2) Cost Analysis of the KVL Method: This section thoroughly reviews the computational complexity of the KVL method to assess its efficiency and scalability. The first stage is data preprocessing, which involves the transformation of a dataset into a formal context. Here, a binary representation of each object with N objects and A attributes is needed, introducing a time complexity of O(NA). After preprocessing, formal concepts are generated and then converted into vectors in an A-dimensional attribute space. For each of the C concepts, an equivalent vector in the attribute space needs to be calculated, leading to a time complexity of O(AC). The algorithm starts by randomly selecting K centroids from the pool of C concepts, resulting in a time complexity of O(K). Subsequent stages include iterative assignment and update, where each concept is assigned to its nearest centroid, and centroids are updated based on new assignments. This incurs a time complexity of O(CK) per iteration. These processes are performed I times until convergence, resulting in a total time complexity of this stage being O(ICK).

Lastly, the mapped vectors are reconverted to their original formal concepts, which involves a time complexity of O(CK). Summing up the complexities from each phase, the total time complexity of the KVL method is estimated to be O (NA+ AC+K+ICK+CK). This is a heuristic estimate, and time complexity could vary based on data distribution and other runtime factors. However, focusing on the dominant term for simplification, the time complexity of the KVL method becomes O(IKC).

VI. Experimental Results

In this section, the presented experimental results aimed at demonstrating the performance of the Dijkstra-Based Distance Measure, as well as the performance and scalability of the K-means Dijkstra on Lattice (KDL) and K-means Vector on Lattice (KVL) methods. The experiments were conducted on a Mac system equipped with an Apple M1 chip and 8GB of RAM, running Mac OS 13.2.1.

A. Testing and Evaluation of Dijkstra-based Distance Measure

In the experimental section, the performance of the distance measure based on Dijkstra's algorithm is rigorously evaluated. The testing process involved the following steps:

1) Random generation of formal contexts: five formal contexts are randomly generated with varying sizes and densities; the characteristics of these formal contexts are described in Table III. The density parameter in this context refers to the proportion of filled entries (1s) compared to the total number of possible entries in the binary matrix representation of the formal context. It quantifies how much information is present regarding the relationship between objects and attributes. To explain the density parameter, let's consider an example from Table III: Formal Context1 with 600 objects and 125 attributes. The density for Formal Context1 is 0.10, indicating that, on average, each entry in the binary matrix has a 0.10 probability of being filled (assigned a value of 1). A lower density value implies sparser relationships, where fewer objects belong to the given attributes or categories. In contrast, a higher density value indicates denser relationships, where a larger number of objects are associated with the given attributes.

2) In the analysis, four datasets from the UCI Machine Learning repository are meticulously examined. Prior to conducting any experiments, these datasets are transformed into formal contexts, with details outlined in Table III. The

datasets were chosen based on two key criteria: public accessibility and the categorical characteristics of their attributes. The selected datasets include:

- The Balance-Scale dataset is designed to replicate psychological experiment results. Each instance in this dataset can be labeled based on whether the balance scale leans to the left, right, or is balanced.
- The Breast Cancer dataset obtained from the University of Wisconsin hospitals, classifies each instance into one of two potential categories: benign or malignant.
- The Car Evaluation dataset, which results from a simple hierarchical decision model initially designed for DEX's demonstration, classifies each instance into one of four classes: unacc, acc, good, and vgood.
- Tae dataset representing teaching performance assessment over five semesters (three regular and two summers) includes 151 teaching assistant assignments from the University of Wisconsin-Madison's Statistics Department. All instances fall into one of three categories: low, medium, and high.

3) Extraction of formal concepts: The NextClosure algorithm was used to extract the set of formal concepts from each formal context. The number of formal concepts generated from each formal context is shown in Table IV.

4) Hasse diagram construction: Utilizing the Ipred algorithm, a Hasse diagram was structured optimally for the case study, as elaborated in Section V. The characteristics of the generated Hasse diagrams are shown in Table V, by considering the density parameter in the construction of the Hasse diagram. The density parameter influences the number of edges and nodes in the Hasse diagram. A denser formal context with a higher density value tends to result in a larger number of formal concepts and, consequently, a more extensive concept lattice with a higher number of edges connecting the concepts. On the other hand, a sparser formal context with a lower density value leads to a smaller concept lattice with fewer edges.

TABLE III. CHARACTERISTICS OF RANDOM AND REAL-WORLD FORMAL CONTEXTS

Formal Contexts	#objects	#attributes	density
Formal Context1	600	125	0.10
Formal Contex2	11000	30	0.10
Formal Context3	1350	120	0.05
Formal Context4	2000	20	0.15
Formal Context5	12000	20	0.23
Balance-Scale	625	20	0.20
Breast Cancer	182	35	0.25
Tae	151	101	0.04
Car Evaluation	1728	21	0.28

TABLE IV. FORMAL CONCEPTS GENERATED FROM THE FORMAL CONTEXTS IN TABLE III

Formal Contexts	#formal concepts.
Formal Context1	29926
Formal Contex2	15117
Formal Context3	9882
Formal Context4	2989
Formal Context5	39931
Balance-Scale	1297
Breast Cancer	2569
Tae	276
Car Evaluation	8001

TABLE V. HASSE DIAGRAM TRAITS VIA IPRED ALGORITHM

Formal Contexts	#formal concepts	Inclusion relationship between concepts (edges)
Concept lattice1	29926	122839
Concept lattice2	15117	67040
Concept lattice3	9882	36797
Concept lattice4	2989	12175
Concept lattice5	39931	228427
Balance-Scale	1297	4945
Breast Cancer	2569	9513
Tae	276	619
Car Evaluation	8001	38928

The analysis involved running a Dijkstra-based distance measure on concept lattices generated from five random formal contexts and four real-world datasets. The formal contexts varied in number of objects, attributes, and density. On the other hand, the real-world datasets were diverse, encompassing balance scale, breast cancer, teaching assistant evaluation, and car evaluation data. After generating the formal contexts and preparing the datasets, Formal Concept Analysis (FCA) using the NextClosure algorithm has been performed to derive formal concepts. The count of these formal concepts varied significantly across the contexts and datasets, ranging from as low as 2989 in Formal Context 4 to as high as 39931 in Formal Context 5. Using these formal concepts, Hasse diagrams (concept lattices) constructed with the help of the Ipred algorithm. The concept lattices illustrated the inclusion relationship between concepts. Again, the number of inclusion relationships was directly related to the complexity and size of the corresponding formal context. Subsequently, the distance measure was evaluated. Concept pairs were randomly chosen from each concept lattice, constituting 25% of the total concepts. The minimum cost of the shortest path between these pairs was calculated using the designated distance measure. This evaluation was performed across ten trials, with both the average runtime and mean distance documented for each.

We're observing a comparison of the average run time of the Dijkstra-based distance measure algorithm and the mean distance between concepts for both randomly generated formal datasets and real-world datasets. As indicated in Fig. 2 and

Fig. 3 for randomly generated datasets, there is a clear correlation between the number of concepts in the lattice and the algorithm's runtime. An increase in the number of concepts leads to a corresponding rise in runtime. This finding aligns with expectations, as a lattice with a greater number of concepts and relationships is likely to be more complex. Consequently, calculating the shortest path between pairs in this intricate structure would naturally demand more computational time and resources. The mean distances were mostly consistent for each context, indicating a relatively stable distance measure despite potential variance in the random generation of the formal contexts. This reinforces the efficacy of the Dijkstra-based distance measure, highlighting its stability across multiple trials of randomly generated data.

Patterns similar to those observed in randomly generated datasets were also evident in real-world datasets, as demonstrated in Fig. 4 and Fig. 5. The runtimes correlate with the number of concepts in the lattice, with larger lattices taking longer to calculate the shortest paths. Interestingly, the Car Evaluation dataset, which had the highest number of concepts (8001), exhibited the shortest mean distance (6.5735) among the real-world datasets. This suggests that although the dataset is complex, the relationships within the data are more straightforward or closer than the other datasets. Meanwhile, the Balance-Scale and Breast Cancer datasets had a more moderate number of concepts (1297 and 2569, respectively) and showed a higher mean distance. This might indicate that, despite having fewer concepts, the relationships in these datasets could be more complex or convoluted.

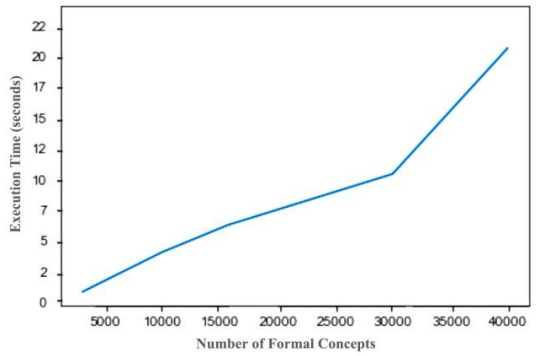

Fig. 2. Average runtime of distance calculation algorithm on different concept lattice sizes for the random contexts in Table IV.

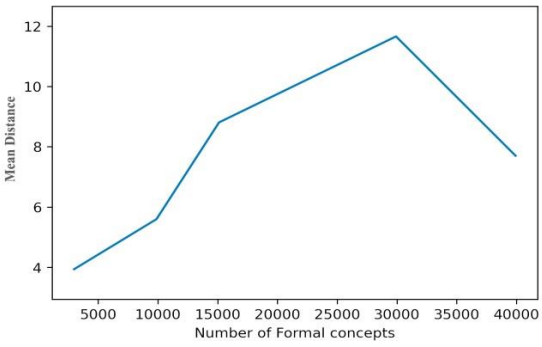

Fig. 3. Mean distance of distance measure algorithm on different concept lattice sizes for the random contexts in Table IV.

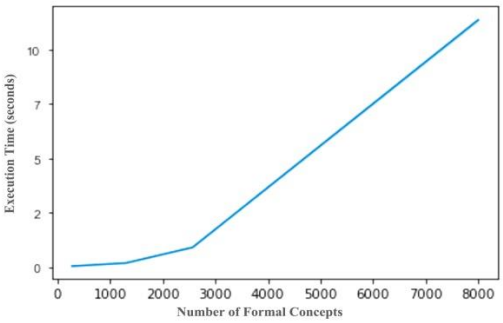

Fig. 4. Average runtime of distance calculation algorithm on different concept lattice sizes of real-world datasets.

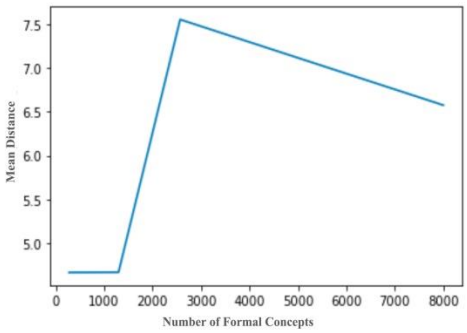

Fig. 5. Mean distance of distance measure algorithm on different concept lattice sizes of real-world datasets.

Overall, the results suggest that the Dijkstra-based distance measure is robust and stable across various randomly generated and real-world contexts. The run time increases as expected with the size and complexity of the dataset, and the measure captures the inherent complexity in the data (as reflected in the mean distances) and provides valuable insights into the structural properties of concept lattices. It allows for identifying concept pairs relatively closer or farther apart within the lattice structure. The results contribute to a better understanding of relationships and structural characteristics within formal contexts and concept lattices. The consistent performance of the measure across different scenarios reinforces its potential utility in handling diverse and complex categorical datasets.

Adapting the FCA, the Dijkstra-based distance measure applies the robust Dijkstra's algorithm to compute the shortest path between two categorical data points. This method, mindful of the hierarchical structure of categorical data, quantifies dissimilarity by evaluating paths within the data space. It provides a viable alternative to Euclidean distance within the clustering context when enhancing the K-means algorithm for categorical data analysis. Replacing Euclidean distance with the Dijkstra-based measure allows the K-means algorithm to cluster categorical datasets better, accurately reflecting the relationships and similarities between categorical variables. Incorporating the Dijkstra-based distance measure in the K-means algorithm aids cluster identification based on categorical patterns, providing meaningful insights and potential applications across numerous domains. It's ushers in new avenues for examining and interpreting categorical data.

B. Clustering Performance

In the present section, the performance of two distinct clustering approaches designed for categorical data: K-means Dijkstra on Lattice (KDL) and K-means Vector on Lattice (KVL) is scrutinized. We've chosen the Silhouette Coefficient and the Davies-Bouldin Index (DBI) as the evaluation metrics due to their ability to assess clustering performance without the need for ground truth labels, making them especially useful in real-world applications where such labels might not be available. The Silhouette Coefficient serves as a measure to ascertain the suitability of a data point's allocation to its cluster in comparison to other clusters. The coefficient fluctuates between -1 and 1, with a high positive value implying a well-clustered data point, while a negative one indicates potential misplacement within a cluster. Here's the mathematical expression for the Silhouette Coefficient:

$$\text{Silhouette Score} = (b - a) \,/\, max\,(a, b) \quad (14)$$

Where $'a'$ is the mean intra-cluster distance and $'b'$ is the mean nearest-cluster distance. In contrast, the Davies-Bouldin Index (DBI) is a metric that evaluates the separation and compactness of clusters. A lower DBI value implies an optimal clustering solution. The DBI is calculated as follows:

1) Calculate the average distance between each point in a cluster S_i and all other points in the same cluster. This is often referred to as the intra-cluster distance. Denote this as SC_i for cluster S_i.

$$SC_i = (1 / n_i)\, \Sigma\, ||x - Z_i||\; for\; x \in S_i$$

where:

- n_i is the number of points in cluster S_i,
- x is a point in cluster S_i,
- Z_i is the centroid of cluster S_i,
- $||x - Z_i||$ is the distance between point x and centroid Z_i.

2) Calculate the distance d_{ij} between cluster S_i and S_j, using a suitable distance measure between the centroids of the clusters.

3) Calculate the ratio R_{ij} between the sum of the intra-cluster distances of cluster S_i and S_j, and the inter-cluster distance between S_i and S_j.

$$R_{ij} = (SC_i + SC_j) \,/\, d_{ij}$$

4) For each cluster S_i, find the maximum ratio R_i which is the maximum R_{ij} for all $j \neq i$.

$$R_i = max(R_{ij}) \text{ for all } j \neq i$$

5) The Davies-Bouldin Index (DBI) is the average of all R_i.

$$\Delta BI = (1 / S) \;\; \sum R_i \quad (15)$$

where, S is the total number of clusters

Again, lower DBI values indicate better clustering because this signifies clusters that are more compact (lower intra-cluster distances SC_i) and better separated (higher inter-cluster distances d_{ij}).

The analysis is based on four real-world datasets described in the previous section as shown in Table V and the results in Tables VI and VII. The clustering is performed by setting the number of clusters (k value) equal to the number of classes for each dataset to maintain consistency with the inherent data structure. These results are recorded from the averages of 100 runs for each method A closer examination of these tables allows for a comparative analysis of the performance of the K-means Dijkstra on Lattice (KDL) and K-means Vector on Lattice (KVL) methods. Regarding the Silhouette Coefficient (Table VI), it is evident that the KDL method, which utilizes the inherent lattice graph structure of categorical data for clustering, consistently outperforms the KVL method, regardless of the dataset used. This outcome is further corroborated by the DBI results (Table VII), where the KDL method again demonstrates superior performance by consistently achieving lower index values across all datasets. This can be attributed to the design of the KDL method. The KDL strategy focuses on integrating the representation of categorical data based on the graph structure, effectively leveraging the potential similarity between these data points. It employs Formal Concept Analysis (FCA), a mathematical framework for generating a concept hierarchy, and Dijkstra's algorithm to calculate the shortest path between formal concepts in the FCA graph. This novel distance measure, which represents the minimal cost of moving from one formal concept to another, facilitates a more accurate clustering process.

On the contrary, the KVL method, while simplifying the clustering process by converting categorical data into numerical vectors and using standard k-means algorithms, risks obscuring the inherent hierarchical relationships between categorical values. This transformation can potentially result in less effective clustering performance.

TABLE VI. SILHOUETTE COEFFICIENT SCORES OF CLUSTERING PERFORMANCE FOR K-MEANS DIJKSTRA ON LATTICE (KDL) AND K-MEANS VECTOR ON LATTICE (KVL) METHODS ACROSS DIVERSE DATASETS

Datasets	KDL	KVL	#Clusters
Balance-Scale	**0.406**	0.128 0.090 0.092 0.106	3
Breast Cancer	**0.239**		2
Tae	**0.300**		3
Car Evaluation	**0.563**		4

TABLE VII. DBI SCORES OF CLUSTERING PERFORMANCE FOR K-MEANS DIJKSTRA ON LATTICE (KDL) AND K-MEANS VECTOR ON LATTICE (KVL) METHODS ACROSS DIVERSE DATASETS

Datasets	KDL	KVL	# Clusters
Balance-Scale	**1.48**	2,64 2,78 2.62 2.92	3
Breast Cancer	**1.83**		2
Tae	**1.49**		3
Car Evaluation	**1.90**		4

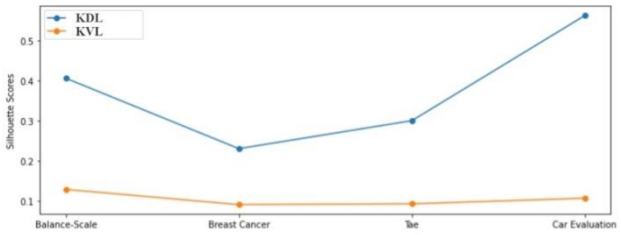

Fig. 6. Silhouette scores by dataset and method.

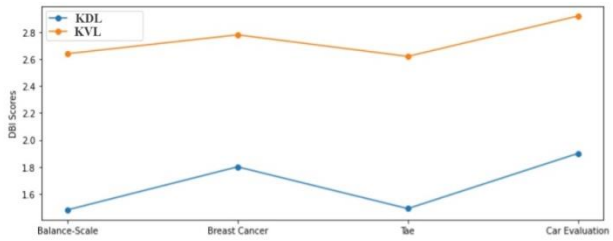

Fig. 7. DBI scores by dataset and method.

The compelling evidence in Tables VI and VII and their corresponding graphical representations in Fig. 6 and Fig. 7 illuminate the KDL method's superior performance over the KVL method for clustering categorical data. By directly handling categorical data and leveraging its inherent hierarchical structure, the KDL method offers more meaningful and accurate clustering results. This finding corroborates the hypothesis that leveraging the inherent similarities and structure of categorical data can yield improved results. While the KVL method simplifies the process by transforming categorical data into numerical vectors, it potentially obscures the intricate hierarchical relationships between categorical values, thus diminishing the method's effectiveness. This is reflected in the lower Silhouette, and higher DBI scores observed for the KVL method. These findings not only present solid evidence in favor of the KDL method as a more potent tool for clustering categorical data, but they also underscore the importance of utilizing the data's inherent structure where possible. However, these conclusions should uphold the utility of the KVL method. Instead, they serve as a critical reminder of the significance of selecting an appropriate tool for the data at hand, considering each method's potential trade-offs and benefits.

C. Scalability Test Results Analysis

1) Scalability in relation to the number of clusters: To bolster the robustness of the study concerning the K-means Dijkstra on Lattice (KDL) and K-means Vector on Lattice (KVL) clustering techniques, a meticulous analytical approach was employed. This rigorous methodology underscores the credibility of the performance assessments and findings presented. All results presented in this analysis were derived from the average runtime of five independent runs. This method was utilized to mitigate any outliers' influence and deliver a more precise portrayal of each method's performance.

The investigation was particularly interested in the scalability of these methods in response to an increase in the number of clusters. The number of clusters was varied from 2 to 18 in the analysis, with the dataset size held constant. This aspect is essential in real-world situations, especially when data is complex and needs to segregate into a limited number of clusters neatly. The performance of both methods in relation to the varying number of clusters was assessed using the 'Car Evaluation' dataset consisting of 8001 formal concepts. From Fig. 8, it is evident that the K-means Vector on Lattice (KVL) method demonstrates scalability. A linear relationship is observed between execution time and the increment in the number of clusters. The execution time varies approximately between 44.48 and 51.56 seconds as the number of clusters changes from 2 to 18. This pattern underscores the efficiency of the KVL method in handling larger and more complex datasets.

This method, thus, shows promise in effectively managing a rise in the number of clusters without causing a substantial increase in execution time. On the other hand, Fig. 9 provides insights into the scalability of the KDL method. This method shows rapid growth in execution time as the number of clusters increases. The time jumps from about 1926.77 seconds for 2 clusters to a massive 49600.10 seconds for 18 clusters. Given the complexity of the lattice graph and the number of formal concepts, the KDL method's computational load increases significantly with the number of clusters, suggesting lower scalability.

The KVL method is better in terms of scalability and efficiency for an increasing number of clusters; the KDL method provides higher-quality clustering, though it demands significantly more computational time and resources. This highlights the importance of finding the right balance between computational efficiency and clustering quality. The preference for one over the other may vary depending on the specific situation and constraints.

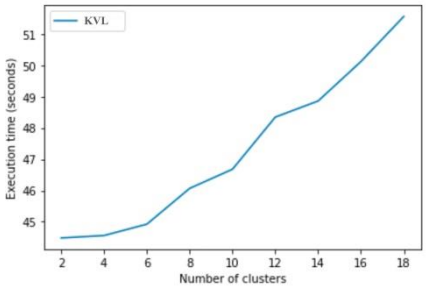

Fig. 8. Scalability of KVL Method to the number of clusters when clustering 8001 formal concepts of the 'car evaluation' dataset.

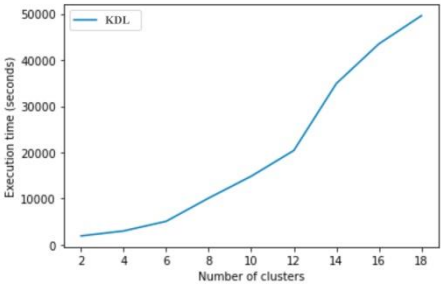

Fig. 9. Scalability of the KDL method to the number of clusters when clustering 8001 formal concepts of the 'car evaluation' dataset.

2) Scalability in relation to the number of formal concepts: In examining the scalability of KDL and KVL, performance was assessed with an increasing number of formal concepts, while keeping the cluster count constant at three. This analysis is grounded in multiple iterations of these methods on a selection of real-world datasets, namely Balance-Scale, Breast Cancer, Tae, and Car Evaluation, described in detail in Tables III, IV, and 5. Fig. 10, and Fig. 11, represent the scalability of the KVL and KDL methods, respectively, demonstrating how they fare with a rising count of formal concepts.

Diving into Fig. 10, it's evident that the KVL method shows admirable consistency. The recorded execution times from five separate runs, 43.14, 43.25, 44.15, and 46.35 seconds, corresponding to the datasets featuring 276, 1297, 2569, and 8001 formal concepts. This suggests that as the number of formal concepts increases, the KVL method retains its efficiency, reflecting robust scalability - an attribute crucial for managing large datasets. In comparison, Fig. 11 encapsulates the performance of the KDL method. The execution times here are noticeably higher, registering at 53.67, 301.47, 830.02, and 2026.79 seconds for the same gradual increase in formal concepts. It's clear that as the number of formal concepts expands, the KDL method's execution time climbs drastically, indicating an intensifying computational requirement and limited scalability when tasked with larger datasets.

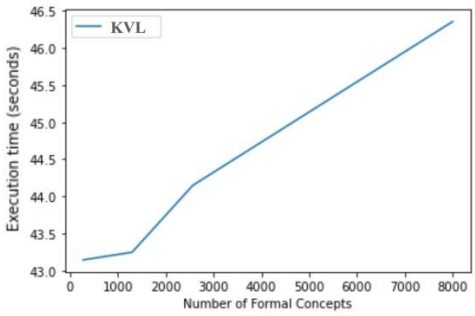

Fig. 10. Scalability of KVL method with increasing number of formal concepts.

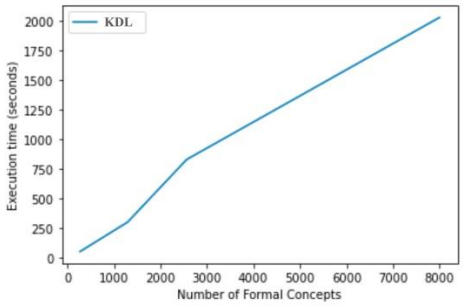

Fig. 11. Scalability of KDL method with increasing number of formal concepts.

This investigation rigorously assesses the K-means Dijkstra on Lattice (KDL) and K-means Vector on Lattice (KVL) algorithms, identifying a trade-off between clustering quality and computational efficiency. KDL excels in quality but is resource-intensive, making it less scalable. Conversely, KVL is more scalable but may compromise on quality. The choice between the two hinges on task-specific needs: KDL is better for quality-focused tasks with sufficient resources, while KVL is ideal for tasks requiring scalability. Future research could aim to optimize each method's shortcomings, offering a more balanced clustering solution. These refinements would bring us closer to a unified, efficient, and high-quality clustering algorithm for handling categorical data.

VII. Conclusion

In the exploration, the efficacy of a Dijkstra-based distance measure is assessed for conceptual clustering across multiple categorical datasets. This distance measures demonstrated a powerful capability in determining hierarchical relationships among categorical variables, even within complex and dense datasets. The evaluations, conducted across randomly generated formal contexts and real-world datasets, confirmed its robust performance, scalability, and reliability. However, a correlation between the average runtime and the number of concepts suggests potential efficiency enhancements.

The clustering tasks employed two methods: the K-means Dijkstra on Lattice (KDL) method, which uses Formal Concept Analysis (FCA) and the Dijkstra-based distance measure; and the K-means Vector on Lattice (KVL) method, which transforms categorical data into numerical vectors and applies standard k-means algorithms. The KDL method yielded high-quality clusters that accurately mirrored the inherent hierarchical relationships within categorical data. However, when handling larger numbers of clusters or formal concepts, scalability emerged as a challenge for this method. On the other hand, the KVL method demonstrated impressive scalability. Nevertheless, due to its conversion of data into numerical vectors, there's a risk of overlooking the hierarchical structure of the data, which could affect the clustering quality.

Future research has several promising pathways. The lattice structure in the KDL method could be simplified to boost scalability, and the KVL method could be further refined to better capture the structure of categorical data. Additionally, the exploration of alternate or complementary distance measures could be beneficial. A particularly intriguing direction for future research is integrating the Dijkstra-based distance measure into the k-means algorithm, which could significantly advance categorical data analysis. The study of the KDL and KVL methods has under-scored their respective strengths and limitations, illuminating potential areas for future research. These findings are instrumental to the ongoing development of categorical data analysis and refining data clustering methodologies. By investigating the complexities of concept lattices and streamlining the knowledge discovery process of FCA, The study offers a foundational understanding that serves as a basis for the development of more scalable and efficient solutions.

Acknowledgment

The authors express sincere gratitude to the Department of Information Technology at the József Hatvany Doctoral School, University of Miskolc, Hungary, for the necessary support and resources for this study.

REFERENCES

[1] V. Ganti, J. Gehrke, and R. Ramakrishnan, "CACTUS—clustering categorical data using summaries," in Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, 1999, pp. 73–83. https://doi.org/10.1145/312129.312201.

[2] S. Guha, R. Rastogi, and K. Shim, "Rock: A robust clustering algorithm for categorical attributes," Inf Syst, vol. 25, no. 5, pp. 345–366, Jul. 2000, https://doi.org/10.1016/S0306-4379(00)00022-3.

[3] Z. Huang, "Extensions to the k-means algorithm for clustering large data sets with categorical values," Data Min Knowl Discov, vol. 2, no. 3, pp. 283–304, 1998. https://doi.org/10.1023/A:1009769707641.

[4] Z. Huang, "Clustering large data sets with mixed numeric and categorical values," in Proceedings of the 1st pacific-asia conference on knowledge discovery and data mining,(PAKDD), Citeseer, 1997, pp. 21–34. https://doi.org/10.4236/ojs.2017.72013.

[5] D. Ienco, R. G. Pensa, and R. Meo, "Context-based distance learning for categorical data clustering," in Advances in Intelligent Data Analysis VIII: 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31-September 2, 2009. Proceedings 8, Springer, 2009, pp. 83–94. https://doi.org/10.1007/978-3-642-03915-7_8.

[6] J. MacQueen, "Classification and analysis of multivariate observations," in 5th Berkeley Symp. Math. Statist. Probability, University of California Los Angeles LA USA, 1967, pp. 281–297. https://doi.org/10.4236/ojpp.2015.56041.

[7] O. M. San, V.-N. Huynh, and Y. Nakamori, "An alternative extension of the k-means algorithm for clustering categorical data," International journal of applied mathematics and computer science, vol. 14, no. 2, pp. 241–247, 2004.

[8] L. Chen and S. Wang, "Central clustering of categorical data with automated feature weighting," in Twenty-Third International Joint Conference on Artificial Intelligence, Citeseer, 2013.

[9] M. Ng, M. Li, J. Huang, and Z. He, "On the Impact of Dissimilarity Measure in k-Modes Clustering Algorithm," IEEE Trans Pattern Anal Mach Intell, vol. 29, no. 3, pp. 503–507, Mar. 2007. https://doi.org/ 10.1109/TPAMI.2007.53.

[10] R. Wille, "Restructuring lattices theory: an approach on hierarchies of concepts." Dordrecht, Holland: Springer, 1982.

[11] R. Ganter and R. Wille, "Formal concept analysis: Mathematical foundations Springer-Verlag Berlin Germany," 1999. https://doi.org/10.1007/978-3-642-59830-2.

[12] K. Sumangali and C. A. Kumar, "A comprehensive overview on the foundations of formal concept analysis," Knowledge Management & E-Learning: An International Journal, vol. 9, no. 4, pp. 512–538, 2017, https://doi.org/10.34105/j.kmel.2017.09.032.

[13] M. Alwersh and L. Kovács, "Survey on attribute and concept reduction methods in formal concept analysis," Indonesian Journal of Electrical Engineering and Computer Science, vol. 30, no. 1, pp. 366–387, Apr. 2023, https://doi.org/10.11591/ijeecs.v30.i1.pp366-387.

[14] T. Abiy, H. Pang, C. Williams, J. Khim, and E. Ross, "Dijkstra's shortest path algorithm," Retrieved from, 2016.

[15] F. Mukhlif and A. Saif, "Comparative study on Bellman-Ford and Dijkstra algorithms," in Int. Conf. Comm. Electric Comp. Net, 2020.

[16] R. Bellman, "On a routing problem," Q Appl Math, vol. 16, no. 1, pp. 87–90, 1958. https://doi.org/10.1090/qam/102435

[17] R. W. Floyd, "Algorithm 97: shortest path," Commun ACM, vol. 5, no. 6, p. 345, 1962. http://dx.doi.org/10.1145/367766.368168.

[18] M. Tropmann-Frick, "Analysis of the Shortest Path Method Application in Social Networks," 2023. https://doi.org/10.3233/FAIA220500.

[19] T.-H. T. Nguyen and V.-N. Huynh, "A k-means-like algorithm for clustering categorical data using an information theoretic-based dissimilarity measure," in Foundations of Information and Knowledge Systems: 9th International Symposium, FoIKS 2016, Linz, Austria, March 7-11, 2016. Proceedings 9, Springer, 2016, pp. 115–130. https://doi.org/10.1007/978-3-319-30024-5_7.

[20] Z. Huang and M. K. Ng, "A fuzzy k-modes algorithm for clustering categorical data," IEEE transactions on Fuzzy Systems, vol. 7, no. 4, pp. 446–452, 1999, http://dx.doi.org/10.1109/91.784206.

[21] F. Cao, J. Liang, D. Li, L. Bai, and C. Dang, "A dissimilarity measure for the k-Modes clustering algorithm," Knowl Based Syst, vol. 26, pp. 120–127, 2012, doi: https://doi.org/10.1016/j.knosys.2011.07.011.

[22] M. Li, S. Deng, L. Wang, S. Feng, and J. Fan, "Hierarchical clustering algorithm for categorical data using a probabilistic rough set model," Knowl Based Syst, vol. 65, pp. 60–71, 2014, doi: https://doi.org/10.1016/j.knosys.2014.04.008.

[23] Bernhard Ganter, "Two basic algorithms in concept analysis. FB4-Preprint No 831, 1984.". https://doi.org/10.1007/978-3-642-11928-6_22.

[24] J. Baixeries, L. Szathmary, P. Valtchev, and R. Godin, "Yet a faster algorithm for building the Hasse diagram of a concept lattice," in Formal Concept Analysis: 7th International Conference, ICFCA 2009 Darmstadt, Germany, May 21-24, 2009 Proceedings 7, Springer, 2009, pp. 162–177. https://doi.org/10.1007/978-3-642-01815-2_13.

[25] D. Schütt, "Abschätzungen für die Anzahl der Begriffe von Kontexten," Master's Thesis, TH Darmstadt, 1987.

[26] L. Kovács, "Efficiency analsyis of concept lattice construction algorithms," Procedia Manuf, vol. 22, pp. 11–18, 2018, https://doi.org/10.1016/j.promfg.2018.03.003.